Chapter 2

# A CLASSIFICATION SCHEME FOR ROBOTIC CELLS AND NOTATION

In this chapter, we discuss a classification scheme for sequencing and scheduling problems in robotic cells and provide notation. As in the classification scheme for classical scheduling problems (Graham et al. [69]), we distinguish problems based on three characteristics: machine environment ($\alpha$), processing characteristics ($\beta$), and objective function ($\gamma$). A problem is then represented by the form $\alpha|\beta|\gamma$. Following the discussion of these characteristics, we detail the classification in Section 2.4 and provide a pictorial representation in Figure 2.2. Finally, we discuss relevant cell data whose values influence a cell's performance and define some basic notation for subsequent use.

## 2.1 Machine Environment

We start by describing characteristics that are represented in the first field of the classification scheme.

### 2.1.1 Number of Machines

If each processing stage has only one machine, the robotic cell is called a *simple robotic cell* or a *robotic flowshop.* Such a cell contrasts with a *robotic cell with parallel machines*, in which at least one processing stage has two or more identical machines. Cells with parallel machines are discussed in Chapter 5.

A typical simple robotic cell contains $m$ processing machines: $M_1$, $M_2$, $\ldots$, $M_m$. Let $M = \{1, 2, \ldots, m\}$ be the set of indices of these machines.

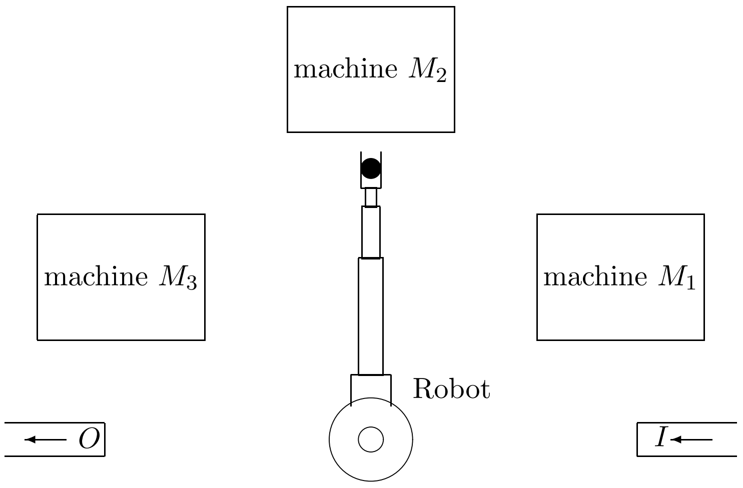


*Figure 2.1.* A Three-Machine Simple Robotic Cell.

The robot obtains a part from the input device ($I$, also denoted as $M_0$), carries the part to the first machine ($M_1$), and loads the part. After $M_1$ completes its processing on the part, the robot unloads the part from $M_1$ and transports it to $M_2$, on which it loads the part. This pattern continues for machines $M_3, M_4, \ldots, M_m$. After the last machine $M_m$ has completed its processing on the part, the robot unloads the part and carries it to the output device ($O$, also denoted as $M_{m+1}$). In some implementations, the input device and the output device are at the same location, and this unit is called a *load lock*. A three-machine simple robotic cell is depicted in Figure 2.1.

This description should not be misconstrued as implying that the robot remains with each part throughout its processing by each machine. Often, after loading a part onto a machine, the robot moves to another machine or to the input device to collect another part to transport to its next destination. Determining which sequence of such moves maximizes the throughput of the cell has been the focus of the majority of research on robotic cell sequencing and scheduling.

### 2.1.2 Number of Robots

Manufacturers employ additional robots in a cell in order to increase throughput by increasing the material handling capacity. Cells with one (resp., more than one) robot are called *single-robot* (resp., *multiple-robot*) *cells.* Most studies in the literature analyze single-robot cells. Multiple-robot cells are discussed in Chapter 8.

### 2.1.3 Types of Robots

A single-gripper robot can hold only one part at a time. In contrast, a dual-gripper robot can hold two parts simultaneously. In a typical use of this capability, the robot holds one part while the other gripper is empty; the empty gripper unloads a machine, the robot repositions the second gripper, and it loads that machine. Dual-gripper robots are discussed in Chapter 4.

