2004

CONTROL SYNTHESIS IN COLORED CONDITION SYSTEMS

Praveen Mandavilli
University of Kentucky, pmandavilli@gmail.com

ABSTRACT OF THESIS

CONTROL SYNTHESIS IN COLORED CONDITION SYSTEMS

With complex systems, monolithic models become impractical and it becomes necessary to model them through subsystems and components. Unless these components and subsystems are structured, exploiting them in a methodical manner to develop a control logic for them also becomes complex. In the previous research, to characterize the input/output behavior of discrete state interacting, systems a condition language framework was defined and algorithms that can automatically generate a controller given the system model and the desired specification using this framework were presented. Though this framework and the control algorithms are ideally suited to simple systems, representation of components with large state spaces requires a more refined approach. In this thesis, we present the modelling framework namely 'Color condition systems', that compactly represent components with large state spaces. We also present algorithms that can automatically generate a controller that consists of a set of action type taskblocks, given the system model and the desired specification described using color condition systems. The modelling framework and the working of the algorithms are illustrated using figures and comments on the possible ways of optimizing the algorithms are also quoted. Finally, in the appendix, we also present the approach that can be taken to implement a few parts of the algorithm.

KEYWORDS: Color Condition systems, Actionblock, Controller, Modeling Framework, Large state space.

Praveen Mandavilli

November 05' 2004

CONTROL SYNTHESIS IN COLOR CONDITION
SYSTEMS

By

Praveen Mandavilli

Dr. Lawrence Emory Holloway.

Director of Thesis.

Dr. Ibrahim Jawahir.

Director of Graduate Studies.

RULES FOR THE USE OF THESES

THESIS

Praveen Mandavilli

The Graduate School

University of Kentucky

2004

CONTROL SYNTHESIS IN COLORED CONDITION SYSTEMS

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree
of Master of Science in Manufacturing Systems Engineering
in the College of Engineering
at the University of Kentucky

By
Praveen Mandavilli

Lexington, Kentucky

Director: Dr. Lawrence E. Holloway,
Professor of Electrical & Computer Engineering,
University of Kentucky,

Lexington, Kentucky

2004

MASTER'S THESIS RELEASE

I authorize the University of Kentucky Libraries to reproduce this thesis in whole or in part for the purpose of research

Signed:__________________________
Date:__________________________

DEDICATION

This thesis would be unfinished with out the mention of the blessings showered by my beloved family on me and the support & encouragement given to me by my cherished professor Dr. Holloway, to whom this thesis is dedicated.

I dedicate this thesis to my parents who have provided me with support emotionally and financially throughout this long journey called college career. Without my professor lifting me up when this thesis seemed interminable, I doubt it should ever have been completed.

ACKNOWLEDGMENTS

No work is ever created alone and no research is ever carried out in solitude. I am greatly indebted to a number of people without whom this thesis might not have been achievable. This thesis is the product of the gentle, encouraging support of my mentor, academic advisor, Dr. Lawrence E. Holloway, who always had answered all my questions and challenged my thinking.

This thesis is written with good support from my friend and research colleague Prashanth Thumu who made the work more enjoyable. I also appreciate the help provided by my friends at university of kentucky.

However there are those whose spiritual support is even more important. Thanks goes to my family back in India. Through every stage I have my full stream support from my parents. There are no words that adequately express my appreciation and gratitude to them.

I would also thank my committee members Dr. Janet Lumpp and Dr. Jawahir for offering suggestions for improvement.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction and Motivation

1.1 Introduction and Motivation

Automated control code synthesis methods for automated manufacturing systems reduce the code creation time, minimize the debug time and simplify the maintenance of the code [Holl00]. Considerable research has been done in this area. Condition systems, a class of condition-event models were used as the modelling framework. The approach has been based on viewing the system as a collection of simple subsystems that interact through "condition signals". The condition systems offer many advantages, allowing the modeler to represent the systems as a set of components. The modelling framework is ideally suited for models of interacting components as long as the systems remain simple with relatively small state spaces.

However, a research issue that must be addressed for complex systems is how to effectively handle subsystems and components with potentially large number of states. Representation of such components using the condition systems modelling framework developed in the prior research becomes almost impractical. As the state space associated with the subsystem increases, it becomes hard to describe the model and at the same time the model becomes awful and unreadable. This problem is typical for many practical applications of condition systems. Hence it becomes necessary to refine the modelling framework in order to compactly represent components with large state spaces. One of the promising directions to tackle this problem of “state space” is to extend the techniques developed in the prior research to high-level models. Colored condition systems framework, developed in the current thesis, allow the modeler to represent complex systems with large state spaces compactly. The goal is to define the modelling framework and TASK CONTROL SYNTHESIS: translating a high-level specification into detailed sequences of control and actuation signals that will accomplish the specified behavior.

This thesis is divided into four parts. Chapter 2 describes the condition systems modelling framework. It gives an overview of the definition of condition systems and briefly describes the logic behind the algorithms used to build the controller that drives the system to a state that would output the target condition set. Chapter 3 defines precisely the modelling framework namely ‘color condition systems’ used to model the systems which have large state spaces associated with them. It introduces the concept of color of a token (hence called color condition systems), the notion of intervals and condition matrices and demonstrates the framework with examples and figures. Chapter 4 introduces linear temporal logic syntax used in this thesis and explains what makes a taskblock to work effectively. The taskblock generated in this thesis is an action type taskblock and is designed to perform a specified action which would lead the system to the target state. Chapter 5 presents the algorithm that is used to generate the action type taskblock. It introduces the assumptions made and illustrates the working of the algorithm with examples and figures.

1.2 Prior Research

One of the primary goals of developing a modelling framework for automatic control synthesis for complex systems is to make it compact. Considerable research has been done in the area of state aggregation and state explosion. To tackle the systems which are of the size and complexity that we find in typical industrial projects, colored petri nets(CP-Nets) were developed by Kurt Jensen [Jen98], [Jen98]. CP-nets is a modelling language, a combination of Petri nets and programming language. The concept of attaching a data type to the token, introduced in CP-Nets, drastically reduced the size of the model, making it more readable and understandable.

Caines et al, introduced the notion of state aggregation for finite machines via the concept of dynamical consistency relation [Cain95], [Cain97]. The state-space is divided into partitions where the dynamic consistency relation relates between the blocks of states in any given partition. Hence, the state aggregation was achieved.

Dwyer et al, presented a compact Petri net representation for concurrent programs in the field of software engineering research [Dwy95]. Since these Petri nets were based on *task interaction graphs*, they are called TIG-based Petri nets. A compact representation was possible by maintaining only those parts of state space that are relevant to the analysis of a particular property.

The approach of extending the already research techniques to high level nets like colored petrinets has been taken by various other researchers like Gries in the performance modeling of various memory architectures [Gries00].

The current thesis introduces the notion of color of a token in the condition systems world. We describe the modeling framework of color condition systems in chapter 3 and explain in detail the taskblock generation technique (Chapter 5).

Chapter 2

Condition Systems

2.1 Introduction

In this chapter we present an informal introduction to condition systems by means of an example and formally define them. A condition system is a form of Petri net that requires conditions to enable transitions and that outputs conditions according to its marking. This type of modelling framework was used to define the plant behavior and the high level specification in [Holl00].

2.1.1 Petri Nets

A Petri net is a graphical and mathematical modeling tool. It consists of places, transitions, and arcs that connect them. Input arcs connect places with transitions, while output arcs start at a transition and end at a place. Carl Adam Petri introduced in 1962, this special class of genaralized graphs to address the problems of concurrency. As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems.