In a single-gripper simple robotic cell, the robot cannot unload a part from machine $M_i, i = 0, ..., m-1$, unless the next machine $M_{i+1}$ is empty. This condition is commonly referred to as a *blocking* condition.

### 2.1.4 Cell Layout

The layout refers to the arrangement of machines within the cell. Most robotic cell models assume one of two layouts: *linear* or *circular.* A semicircular arrangement of machines has also been referred to in the literature. However, all our results for a linear layout (Figure 1.3) remain valid for a semicircular layout (Figure 1.1). Unless specified otherwise, we assume a linear/semicircular layout. Cells employing a circular layout (Figure 1.5) are discussed in Chapters 4 and 7.

## 2.2 Processing Characteristics

Four different processing characteristics are specified in the second field. We describe three in this section. The fourth, called the production strategy, is detailed in Section 2.4.

### 2.2.1 Pickup Criterion

Most of the discussion in this book concerns robotic cells with no buffers for intermediate storage. For such cells, all parts must be either in the input device, on one of the machines, in the output device, or with the robot.

Robotic cells can be partitioned into three types – *free pickup, no-wait,* and *interval* – based on the pickup criterion. For all three types, a part that has completed processing on $M_i$ cannot be loaded onto $M_{i+1}$

for its next processing unless $M_{i+1}$ is unoccupied, $i = 0, \ldots, m$. In free-pickup cells, this is the only pickup restriction; there is no limit on the amount of time a part that has completed processing on a machine can remain on that machine.

For the more restrictive no-wait cells, a part must be removed from machine $M_i$, $i \in M$, and transferred to machine $M_{i+1}$ as soon as $M_i$ completes processing that part. Such conditions are commonly seen in steel manufacturing or plastic molding, where the raw material must maintain a certain temperature, or in food canning to ensure freshness (Hall and Sriskandarajah [77]). Results for no-wait cells are discussed in Chapter 9.

In interval robotic cells, each stage has a specific interval of time – a processing time window – for which a part can be processed at that stage. Thus, if $[a_i, b_i]$ is the processing time window at stage $i, i \in M$, then a part must be processed for $a_i$ time units on stage $i$, and must be transferred to stage $(i+1)$ within $(b_i - a_i)$ time units after its completion of processing on stage $i$. This is applicable, for example, for the hoist scheduling problem on an electroplating line (Che et al. [30], Chen et al. [33], Lei and Wang [108]): printed circuit boards are placed in a series of tanks with different solvents. Each tank has a specific interval of time for which a card can remain immersed. Interval cells are discussed in Chapter 9.

Unless specified otherwise, the cells we discuss in the chapters that follow have the free-pickup criterion.

### 2.2.2 Travel-Time Metric

The robot's travel time between machines greatly influences a cell's performance. One common model often applies when the machines are arranged in numeric order in a line (Figure 1.3) or semicircle (Figure 1.1). The robot's travel time between adjacent machines $M_{i-1}$ and $M_i$, denoted $d(M_{i-1}, M_i)$, equals $\delta$, for $i = 1, \ldots, m+1$, and is *additive*. That is, the travel time between any two machines $M_i, M_j, 0 \leq i, j \leq m+1$ is $d(M_i, M_j) = |i - j|\delta$. This scheme is easily generalized to the case of unequal travel times between adjacent machines (Brauner and Finke [20]): $d(M_{i-1}, M_i) = \delta_i$, $i = 1, \ldots, m+1$, and $d(M_i, M_j) = \sum_{k=i+1}^{j} \delta_k$, for $i < j$. If $d(M_{i-1}, M_i) = \delta$, $i = 1, \ldots, m+1$, then we call the travel-time metric *regular additive*. If $d(M_{i-1}, M_i) = \delta_i$, $i = 1, \ldots, m+1$, then the cell has *general additive* travel times.

There are also additive travel-time cells in which the machines are arranged in a circle so that $I$ and $O$ are adjacent or in the same location (Drobouchevitch et al. [50], Geismar et al. [61], Sethi et al. [143], Sriskandarajah et al. [146]). In these cells, the robot may travel in either direction to move from one machine to another; e.g., to move from $M_1$ to $M_{m-1}$, it may be faster to go via $I$, $O$, and $M_m$, than to go via $M_2, M_3, \ldots, M_{m-2}$. For circular cells with regular additive travel times, $d(M_i, M_j) = \min\{|i-j|\delta, (m+2-|i-j|)\delta\}$. For general additive travel-time cells, $d(M_i, M_j) = \min\{\sum_{k=i+1}^{j} \delta_k, \sum_{k=1}^{i} \delta_k + \delta_{0,m+1} + \sum_{k=j+1}^{m+1} \delta_k\}$ for $i < j$. Most studies assume that the travel times are symmetric, i.e., $d(M_i, M_j) = d(M_j, M_i), 0 \leq i, j \leq m+1$, and that the travel time between any two machines does not depend on whether or not the robot is carrying a part.

To make this model better represent reality, it can be enhanced to account for the robot's acceleration and deceleration (Logendran and Sriskandarajah [116]). The travel times between adjacent machines do not change. However, the travel time between nonadjacent machines is reduced. For each intervening machine, the robot is assumed to save $\eta$ units of time. Therefore, for $0 \leq i, j \leq m+1$, if $d(M_{i-1}, M_i) = \delta_i$, then

$$d(M_i, M_j) = \sum_{k=\min(i,j)+1}^{\max(i,j)} \delta_k - (|i-j| - 1)\eta.$$

We use this model in our discussions in Chapter 6.

For certain cells, additive travel times are not appropriate. Dawande et al. [47] discuss a type of cell for which the robot travel time between *any* pair of machines is a constant $\delta$, i.e., $d(M_i, M_j) = \delta, 0 \leq i, j \leq m+1$, $i \neq j$. This arises because these cells are compact and the robots move with varying acceleration and deceleration between pairs of machines.

The most general model, one that can represent all the travel-time metrics typically encountered in practice, assigns a value $\delta_{ij}$ for the robot travel time between machines $M_i$ and $M_j, 0 \leq i, j \leq m+1$. These travel times are, in general, neither additive nor constant. Brauner et al. [24] address this problem by making three assumptions that conform to basic properties of Euclidean space:

1. The travel time from a machine to itself is zero, that is, $\delta_{ii} = 0, \forall i$.
2. The travel times satisfy the triangle inequality, that is, $\delta_{ij} + \delta_{jk} \geq \delta_{ik}, \forall i, j, k$.
3. The travel times are symmetric, that is, $\delta_{ij} = \delta_{ji}, \forall i, j$.

A robotic cell that satisfies Assumptions 1 and 2 is called a *Euclidean robotic cell*, and one that satisfies Assumptions 1, 2, and 3 is called a *Euclidean symmetric robotic cell*. As we shall discuss in Chapter 3, the robot move sequencing problem for either case is strongly NP-hard (Brauner et al. [24]). This is also why most studies approximate reality with additive or constant travel-time models, depending on which of the two is a better fit.

To summarize, three different robot travel-time metrics have been addressed in the literature: additive, constant, and Euclidean. Most studies assume one of these. Therefore, many results in the field have been proven only for one travel-time metric rather than for all three.

### 2.2.3 Number of Part-Types

A cell producing identical parts is referred to as a *single-part-type cell*. In contrast, a *multiple-part-type cell* processes lots that contain different types of parts. Generally, these different part types require different processing times on a given machine. Multiple-part-type cells are discussed in Chapters 6 and 7. Throughout the rest of the book, unless specified otherwise, the cell under consideration processes identical parts.

## 2.3 Objective Function

From an optimization aspect, the objective that is predominantly addressed in the literature is that of maximizing the *throughput* – the long-term average number of completed parts placed into the output buffer per unit time. This will be our objective throughout the book. A precise definition of throughput is provided in Chapter 3.

## 2.4 An $\alpha|\beta|\gamma$ Classification for Robotic Cells

Figure 2.2 is a pictorial representation of the classification discussed in the preceding text. A problem is represented using the form $\alpha|\beta|\gamma$, where

(a) $\alpha = RF^{g,l}_{m,r,\bar{b}}(m_1, ..., m_m)$. Here, $RF$ stands for "Robotic Flowshop," $m$ is the number of processing stages, and the vector $(m_1, m_2, ..., m_m)$ indicates the number of identical machines at each stage. When this vector is not specified, $m_i = 1, i = 1, ..., m$, and the cell is a simple cell. The second subscript $r$ denotes the number of robots; when not specified, $r = 1$. For cells with output buffers at the various stages of the cell, the vector $\bar{b} = (b_1, ..., b_m)$ denotes the sizes of the buffers. At stage $i$, the size of the output buffer is denoted by $b_i, i = 1, ..., m$; this notation is omitted for cells without buffers. The first superscript $g$ denotes the type of robot used. For example, $g = 1$ (resp., $g = 2$) denotes a single-gripper (resp., dual-gripper) cell. If $g$ is not specified, then $g = 1$. The second superscript $l$ indicates the layout of the cell; a linear/semicircular (resp., circular) layout is indicated by $\sqcup$ (resp., $\circ$). Most of our discussion is for linear or semicircular layouts; unless specified otherwise, such a layout is assumed, and the notation is omitted.