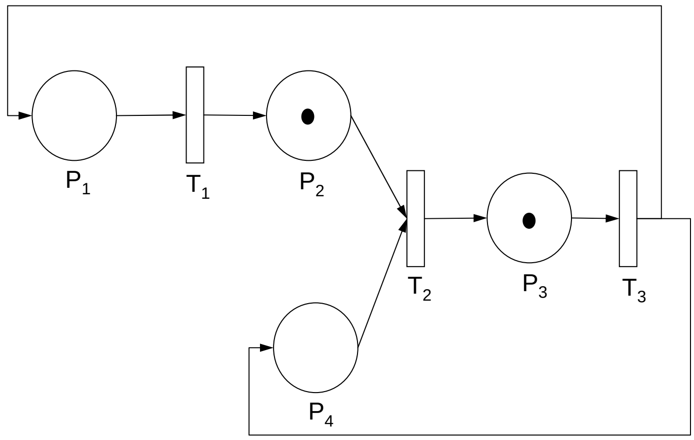

Figure 2.1: Example of a Petri net.

2.2 Condition Systems

The Condition systems interact with each other and their outside environment through *conditions*. [Holl00] A condition can be considered as a signal that either has value “true” or “false”. *AllC* represents the set of all conditions, such that for each condition *c* in *AllC*, there also exists a negated condition denoted $\neg c$, where $\neg(\neg c) = c$. Given a subset of conditions $C \subseteq AllC$, C is said to have a *contradiction* if for some $c \in C$, the negation of $\neg c$ is also in C.

A condition system is defined as a form of Petri Net that requires conditions for enabling of transitions, and that outputs conditions (establishes the truth of certain conditions) according to its marking.

DEFINITION 2.1 A condition system G is characterized by a set of states M_G, a next state mapping $f_G : M_G \times 2^{AllC} \longrightarrow 2^{M_G}$, and a condition output mapping $g_G : M_G \longrightarrow 2^{AllC}$. In this paper, we assume that M_G, f_G, and g_G are defined through a form of Petri net consisting of a set of places P_G, a set of transitions T_G, a set of directed arcs A_G between places and transitions, and a condition mapping function $\Phi(\cdot)$, where $(\forall p)\Phi(p) \subseteq AllC$ maps output conditions to each place, and

$(\forall t)\Phi(t) \subseteq \mathit{AllC}$ maps ENABLING CONDITIONS to each transition. The net is related to M_G, f_G, and g_G in the following manner:

1. THE STATES ARE THE MARKINGS OF THE PETRI NET: each state $m \in M_G$ is a function over P_G that represents a mapping of nonnegative integers to places.

2. THE OUTPUT CONDITIONS RESULT FROM MARKED PLACES: for any $m \in M_G$, $g_G(m) = \{c | \exists p \text{ s.t. } c \in \Phi(p) \text{ and } m(p) \geq 1\}$

3. NEXT-STATE DYNAMICS DEPEND ON STATE ENABLING AND CONDITION ENABLING: for any $m \in M_G$ and any $C \subseteq \mathit{AllC}$, $m' \in f_G(m, C)$ if and only if there exists some transition set T such that

   (a) T is STATE-ENABLED, meaning $(\forall p \in P_G)\ m(p) \geq |\{t \in T | p \text{ is input to } t\}|$

   (b) T is CONDITION-ENABLED, meaning $(\forall t \in T)\ \Phi(t) \subseteq C^*$

   (c) the next marking m' satisfies $\forall p \in P_G$,

   $m'(p) = m(p) - |\{t \in T|\ p \text{ is input to } t\}| + |\{t \in T|\ p \text{ is output of } t\}|$

4. M_G IS CLOSED UNDER $f_G(\cdot)$: if $m \in M_G$ and $m' \in f_G(m, C)$ for some $C \subseteq \mathit{AllC}$, then $m' \in M_G$.

We note that items in 3a and 3c above correspond to standard Petri net state enabling and firing of a transition set, respectively. Item 3b adds an additional transition set enabling constraint that the input conditions to each transition must also be within the considered set C^* of true conditions.

We define the output condition set for a system G as $C_{out}(G) = \{ c \in \Phi(p) \mid p \in G \}$. And similarly, define $C_{in}(G) = \{ c \in \Phi(t) \mid t \in G \}$

2.2.1 Example

Consider a passenger car seat of Benz [URL]. The seat is equipped with motors and sensors. A seat control unit is installed for controlling the seat. Consider the

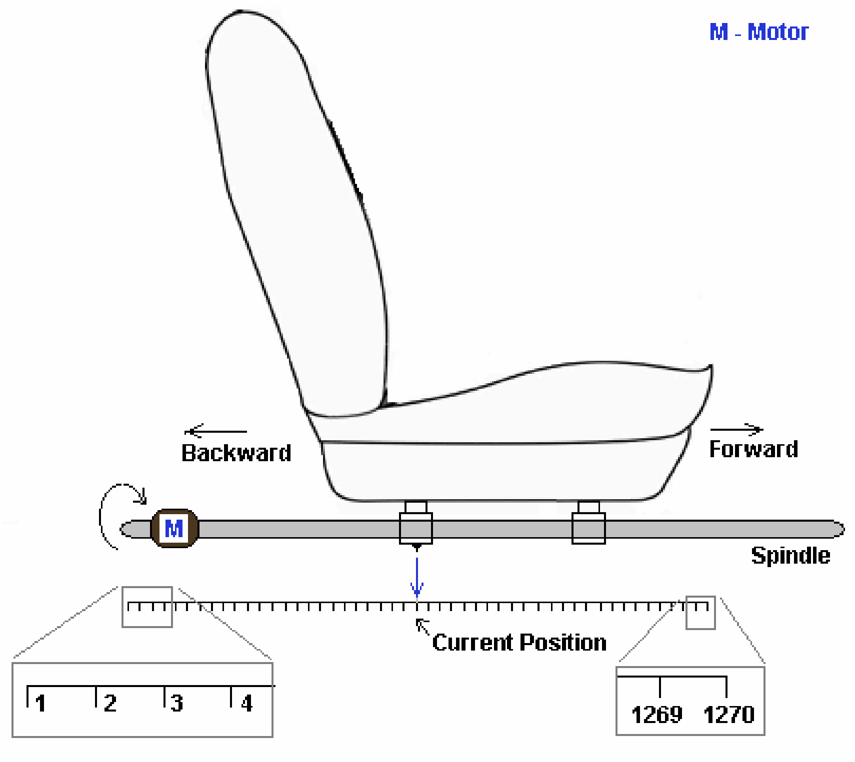

Figure 2.2: Longitudinal Adjustment Drive

longitudinal Adjustment Drive (LAD). This motor(LAD) is fitted with a Hall sensor which indicates the movement of the adjustment axis through ticks and can move the seat both forward and backward. A tick is one complete rotation of the spindle on which the seat is rested. The total number of longitudinal adjustment ticks are 1270.

In fig. 2.3, we modelled the plant described above in the condition systems framework [Holl00]. The conditions 'MF' and 'MB' on the transitions are Motor Forward and Motor Backward respectively. From the model we see that there are 2540 different possible states.