(b) $\beta$ =(*pickup, travel-metric, part-type, prod-strategy)*, where

- *pickup* $\in$ {*free, no-wait, interval*} specifies the pickup criterion.
- *travel-metric* $\in$ {*A, C, E*} specifies the travel-time metric, where *A*, *C*, and *E* denote the additive, the constant, and the Euclidean travel-time metric, respectively.
- If *part-type* is not specified, the cell produces a single part-type. Otherwise, *part-type* = *MP* denotes a cell producing multiple part-types.
- *prod-strategy* $\in$ {*cyclic-k, LCM, all, CRM*} denotes the specific production strategy employed. The detailed descriptions of these strategies appear in later chapters, so we limit our description here and refer the reader to the corresponding chapter.
  (i) In a cell producing either a single part-type or multiple part-types, *cyclic-k* refers to a *cyclic production* strategy wherein exactly $k$ units are produced per cycle (Chapter 3). When the integer $k$ is not specified, the production strategy includes all $k$-unit cycles, $k \geq 1$. *LCM cycles* form a subclass of cyclic solutions, and are discussed in Chapters 5 and 8.

(ii) In a cell producing either a single part-type or multiple part-types, *all* refers to a production environment where all production strategies (i.e., cyclic as well as noncyclic) are considered (Chapters 3 and 10).

(iii) In robotic cells producing multiple part-types (Chapters 6 and 7), *CRM* refers to the *concatenated robot-move sequence* strategy.

(c) $\gamma = \mu$ denotes the objective function of maximizing the throughput. Although this is the only objective function addressed in our discussion, we use a separate field to allow for future work involving different objective functions.

We now illustrate our classification with a few examples.

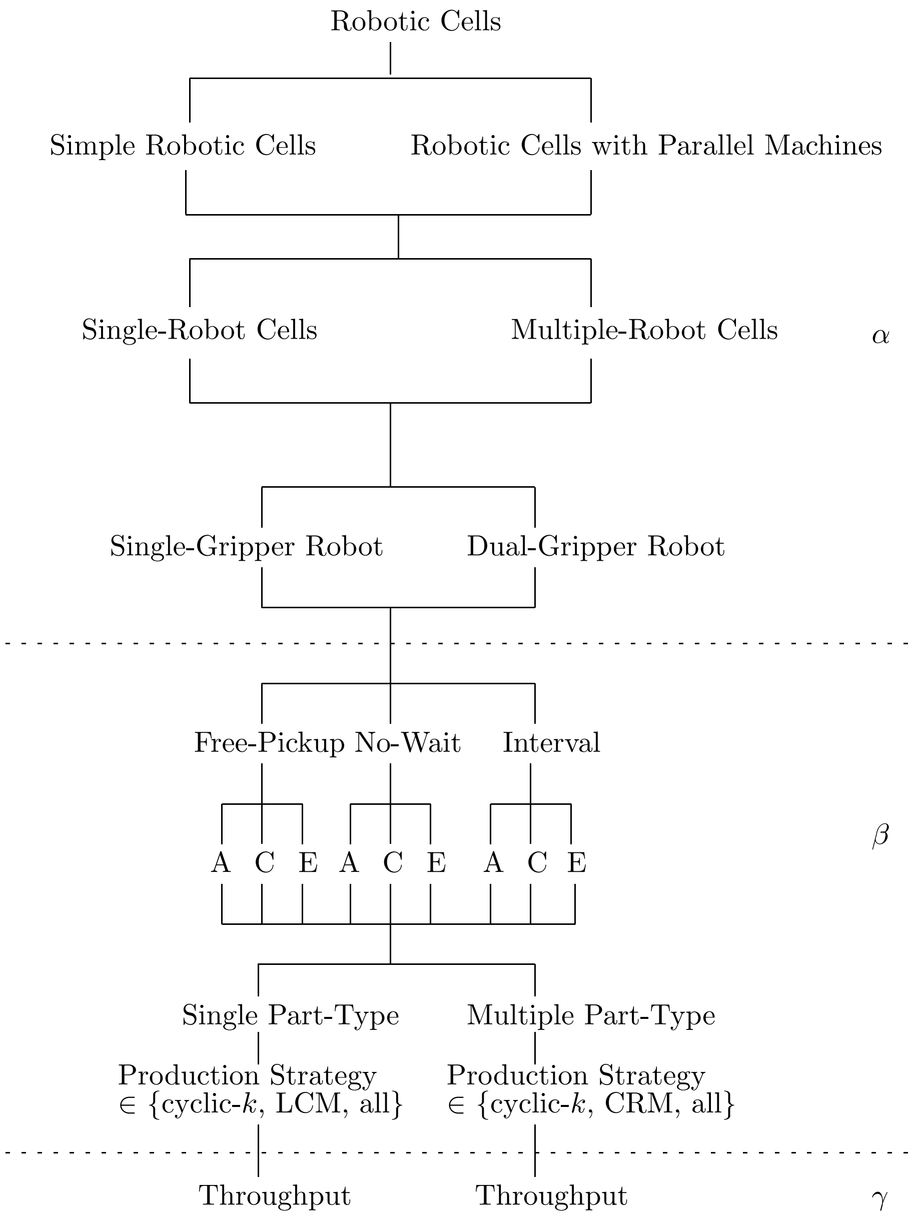
1. $RF_4|(\textit{free,A,cyclic-1})|\mu$ represents a four-machine simple robotic cell with one single-gripper robot, a free-pickup criterion, and additive travel-time metric. It produces a single part-type and operates a cyclic production strategy wherein one unit is produced per cycle. The objective function is that of maximizing the throughput.