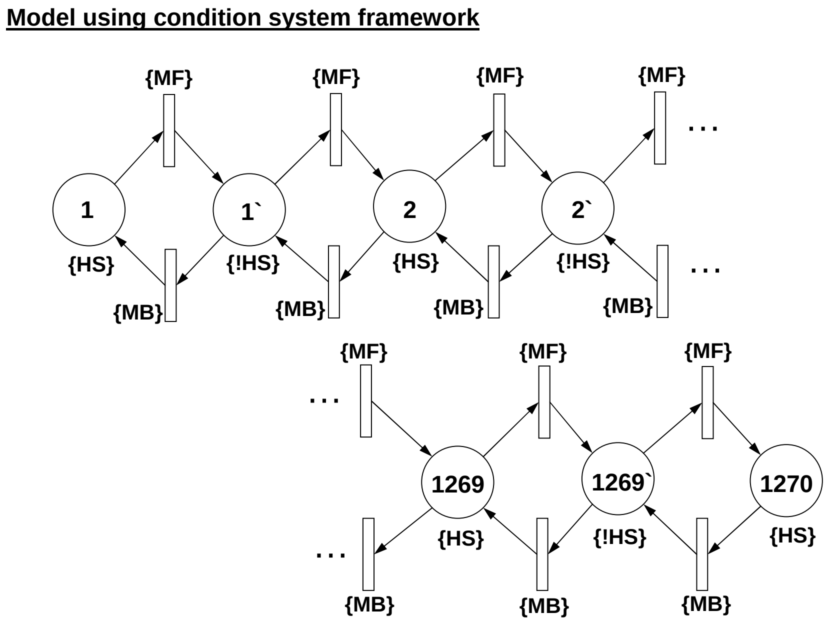

Figure 2.3: Model using condition system framework

2.2.2 Summary

In the prior research on condition systems, algorithms for synthesizing action type and maintain type taskblocks that can be used as a controller to drive the system to a given target condition [Holl00], [Holl02] were developed for the systems modelled in framework described in the previous section. The action blocks created were intended to be sequenced together to create control structures that drive a system through a sequence of conditions. The research was extended from analyzing system nets that were one layer deep to models that are several layers deep.

Since every output state cannot be associated with a sensor in practice all the time, State Observers were introduced [Holl03] to observe the state of a system.

The systems with larger state spaces similar to the example cited in section 2.2.1 need a more formal approach. Notice that the example shows 2540 different possible states which makes the modelling very difficult. The current thesis aims at defining a modelling framework to counter this problem and describing algorithms to synthesize action type taskblocks that can behave as a controller to drive the system to a given target condition.

Chapter 3

Modelling Framework

3.1 Introduction

The purpose of this chapter is to introduce colored condition systems. A detailed description of the modelling framework that we use to define the plant behavior is discussed in this chapter. We start with Preliminaries section 3.2 stating the notation we follow in this thesis and describing the notion of intervals. We then introduce the concept of color of a token (sec: 3.3.1), before defining color condition systems (sec: 3.3.2). At the end of this chapter we illustrate the color condition system by means of an example (sec: 3.3.3).

3.2 Preliminaries

The following notation is used in this chapter and thesis.

- $\mathbb{R}$: Set of Real Numbers.
- $\mathbb{Z}$: Set of Integers.
- $\mathbb{I}$: Set of closed intervals of integers in addition to the semi-closed intervals $(-\infty, \alpha]$ and $[\alpha, \infty)$ for any $\alpha \in \mathbb{Z}$, as well as the interval $(-\infty, \infty)$
- We use $\mathbb{R}^n$, $\mathbb{Z}^n$, $\mathbb{I}^n$ to denote the set of column matrices of dimension 'n' of real numbers, integers and intervals respectively and $\mathbb{R}^{n\times m}$, $\mathbb{Z}^{n\times m}$, $\mathbb{I}^{n\times m}$ to

represent matrices of dimension n by m.

- *AllC*: The universe of all conditions.
- `I`: Identity Matrix.
- `O`: Null Matrix.

Note: In the definition of color condition systems, we use N to represent dimensions. For example, $N_{cg} = |C_G|$ and since C_G is an ordered set of conditions (as seen in the definition later), the notation c_j (where $c_j \in C_G$ and $1 \leqslant j \leqslant N_{cg}$) will represent the j^{th} element in the set.

3.2.1 Intervals

Integer Matrix, $\upsilon \in \mathbb{Z}^n$

We use υ to denote a column integer matrix. $N(\upsilon)$ represents the dimension of the column matrix. We refer an element at column 'x' as $(\upsilon)_x$.
Example

- $\upsilon_1 = \begin{pmatrix} 1 \\ 10 \\ 30 \end{pmatrix}$
  where,
  $(\upsilon_1)_1 = 1$, $(\upsilon_1)_2 = 10$, $(\upsilon_1)_3 = 30$. and $N(\upsilon_1) = 3$.

Note : We use Υ to denote a set of above stated integer matrices.

Matrix of Integer Intervals, $\mathcal{I} \in \mathbb{I}^n$

We use '$\mathcal{I}$' to denote a column matrix of integer intervals. An interval *r*, can be one of the following four types.

1. $r = [\![r_1, r_2]\!] = \{ r_x \mid r_1 \leqslant r_x \leqslant r_2, r_x \in \mathbb{Z}\}$

2. $r = (\!(r_1, r_2)\!) = \{ r_x \mid r_1 < r_x < r_2, r_x \in \mathbb{Z}\}$

3. $r = [\![r_1, r_2)\!) = \{ r_x \mid r_1 \leqslant r_x < r_2, r_x \in \mathbb{Z}\}$

4. $r = (\!(r_1, r_2]\!] = \{ r_x \mid r_1 < r_x \leqslant r_2, r_x \in \mathbb{Z}\}$

$\mathrm{N}(\mathcal{I})$ represents the dimension of the column matrix. We refer to an element at column 'x' as $(\mathcal{I})_x$.

Definition 3.1 $\upsilon \in \mathcal{I}$

Given some $\mathcal{I} \in \mathbb{I}^n$ and $\upsilon \in \mathbb{Z}^n$, we define the notation $\upsilon \in \mathcal{I}$, if and only if

- $\mathrm{N}(\upsilon) = \mathrm{N}(\mathcal{I})$ and
- $(\upsilon)_x \in (\mathcal{I})_x \quad \forall\, 1 \leqslant x \leqslant n$

Example

$$\mathcal{I}_1 = \begin{pmatrix} [\![1, 10]\!] \\ [\![2, 100]\!] \end{pmatrix}$$

where,

$(\mathcal{I}_1)_1 = [\![1, 10]\!]$, $\quad (\mathcal{I}_1)_2 = [\![2, 100]\!]$. Now given, $\upsilon = \begin{pmatrix} 3 \\ 10 \end{pmatrix}$ then $\upsilon \in \mathcal{I}$

since,

- $\mathrm{N}(\upsilon) = \mathrm{N}(\mathcal{I}_1)$
- $3 \in [\![1, 10]\!] \quad [(\upsilon)_1 \in (\mathcal{I}_1)_1]$
- $10 \in [\![2, 100]\!] \quad [(\upsilon)_2 \in (\mathcal{I}_1)_2]$

Note: Set of matrices of integer intervals, 'I': We use 'I' to denote a set of matrices of integer intervals.

Example

$I = \{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3\}$

Addition: $\mathcal{I}+\upsilon$

$\mathcal{I} + \upsilon = \mathcal{I}'$ where, $l(\mathcal{I}') = l(\mathcal{I}) + \upsilon, \quad u(\mathcal{I}') = u(\mathcal{I}) + \upsilon$

Union: $\mathcal{I}_1 \cup \mathcal{I}_2$

$\mathcal{I}_1 \cup \mathcal{I}_2 = I$

where,

- $I = \{\, \mathcal{I}_1, \mathcal{I}_2 \,\}$ if $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$
- $I = \{\, \mathcal{I}' \,\}$ if $\mathcal{I}_1 \cap \mathcal{I}_2 \neq \emptyset$

  where, $l(\mathcal{I}') = \mathrm{MIN}(\, l(\mathcal{I}_1), l(\mathcal{I}_2) \,)$ and $u(\mathcal{I}') = \mathrm{MAX}(\, u(\mathcal{I}_1), u(\mathcal{I}_2) \,)$

Note: The lower limit of an interval r is denoted by $l(r)$ and the upper limit of the interval is denoted by $u(r)$.

Union: $\boldsymbol{I}_1 \cup \boldsymbol{I}_2$

$$I_1 \cup I_2 = \bigcup_{\forall \mathcal{I} \in I_1 \text{ and } I_2} \mathcal{I}$$

3.3 Colored Condition models

Condition systems communicate with each other and with their outside environments through CONDITIONS. In colored condition systems, a condition can be one of the following two types.

1. Binary valued conditions $(\mathrm{C_B})$
2. Multi valued conditions $(\mathrm{C_M})$

Thus, *AllC* is partitioned into the sets $\mathrm{C_B}$ and $\mathrm{C_M}$. $\mathrm{c} \in \mathrm{C_B}$, like in condition systems [Holl00], is a signal that either has the value “True” or “False”. A $\mathrm{c} \in \mathrm{C_M}$ on the other hand is a signal that has either a zero or a positive integer value. For each binary condition $c \in AllC$, there exists a logically negated condition denoted by $\neg\, c$, where $\neg\, (\neg\, c) = c$.

DEFINITION 3.2 : *'dc' or 'Don't Care'*

We define the symbol 'dc' as a 'do not care' signal. A condition with a value 'dc' implies that we do not care about that condition and what ever value it has at that time instant is immaterial. Hence, 'dc' is any value in the interval $(\!(-\infty, \infty)\!)$.

DEFINITION 3.3 *basis(υ) = C*

The function basis(υ) = C, defines that the value of c_j in the ordered condition set 'C' is equal to υ_j of column matrix element υ, where $1 \leqslant j \leqslant N_C$.

Example:

Let C = {c_1, c_2, c_3}. Given a matrix υ, If basis of υ is C, then υ is a column matrix of size 3(i.e., N_C) and the values of c_1, c_2 and c_3 are υ_1, υ_2 and υ_3 respectively or $\upsilon(c_1) = \upsilon_1$, $\upsilon(c_2) = \upsilon_2$, $\upsilon(c_3) = \upsilon_3$

3.3.1 Color of a token

In condition systems discussed in chapter 2, the presence of a token in a place for a system G_{sys} at any instant of time would define the system state. This would require a modeler to represent each state possible in the system by a place. Hence, as the states increase, the number of places in the system also increase and for a system with larger number of states the net becomes very large.

In order to represent a condition system more compactly we equip each token with a color. The color of a token is the data value attached to it. A token color is defined through a column matrix of color elements and let N_k be the number of these elements. Each matrix element called '*color element*' can take an integer value. The matrix is denoted by K_{token} and the i^{th} element is referred as $k_{i-token}$.

The first color element of matrix K_{token} is either a zero or a one. The significance of this color element can be seen in the definition of marking of the Petri net (refer section 3.3.2). The other color elements represent the values of the multi valued conditions in the system, that are output by the system and they will be zero when the first color element is zero.

We now formally define color condition systems, followed by an example that illustrates the definition.

3.3.2 Definition of Color Condition System

DEFINITION 3.4 A colored condition system *G* is a form of Petri net consisting of

1. An ordered set, C_G , of binary conditions followed by multi valued conditions of the system *G*. $N_{cg} = N_{cgb} + N_{cgm}$

   ($N_{cgb} = |\text{Binary Conditions}|$ and $N_{cgm} = |\text{Multi Valued Conditions}|$)

2. A set of places P_G,

3. A set of transitions T_G,

4. A set of input arcs A_G^{pt} directed from a place $p \in P_G$ to transition $t \in T_G$, and a set of output arcs A_G^{tp} directed from a transition $t \in T_G$ to a place $p \in P_G$, $(A_G = A_G^{pt} \cup A_G^{tp})$,

5. A Visibility matrix, $V(p)$ where $V(p) \in \mathbb{Z}^{N_{C_G} \times N_{K_{token}}}$ indicates the conditions that will be visible when the place 'p' is marked.

$$V(p) = \begin{pmatrix} \upsilon & 0 \\ 0 & A \end{pmatrix}$$

   where $\upsilon \in \{0, 1\}^{N_{cgb}}$ and $A \in \mathbb{Z}^{(N_{K_{token}}-1)\times(N_{K_{token}}-1)}$

6. A *Condition Acceptance Interval function* $CA_G(t)$ where $CA_G(t) \in \mathbb{I}^{N_{cg}}$ gives *acceptance intervals* for condition values ($\forall t \in T_G$). These define condition enabling for the transition as described below.

7. *A Token Acceptance Interval function* $TA_G(t)(p)$: For any given transition 't' and place 'p' such that $p \in {}^{(p)}t$, then $TA_G(t)(p) \in \mathbb{I}^{N_k}$ is a N_k-dimension interval of integer matrices, where N_k is the color dimension defined in the section 3.3.1 This defines state enabling for the transition as described below.

8. *Token Assignment Expression Mapping function* $AE_G(t)(p)$ where ($\forall t \in T_G$ and $\forall p \in t^{(p)}$, $\forall p' \in {}^{(p')}t$) $AE_G(t)(p)$ is an expression of the form
$$m'(p) = \sum_{\forall p' \in {}^{(p)}t} (\Omega_t(p', p) * m(p')) + \alpha_{(t)(p)}$$

   where $m'(\cdot)$ and $m(\cdot)$ are markings defined below, and

   Coefficient Matrix $\Omega_t(p, p') \in \mathbb{Z}^{N_k \times N_k}$
   The Coefficient Matrix $\Omega_t(p, p')$ is a N_k-ordered diagonal matrix.

   Constant Matrix $\alpha_{(t)(p)} \in \mathbb{Z}^{N_k}$
   The Constant Matrix $\alpha_{(t)(p)}$ is a N_k-ordered column matrix.

G is characterized by a set of markings M_G, a next state mapping $\mathfrak{f}_G$, and a condition output mapping g_G. The net is related to M_G, g_G, f_G as follows

1. MARKING
   The states are markings of the Petri net. Each state $m \in M_G$ is a function over P_G that represents a mapping of matrices to places. A marking for a place p is defined as follows.

   $(\forall p)\ m(p) \in \mathbb{Z}^{N_k} \cup 0^{N_k}$ where
   $m(p) = 0^{N_k}$ represents no token in the place p and
   $m(p) = \vec{z} \in \mathbb{Z}^{N_k}$ with $\vec{z} \neq 0$ represents a token of color vector value $\vec{z}$ at place p.

2. THE OUTPUT CONDITIONS RESULT FROM MARKED PLACES AND COLOR OF THE TOKENS
   For any $m \in M_G$, $g_G(m) = \max_{p \in P_G} (V(p) * m(p))$.

Note that $basis(g_G(m)) = C_G$

3. NEXT STATE DYNAMICS

   For any $m \in M_G$ a transition 't' can fire if and only if

   - 't' is token enabled. A Transition 't' is token enabled under marking 'm' if every input place to 't' has a token and the token belongs to the token acceptance interval. Formally,

     A transition 't' is token enabled iff

     $\forall\ p \in {}^{(p)}t$, $m(p) \neq 0$ and $m(p) \in TA_G(t)(p)$

   - 't' is condition enabled. Given some condition value matrix υ such that $C_G \subseteq basis(\upsilon)$, a transition 't' is condition enabled under υ if for each $c \in C_G$, $\upsilon_c \in CA_G(t)$

     In words, the value of condition c is within the token acceptance interval for transition t.