2. $RF_5(1,4,2,3,2)|(\textit{no-wait,E,cyclic-2})|\mu$ refers to the problem of maximizing throughput for a five-stage robotic cell with parallel machines that has one, four, two, three, and two machines, respectively, in stages 1, 2, 3, 4, and 5. The cell produces a single part-type, has one single-gripper robot, employs a no-wait pickup criterion and a Euclidean travel-time metric, and produces two units per cycle.

3. $RF^2_{m,3}|(\textit{interval,C,MP,CRM})|\mu$ considers the problem of throughput maximization in an $m$-machine simple robotic cell with three dual-gripper robots, an interval pickup criterion, constant travel-time metric, and multiple-part-type production using a CRM production strategy.

4. $RF^{2,\circ}_{m,\bar{1}}|(\textit{free,A,cyclic-k})|\mu$ is the problem of maximizing the throughput over all cyclic schedules in an $m$-machine dual-gripper cell with an output buffer of size one at each machine. The travel-time metric is additive, and the layout of the cell is circular.

In the chapters that follow, we use this classification to specify the problem under consideration.



A, C, E denote Additive, Constant, Euclidean Travel Time, respectively

*Figure 2.2.* A Classification of Robotic Cells

## 2.5 Cell Data

In addition to the robot's travel-time metric, the processing times at the various stages and the times required for loading and unloading a machine influence the cell's throughput. We now discuss these characteristics and the notation for representing the actions of the robot and the states of the cell. First, we list the basic assumptions throughout most studies:

- All data and events are deterministic.
- All processing is nonpreemptive.
- Parts to be processed are always available at the cell's input device.
- There is always space for completed parts at the output device.
- All data are rational.

### 2.5.1 Processing Times

Since each of the $m$ stages performs a different function, each, in general, has a different processing time for a given part. For cells with free pickup or no-wait pickup, the processing time of a machine in stage $j$ is denoted by $p_j, j \in M$. If a cell processes $k$ different types of parts, the processing time of part $i$ at stage $j$ is denoted by $p_{ij}, i = 1, \ldots, k; j \in M$. In interval robotic cells, the processing time of machine $M_j$ is specified by a lower bound $l_j$ and an upper bound $u_j \geq l_j$. For example, the time that a printed circuit board spends in tank $j$ must be in the interval $[l_j, u_j]$. If multiple part-types are processed in an interval robotic cell, the processing interval for part-type $i$ is denoted by $[l_{ij}, u_{ij}]$.