   A transition 't' firing results in a new marking m', such that

$$m'(p) = \begin{cases} \sum\limits_{\forall p' \in^{(p)} t} \Omega_t(p, p') * m(p') + \alpha_{tp} & \forall p \in t^{(p)} \\ 0 & \forall p \in^{(p)} t \\ m(p) & \text{otherwise} \end{cases}$$

4. We define the function $f_G(m, \upsilon)$ as the set of all markings resulting from firing a transition that is enabled under marking m and condition vector υ where $C_G \subseteq basis(\upsilon)$

5. M_G is closed under $f_G(\cdot)$: if $m \in M_G$ and $m' \in f_G\ (m, \upsilon)$ for some υ, such that $basis(\upsilon) \subseteq AllC$, then $m' \in M_G$.

The output condition set($C_{out}\ (G)$) and input condition set($C_{in}\ (G)$) for a system G are defined as

- $C_{out}(G) = \{ c_j \mid c_j \in C_G, V_{j,l}(p) = 1$ for some $l, p \in P_G, 1 \leqslant j \leqslant N_{cg}\}$
- $C_{in}(G) = \{ c_j \mid c_j \in C_G, [CA_G(t)]_j \neq dc, t \in T_G, 1 \leqslant j \leqslant N_{cg} \}$

Note: We refer the place where the token attains its initial color as *home* and the initial color as k_{home}.

3.3.3 Assumptions

A color condition system defined above has the following assumptions

1. *Assumes safe marking:*

   A marking M for a Petri net is bounded if there is some positive integer n having the property that in any firing sequence, no place ever receives more than n tokens. If a marking M is bounded and in any firing sequence no place ever receives more than one token, we call M a safe marking. Note that by the "Next State Dynamics" definition above, no place can ever contain multiple tokens.

2. *No self loops:*

   In system G, $\forall\, t \in T_G, {}^{(p)}t \cap t^{(p)} = \phi$.

   There exists no transition which has the same place as its input as well as output.

3. *At most one transition firing can occur at a time.*

Example

Consider an elevator car of a building with 100 floors. The car glides on a vertical shaft. A magnetic sensor on the side of the car reads a series of holes on a long vertical tape in the shaft. By counting the holes (100 in total) speeding by, one can determine the location of the car in the building. Let us model this example in the color condition systems framework.

The elevator motor, 'M' can move the elevator up or down. Let these input conditions be 'M_{up}' and 'M_{down}'. Let the condition output by the magnetic sensor be 'Mag_Sens'. The sensor 'Up_Sens' goes high if the elevator is going up. All these four are binary conditions. Let us introduce a color element 'index' into the model to keep track of the hole count.

Hence, $C_G = \{\text{Mag\_Sens}, \text{Up\_Sens}, M_{up}, M_{down}, \text{index}\}$,

Color Matrix, $K_{token} = \begin{pmatrix} 1 & \text{index} \end{pmatrix}^T$

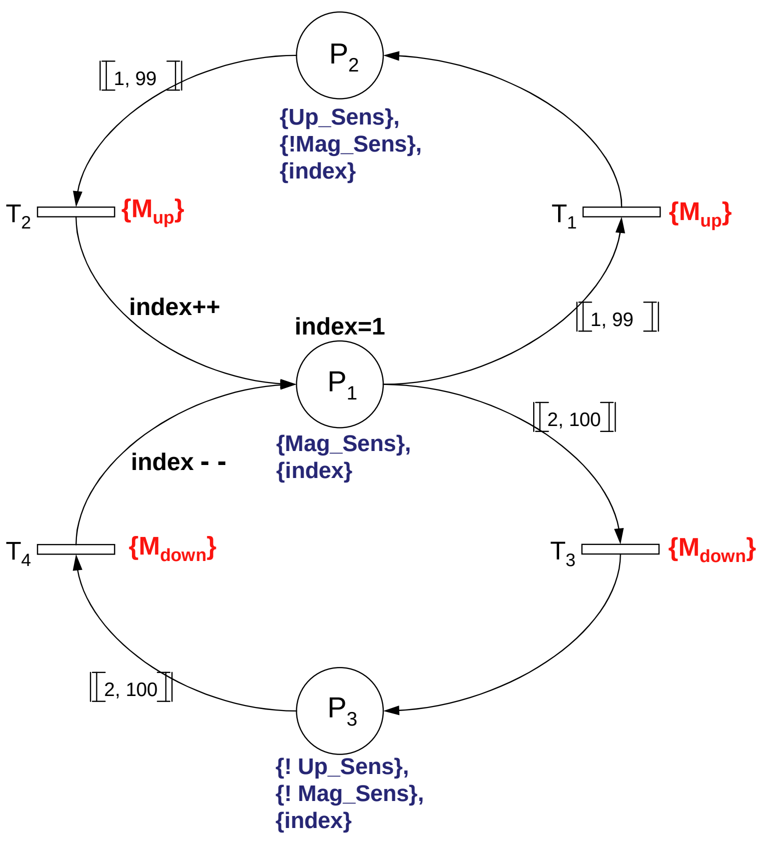

Figure 3.1: User Elevator Car Model

1. The set of places is, $P_G = \{P_1, P_2, P_3\}$

   In this net the place P_1 is *home*. Observe that the token gets initialized with a

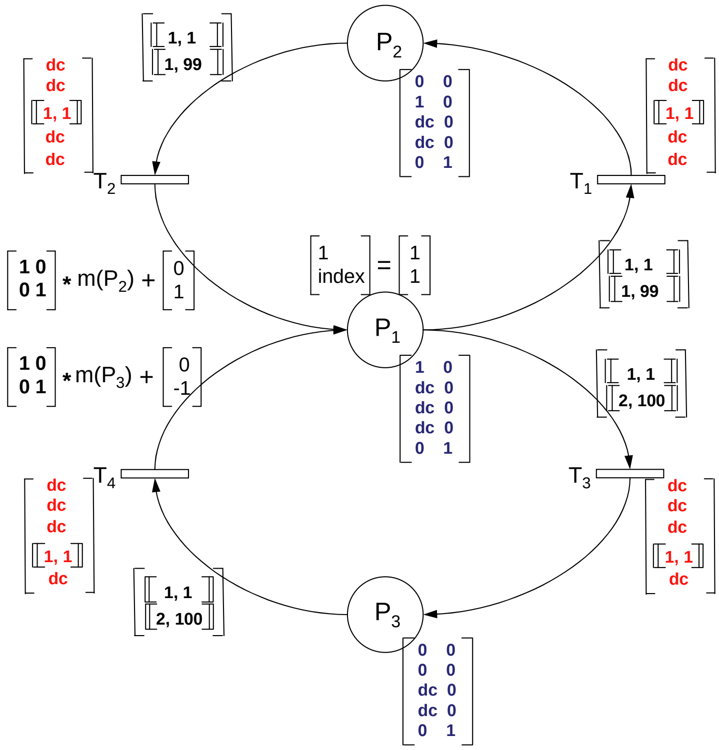

Figure 3.2: Actual Elevator Car Model

color(index = 1) at P_1. So $m_0(p_1) = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, $m_0(p_2)$, $m_0(p_3) = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$,

2. The set of transitions is, $T_G = \{T_1, T_2, T_3, T_4\}$

3. The Visibility Matrix $(\forall p \in P_G)\ V(p) \in \mathbb{Z}^{N_{C_G} \times N_{K_{token}}}$ is defined (using the ordered set of C_G) as

   - $V(P_1) = \begin{bmatrix} 1 & dc & dc & dc & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T$
   - $V(P_2) = \begin{bmatrix} 0 & 1 & dc & dc & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T$,
   - $V(P_3) = \begin{bmatrix} 0 & 0 & dc & dc & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T$

when the place P_1 is marked, then according to the definition of color condition systems, since the output conditions result from marked place and color of the token, we see that the output conditions associated with P_1 are {Mag_Sens, index} since

$$\begin{bmatrix} 1 & 0 & dc & dc & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T * \begin{pmatrix} 1 & index \end{pmatrix}^T = \begin{bmatrix} 1 & dc & dc & dc & index \end{bmatrix}^T$$

as the *basis* is C_G

4. The Condition Acceptance Interval functions $CA_G(t)$ (defined using ordered set C_G)

   - $CA_G(T_1) = \begin{bmatrix} dc & dc & [\![1, 1]\!] & dc & dc \end{bmatrix}^T$
   - $CA_G(T_2) = \begin{bmatrix} dc & dc & [\![1, 1]\!] & dc & dc \end{bmatrix}^T$
   - $CA_G(T_3) = \begin{bmatrix} dc & dc & dc & [\![1, 1]\!] & dc \end{bmatrix}^T$
   - $CA_G(T_4) = \begin{bmatrix} dc & dc & dc & [\![1, 1]\!] & dc \end{bmatrix}^T$

   Thus, M_{up} must have a value ‘1’ for T_1 or T_2 to be condition enabled and M_{down} must have a value ‘1’ for T_3 or T_4 to be enabled.

5. The Token Acceptance Interval function $TA_G(t)(p)$

   - $TA_G(T_1)(P_1) = \begin{bmatrix} [\![1, 1]\!] & [\![1, 99]\!] \end{bmatrix}^T$
   - $TA_G(T_2)(P_2) = \begin{bmatrix} [\![1, 1]\!] & [\![1, 99]\!] \end{bmatrix}^T$
   - $TA_G(T_3)(P_1) = \begin{bmatrix} [\![1, 1]\!] & [\![2, 100]\!] \end{bmatrix}^T$
   - $TA_G(T_4)(P_3) = \begin{bmatrix} [\![1, 1]\!] & [\![2, 100]\!] \end{bmatrix}^T$

6. Token Assignment Expression Mapping function $AE_G(t)(p)$

   - $AE_G(T_2)(P_1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * m(P_2) + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

- $AE_G(T_4)(P_1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * m(P_3) + \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

Dynamics:

The initial marking of the net is

$$\begin{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \emptyset & \emptyset \end{pmatrix}$$

since initially, there is a token at place P_1 with color $\begin{pmatrix} 1 & 1 \end{pmatrix}^T$. This implies that the elevator is in the first floor when the system is initialized.

Note: The presence of $\emptyset$ in the above initial marking implies the absence of a token in the corresponding place. In other words, it is equivalent to a token with all zero values.

Now let us consider the marking

$$\begin{pmatrix} \emptyset & \begin{pmatrix} 1 \\ 47 \end{pmatrix} & \emptyset \end{pmatrix}.$$

This is the instant where the elevator car is moving up and has just passed the 47_{th} floor. In other words, the place P_2 has a token with color $\begin{pmatrix} 1 & 47 \end{pmatrix}^T$

Consider that at this time instant the transition T_1 is *condition enabled*. In other words, M_{up} is TRUE i.e., the elevator continues to go up. Since the input place to transition T_1 i.e., P_2 has a token and the token belongs to the token acceptance interval $TA_G(T_1)(P_1)$ i.e,

$$\begin{pmatrix} 1 & 1 \end{pmatrix}^T \in \begin{bmatrix} [\![1, 1]\!] & [\![1, 99]\!] \end{bmatrix}^T$$

the transition is also *token enabled*. As T_1 is both token and condition enabled, it can fire. When T_1 fires then according to next state dynamics (sec: 3.3.2 point 3) in the definition of color condition systems, the resultant new marking would be

$$\begin{pmatrix} \begin{pmatrix} 1 \\ 48 \end{pmatrix} & \emptyset & \emptyset \end{pmatrix}$$

Explanation: From the definition

- $m'(P_1)$
  $$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{pmatrix} 1 \\ 47 \end{pmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{pmatrix} 1 \\ 48 \end{pmatrix}$$
  since $P_1 \in t^{(p)}$
- $m'(P_2) = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$. since $P_2 \in p^{(t)}$
- $m'(P_3) = m(P_3)$ since $P_3 \notin p^{(t)} \cap {}^{(t)}p$

Chapter 4

Task Block Effectiveness

4.1 Introduction

We use color condition models to model plants that we control. These models represent the components of the plant. Let the set of color condition models representing components be denoted as $\mathcal{G}_{\mathtt{compo}}$. The controllers that we consider are also represented as collections of color condition models. We use $\mathcal{G}_{\mathtt{tasks}}$ to denote the controller models, representing elements of control logic. These controller models are called *taskblocks*.

In the following sections we define V-sequences and language of a plant. Linear Temporal Logic(LTL) syntax that we will use through out this chapter is also discussed.

4.2 V-Sequence

Given an ordered condition set C, we define $\mathrm{AllV}|_C$ as a set of integer vectors such that for each $\upsilon \in \mathrm{AllV}|_C$, $basis(\upsilon) = C$.

A V-sequence(V_{seq} or s) over an ordered condition set C is a finite sequence of elements(with possible repetitions) of $\mathrm{AllV}|_C$.

A language, given an ordered condition set C, is the set of all V-sequences over C. We denote this language as $\mathcal{L}_C$

DEFINITION 4.1 Given a system G and an ordered condition set C, such that $C_{out}(G) \cup C_{in}(G) \subseteq C$, define function $V_{map}(m) \subseteq \mathcal{L}_C$ recursively as follows.

1. Given m_0 and some $V_0 \in AllV|_C$, then $V_0 \in V_{map}(m_0)$ if

   $\forall$ c $\in$ *basis*$(g_G(m_j))$, $(g_G(m_j))_c = (V_0)_c$. Thus, the value of condition c, driven by the system is the same as the value in V_0.

2. Given some $m_0 \ldots m_y$ and some $V_0 \ldots V_x$ s.t. $(V_0 \ldots V_x) \in V_{map}(m_0 \ldots m_y)$ then we have the following three cases.

   (a) Given an $m_{y+1} \in \mathfrak{f}_G(m_y, V_x)$ and given a condition vector $V_{x+1} \in AllV|_C$ then $V_0 \ldots V_x \in V_{map}(m_0 \ldots m_y)$ if $\forall$ c $\in$ *basis*$(g_G(m_{y+1}))$, $g_G(m_{y+1})_c = (V_{x+1})_c$

   (b) Given $(V_0 \ldots V_{x-1}, V_x) \in V_{map}(m_0 \ldots m_y)$ if $V_{x-1} = V_x$ then $V_0 \ldots V_{x-1} \in V_{map}(m_0 \ldots m_y)$

   (c) Given V_{x+1}, such that $\forall$ c $\in$ *basis*$(g_G(m_y))$ then $(g_G(m_y))_c = (V_{x+1})_c$, then $V_0 \ldots V_{x+1} \in V_{map}(m_0 \ldots m_y)$

4.2.1 Example

Consider the Elevator car example 3.3.3. The initial marking m_0 is $\left(\begin{pmatrix} 1 \\ 12 \end{pmatrix} \; \emptyset \; \emptyset \right)$. The following are the V-sequences that can be possible in $L(G, m_0)$:

- $s_1 = (\upsilon_1, \upsilon_2, \upsilon_3)$ corresponding to making the sequence $\left(\begin{pmatrix} 1 \\ 12 \end{pmatrix} \; \emptyset \; \emptyset \right)$, $\left(\emptyset \; \begin{pmatrix} 1 \\ 12 \end{pmatrix} \; \emptyset \right)$, $\left(\begin{pmatrix} 1 \\ 13 \end{pmatrix} \; \emptyset \; \emptyset \right)$

  where

  $\upsilon_1(Mag\_Sens) = 1, \upsilon_1(M_{up}) = 1, \upsilon_1(index) = 12.$

$\upsilon_2(\text{Mag\_Sens}) = 0, \upsilon_2(\text{Up\_Sens}) = 1, \upsilon_2(M_{up}) = 1, \upsilon_2(\text{index}) = 12$.