### 2.5.2 Loading and Unloading Times

Another factor that influences the processing duration for a part is the time required for loading and unloading at each machine. For uniformity, picking a part from $I$ is referred to as unloading $I$, and dropping a part at $O$ is referred to as loading $O$. Typically, models assume that the loading and unloading times are equal ($\epsilon$) for all machines. This will be our assumption as well for most of the discussion. More sophisticated models have different values for loading and unloading at each machine:

the loading (resp., unloading) time for $M_i$ is $\epsilon_{2i}$ (resp., $\epsilon_{2i+1}$), $i = 1, ..., m$; the unloading (resp., loading) time at $I$ (resp., $O$) is $\epsilon_1$ (resp., $\epsilon_{2(m+1)}$). We use this more general notation in Chapter 6.

### 2.5.3 Notations for Cell States and Robot Actions

By the state of the robotic cell at any given instant of time, we mean a sufficient description of the cell required for the purpose of our analysis. To keep the notation simple, our discussion in this section is limited to simple robotic cells with the free-pickup criterion; appropriate enhancements can be made for other classes of cells.

Ideally, a precise mathematical description of the state of the cell would include the following.

- The occupancy state of each machine. That is, whether a machine contains a part or it is empty.
- If a machine contains a part, then the time remaining on its current processing.
- The location of the robot.
- The occupancy state of the robot, that is, whether the robot arm has a part or not.

Before we formalize the state space, note that since we are interested in maximizing the throughput of the cell, it is not necessary to consider "wasteful" robot actions such as unnecessary waiting at a location or moving to a location without performing at least one of the loading or unloading operations. Also, since this is a deterministic problem, it is sufficient to define decisions regarding the robot's moves only at those epochs when the robot has just finished loading or unloading a part at a machine. It follows that it is sufficient to consider the state when the robot's position is at these epochs.[1]

Our focus in this book is on a steady-state analysis of a certain class of solutions referred to as cyclic solutions (discussed in Chapter 3). Typ-

[1]In the stochastic setting, say when the processing times are random variables, a throughput maximizing operation may require the robot arm to change its traversal path while the robot is in transition, at a time when some new information becomes available. To allow for this, a continuous state space and continuous decision making over time are required.

ically, this analysis does not require a detailed state description since the definition of cyclic solutions involves a requirement that completes the missing information. In view of this, all that is needed is a specification of each machine in terms of whether it is occupied or not. Such a simplified description can be presented as an $(m+1)$-dimensional vector $(e_1, \ldots, e_{m+1})$ (Sethi et al. [142]). Each of the first $m$ dimensions corresponds to a machine: $e_i = \emptyset$ if $M_i$ is unoccupied; $e_i = \Omega$ if $M_i$ is occupied, $i = 1, \ldots, m$. The last dimension represents the robot; $e_{m+1} = M_i^-$ indicates that the robot has just completed loading a part onto $M_i, i = 1, \ldots, m+1$, and $e_{m+1} = M_i^+$ indicates that the robot has just completed unloading a part from $M_i, i = 0, \ldots, m$.

EXAMPLE 2.1 For $m = 4$, consider the state $(\emptyset, \Omega, \emptyset, \Omega, M_2^-)$: $M_1$ and $M_3$ are unoccupied, $M_2$ and $M_4$ are occupied, and the robot has just completed loading $M_2$. Suppose that the robot's next actions were to travel to $I$, unload a part from $I$, travel to $M_1$, and load that part onto $M_1$. The states corresponding to these actions are $(\emptyset, \Omega, \emptyset, \Omega, M_0^+)$ and $(\Omega, \Omega, \emptyset, \Omega, M_1^-)$. Note that listing the state $(\emptyset, \Omega, \emptyset, \Omega, M_0^+)$ is superfluous. To transition from $(\emptyset, \Omega, \emptyset, \Omega, M_2^-)$ into $(\Omega, \Omega, \emptyset, \Omega, M_1^-)$, the robot must have first traveled to $I$.