$\upsilon_3(\text{Mag\_Sens}) = 1, \upsilon_3(M_{up}) = 1, \upsilon_3(\text{index}) = 13.$

- $s_2 = (\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4)$ where

  $\upsilon_1(\text{Mag\_Sens}) = 1, \upsilon_1(\text{index}) = 12.$

  $\upsilon_2(\text{Mag\_Sens}) = 1, \upsilon_2(M_{down}) = 1, \upsilon_2(\text{index}) = 12.$

  $\upsilon_3(M_{down}) = 1, \upsilon_3(\text{index}) = 12.$

  $\upsilon_4(\text{Mag\_Sens}) = 1, \upsilon_4(M_{down}) = 1, \upsilon_4(\text{index}) =11.$

- $s_3 = (\upsilon_1, \upsilon_2, \upsilon_3)$ where

  $\upsilon_1(\text{Mag\_Sens}) = 1, \upsilon_1(M_{up}) = 1, \upsilon_1(M_{right}) = 1, \upsilon_1(\text{index}) = 47.$

  $\upsilon_2(\text{Up\_Sens}) = 1, \upsilon_2(M_{up}) = 1, \upsilon_2(\text{index}) = 47$.

  $\upsilon_3(\text{Mag\_Sens}) = 1, \upsilon_3(M_{up}) = 1, \upsilon_3(\text{index}) = 48.$

We observe that the marking corresponding to the sequence s_1 is

$$\left(\left(\begin{bmatrix} 1 \\ 12 \end{bmatrix} \ \emptyset \ \ \emptyset \right), \left(\emptyset \ \begin{bmatrix} 1 \\ 12 \end{bmatrix} \ \emptyset \right), \left(\begin{bmatrix} 1 \\ 13 \end{bmatrix} \ \emptyset \ \ \emptyset \right) \right)$$

Note: $M_{right} \notin C_G$, but still can be a part of V-Sequence in the language $L(G, m_0)$.

4.3 Taskblocks

4.3.1 Linear Temporal Logic Syntax

We will use the following temporal operators of LTL, where the operators have the mentioned intuitive meaning.

1. $\mathbb{U}$: The operator $\mathbb{U}$ represents the until operation between two propositions. If p and q represent two propositions then the formula $p \ \mathbb{U} \ q$ predicts the

eventual occurrence of q and stating that p holds continuously at least until the (first) occurrence of q.

2. $\mathbb{G}$: The $\mathbb{G}$ operator represents "Globally" or "always". $\mathbb{G}p$ read as 'always p' represents the specification that the proposition p is true for all time.

3. $\mathbb{F}$: The $\mathbb{F}$ operator represents "eventually" or "in the future". $\mathbb{F}p$ read as 'eventually p' represents that the proposition p will hold eventually sometime in future.

Notation

- $\top$: Indicates a formula that is always satisfied (always true).
- $\models$: Satisfaction Relation. Read as *satisfies*.

DEFINITION 4.2 $\textsc{ACon}()$, $\textsc{IdleCon}()$, $\textsc{ComCon}()$ For any condition vector υ_{targ} with *basis*$(\upsilon_{targ}) = \mathrm{C_G}$, we define three unique conditions , ,

1. $\textsc{ACon}(\upsilon_{targ})$
2. $\textsc{IdleCon}(\textsc{ACon}(\upsilon_{targ}))$
3. $\textsc{ComCon}(\textsc{ACon}(\upsilon_{targ}))$

Each taskblock has a specific control function. An *activation condition* uniquely identifies a taskblock, which activates the taskblock to begin its control function. If 'Υ' represents a set of target condition matrices, then let $\mathrm{C_{do}} \subset$ *AllC* represent the set of *activation conditions* where

$$\mathrm{C_{do}} = \{\textsc{ACon}(\upsilon_{targ}) \mid \upsilon_{targ} \in \Upsilon\}$$

For each element *do* $\in \mathrm{C_{do}}$ we associate the following:

1. *TB(do)* $\in \mathcal{G}_{tasks}$ is the unique taskblock for which *do* $\in \mathrm{C_{in}}$(*TB(do)*). No other taskblocks or components have *do* as an input.

2. $\textsc{ComCon}(do) \in C_{\text{out}}(TB(do))$ is a condition indicating task completion. This is output from the taskblock

3. $\textsc{IdleCon}(do) \in C_{\text{out}}(TB(do))$ is a condition indicating that the taskblock is yet to be activated and is also an output from the taskblock. Exactly one place '*p*' in *TB(do)* is associated with a visibility matrix which when marked would output $\textsc{IdleCon}(do)$. Furthermore this is the only condition output by the place '*p*'.

4. $G_{\text{compo}} \in \mathcal{G}_{\text{compo}}$ is a component model associated with the task *do*.

5. $goal(do) \in C_{\text{out}}(G_{\text{compo}})$ is a condition output from the component model.

4.3.2 Effectiveness

This section defines '*effective*', which formally describes the behavior of a taskblock when it is interacting with a system.

Definition 4.3 Given a system $\mathcal{G} \subseteq \mathcal{G}_{\text{tasks}} \cup \mathcal{G}_{\text{compo}}$ with initial state 'm_0' and a condition $do \in C_{\text{in}}(\mathcal{G}) \cap C_{\text{do}}$ such that $g(m_0)(\textsc{IdleCon}(do)) = 1$ and 'υ' such that $\text{basis}(\upsilon) = AllC$, *do* is effective for $\mathcal{G}$ under m_0 if each of the following statements are true

1. *Continued Activation implies eventual completion*

   $\forall s \in L(\mathcal{G}, m_0)$, if $s \models \mathbb{F}\mathbb{G}(\upsilon(do)=1)$, then for any formula f_{ext} such that condition basis of the formula does not include conditions in $C_{\text{out}}(G) \cup C_{\text{out}}(G_{\text{task}})$ then there exists some s' such that

   (a) $ss' \in L(\mathcal{G}, m_0)$

   (b) $s' \models f_{\text{ext}}$

   (c) $s' \models (\upsilon(do)=1)\mathbb{U}(\upsilon(do)=1 \wedge \upsilon\ (\textsc{ComCon}(do))=1\ )$

2. *Completion implies earlier activation*

   $\forall s \in L(\mathcal{G}, m_0)$, if $\mathbb{F}(\upsilon\ (\textsc{ComCon}(do))=1)$ then

   $\mathbb{F}((\upsilon(\mathrm{do}) = 1)\ \mathbb{U}\ (\textsc{ComCon}(do))=1))$

3. *Completion implies achieved goal*

   Any condition matrix sequence 's' and any condition matrix 'υ' such that $s\upsilon \in L(\mathcal{G}, m_0)$, if $\upsilon(\textsc{ComCon}(do)) = 1$ then $\upsilon(\mathrm{goal(do)}) = 1$

4. *Leaving completion means earlier deactivation*

   $\forall s \in L(\mathcal{G}, m_0)$, if $\mathbb{F}(\upsilon(\textsc{ComCon}(do))=1\ \mathbb{U}\ \upsilon(\textsc{ComCon}(do))=0)$, then

   $\mathbb{F}(\upsilon(\textsc{ComCon}(do))=1 \wedge (\upsilon(\mathrm{do}) = 0)\ \mathbb{U}\ \upsilon(\textsc{ComCon}(do))=0)$

5. *Deactivation implies eventual return to idle*

   $\forall s \in L(\mathcal{G}, m_0)$, if $s \models \mathbb{F}\mathbb{G}(\upsilon(\mathrm{do})=0)$, then for any formula f_{ext} such that condition basis of the formula does not include conditions in $C_{out}(G) \cup C_{out}(G_{task})$ then there exists some s' such that

   (a) $ss' \in L(\mathcal{G}, m_0)$

   (b) $s' \models f_{ext}$

   (c) $s' \models (\upsilon(\mathrm{do})=0)\mathbb{U}(\upsilon(\mathrm{do})=0 \wedge \upsilon(\mathrm{idle(do)})=1)$

In the above definition, the first statement states that when the condition '*do*' turns true and continues to remain true, then eventually a completion condition COMCON*(do)* will follow from the taskblock. The statement holds true for any external formula f_{ext}.