In general, a series of robot actions can be completely represented by a string of $M_i^-$ symbols. For example, $M_2^- M_4^- M_5^-$ means that the robot unloads a part from $M_1$, travels to $M_2$, and loads the part onto $M_2$. The robot next travels to $M_3$, waits for $M_3$ to finish processing (if required), unloads a part from $M_3$, travels to $M_4$, and loads the part onto $M_4$. The robot waits at $M_4$ while the part is being processed. The robot then unloads the part from $M_4$, carries it to $M_5$, and loads $M_5$.

A different notation has largely supplanted the $M_i^-$ notation in the literature. This more popular notation is based on the concept of an *activity*. Activity $A_i$ consists of the following sequence of actions:

- The robot unloads a part from $M_i$.
- The robot travels from $M_i$ to $M_{i+1}$.
- The robot loads this part onto $M_{i+1}$.

The sequence of actions discussed above ($M_2^- M_4^- M_5^-$) would be represented by $A_1 A_3 A_4$. Since a part must be processed on all $m$ machines and then placed into the output buffer, one instance of each of the $m+1$ activities $A_0, A_1, \ldots, A_m$, is required to produce a part.

It is easy to use the activity-based notation to represent the cell's current status. Let $e_{m+1} = A_i$ indicate that the robot has just completed activity $A_i$; $e_i, i = 1, 2, \ldots, m$, will have the same meaning as before.

Example 2.2 For $m = 4$, an example state is $(\Omega, \emptyset, \emptyset, \Omega, A_3)$: $M_2$ and $M_3$ are unoccupied, $M_1$ and $M_4$ are occupied, and the robot has just completed loading $M_4$. From this point, let us now consider what happens if the robot executes activity sequence $A_1 A_2 A_4$: the robot moves to $M_1$, waits (if required) for $M_1$ to finish processing, unloads a part from $M_1$, travels to $M_2$, and loads the part onto $M_2$. At this instant, the state of the cell is $(\emptyset, \Omega, \emptyset, \Omega, A_1)$. The robot waits at $M_2$ for the entirety of the part's processing. The robot then unloads the part from $M_2$, carries it to $M_3$, and loads the part onto $M_3$. The cell's state is now $(\emptyset, \emptyset, \Omega, \Omega, A_2)$. The robot next travels to $M_4$, waits (if required) for $M_4$ to finish processing, unloads a part from $M_4$, travels to the output buffer, and loads the part onto the output buffer, so the cell's state is $(\emptyset, \emptyset, \Omega, \emptyset, A_4)$.

For most of the discussion in this book, we will represent robot actions by using the activity notation: $A_i, i = 0, 1, ..., m$. The discussion for robotic cells producing multiple part-types, however, is easier with the $M_i^-$ notation; we will use it in Chapters 4 and 6. The $M_i^-$ notation is also convenient for describing moves in a dual-gripper robotic cell (Chapter 4).

The simplified state description above omits information representing the extent of the processing completed on the parts on the various machines. A more precise representation of a state is an $(m+1)$-tuple $\Gamma = (s_1, \ldots, s_{m+1})$, where $s_i \in \{-1, r_i\}, i \in M$. If $s_i = -1$, machine $M_i$ has no part on it; otherwise $r_i$ is the time remaining in the processing of the current part on $M_i$. As before, $s_{m+1} \in \{A_i, i = 0, ..., m\}$ denotes that the robot has just completed activity $A_i$ (i.e., loaded a part onto machine $M_{i+1}$).

Example 2.3 For $m = 4$, the state vector $\Gamma = (5, 0, -1, p_4, A_3)$ indicates that the part on $M_1$ has five time units of processing remaining, $M_2$ has completed processing a part and that part still resides on $M_2$, and $M_3$ is empty. The robot has unloaded a part from $M_3$, carried it to $M_4$, and just completed loading it onto $M_4$.

There is another important observation to be made here. Note that even with integer data, the remaining processing times are in general real numbers. However, since we need to consider the system state only at the epochs mentioned above, the state description will be integral provided the initial state of the system is restricted to be in integer terms. This restriction can be imposed without loss of generality since some initial adjustments can be made at the beginning to bring the state to integral terms, and the time taken to make these adjustments is of no consequence in the context of the long-term average throughput criterion. Thus, in any state description $\Gamma = (s_1, \ldots, s_{m+1})$, $s_i \in \{-1, r_i\}$ with $r_i \in \{k \in \mathbb{Z} : 0 \leq k \leq p_i\}$, $i \in M$. We thus have a *finite-state* dynamic system.