The second statement in the definition implies that if the completion condition COMCON*(do)* eventually follows in s, then the '*do*' condition should have been always true in the string s.

The third statement refers to the fact that eventual occurrence of the completion condition COMCON*(do)* simultaneously makes the goal condition *goal(do)* also true.

The fourth statement simply states that eventual truth of the completion condition ComCon*(do)* will maintain the system in the completion condition until the *do* becomes false.

The last statement says that if *do* is false, then eventually IdleCon*(do)* will become true.

Chapter 5

Action Block

5.1 Introduction

In this chapter, we present an algorithm to generate an action type taskblock. The taskblocks we generate are designed to perform a specific control function. They communicate with the system and the other taskblocks through condition signals. In other words, a condition which is output by the system could be an input to the controller. This input condition can enable a transition in the controller which could trigger a state change in it. Hence the controller would output a condition as a result of the state change, which in turn could be an input condition in the system. This would enable or disenable certain transitions of the system triggering a state change in the system. The working of the algorithm is explained by means of an example which illustrates the above communication more precisely.

We start stating the assumptions that the system must satisfy, define paths in a net and discuss further the intervals that were introduced in chapter 3. The major part of this chapter describe the procedures that make the generation of action-blocks possible. In the current synthesis technique, we assume that the operation of the controller is faster than the system.

5.2 System Structure Assumption, (SSA)

A component(G) of colored condition system satisfies the System Structure Assumption(SSA) if the following statements are true.

1. *Structure:* G is a state graph i.e., For all transitions t in G, there exists exactly one input place and one output place for t.

2. *States:* M_G consists of all states with a single place marked.

3. *Color Mapping Function:* For each place p, there exists a color mapping function $K(p) \subseteq K_G$ such that

$$K(p) = \bigcup_{\forall t_i \in {}^{(t)}p,\; p_j' \in {}^{(p)}t_i} [TA(t_i)(p_j') + \alpha_{t_i p}]$$

4. *Observability:* For any two places p, p′ and any $k \in K(p)$, $k' \in K(p')$

$$V(p) * k \neq V(p') * k'$$

5. *Token Assignment Mapping:* For any place p and p' in G such that $p' \in t^{(p)}$ where $t \in p^{(t)}$ then $\Omega(p, p') = \emptyset$ or $\Omega(p, p') = \mathtt{I}$.

6. *Transition Selectability:* For any place p in G, for all transitions $t, t' \in p^{(t)}$, where $t \neq t'$, then either $CA_G(t) \nsubseteq CA_G(t')$ or if $CA_G(t) \subseteq CA_G(t')$ then $TA_G(t)(p) \bigcap TA_G(t')(p) = \emptyset$. Also for all transitions t in G, $CA_G(t)$ is nonempty.

SSA: 1 and SSA: 2 state that the system is a state graph and there is only one token in the system at any time instant. The color mapping function (SSA:3) for a place p defines the color a token could possibly attain when it reaches the place p. The token at place p cannot attain any other color which is outside the range defined by this function. The observability statement (SSA:4) defined above states that the marking of the system at any instant uniquely identifies the state of the system. The token assignment mapping function (SSA:5) simply defines that the value of the token either increments only by an integer value or a completely new value gets assigned. A path is defined to be value dependent or value independent based on this fact (see section: 5.3.1). SSA: 6 states that there is never an ambiguity in selection of transitions. If there is more than one output transition for a place, then each transition is uniquely identifiable as none of their condition acceptance intervals are same. Even if any two output transitions from the same place have same condition acceptance intervals, the token acceptance intervals with respect to the common input places are non-intersecting.

Assumption on Prompt Control:

In a system $\mathcal{G} = \mathcal{G}_{sys} \bigcup \mathcal{G}_{ctrl}$ with a state $m \in M_{\mathcal{G}}$ where $M_{\mathcal{G}} = m_{sys} \bigcup m_{ctrl}$ and where $\mathcal{G}_{ctrl}$ represents a controller, if a transition is enabled in $\mathcal{G}_{ctrl}$ and another transition is enabled in $\mathcal{G}_{sys}$, the transition in the controller will fire first.

5.3 Actionblocks

An action-block is generated given a system G_{sys} and the target condition matrix υ_{targ}. The purpose of this taskblock, which is in the form of a color condition system, is to drive the system G_{sys} to the target condition matrix υ_{targ}. In other words it represents the control logic that takes the system to the target state which outputs υ_{targ}.

Before presenting the algorithm, we define the terms used in the algorithm.

5.3.1 Paths

Path

A path (denoted by π) is an alternating sequence of places and transitions, corresponding to a directed path in a color condition system. A path starts and ends with a place. It is denoted by π.

Example: The following are a few of the possible paths in the elevator car example 3.1 described in the chapter 3

- $\pi_1 = (P_1, T_1, P_2)$.
- $\pi_2 = (P_1, T_3, P_3, T_4, P_1)$.
- $\pi_3 = (P_1, T_1, P_2, T_2, P_1, T_3, P_3, T_4, P_1)$.

Cycle

A cycle is defined as a path with the same start and end place. It is denoted by π_c

Example: A few cycles that can be found in example 3.1 are

- $\pi_1 = (P_1, T_3, P_3, T_4, P_1)$.
- $\pi_2 = (P_1, T_1, P_2, T_2, P_1, T_3, P_3, T_4, P_1)$.

Paths can be classified into two types namely *Value-independent path* and *Value-dependent path*.

- *Value-independent path:* If there exists a subpath p_i, t_i, p_{i+1} in π such that $\Omega_{t_i}(p_i, p_{i+1}) = \emptyset$ then the path is a *Value-independent path*.

  Notice that the presence of the above condition would assign an entirely new color to the token, irrespective of its previous color. Hence there exists no constant relation between the previous and the next color.

- *Value-dependent path:* If there exists *no* subpath p_i, t_i, p_{i+1} in π such that $\Omega_{t_i}(p_i, p_{i+1}) = \emptyset$ then the path is a *Value-dependent path*.

  Notice that in the current case, the next color of the token is always dependent on the previous color of the token. Hence there exists a constant relation between the previous and the next color.

Weight

The weight of a *Value-dependent path* path is equal to the total change in the color when a token traverses along it. If the path is a cycle, then weight is equal to the total change in color when a token traverses along it once.

Lemma 5.1 Under the SSA, the weight of a Value-dependent path $\pi = (p_0, t_0, p_1, t_1, \ldots, t_{n-1}, p_n)$, denoted by $W(\pi) \in \mathbb{Z}^{N_k}$ is

$$W(\pi) = \alpha_{t_0 p_1} + \alpha_{t_1 p_2} + \ldots + \alpha_{t_{n-1} p_n}$$

Proof: Under the token assignment mapping assumption (SSA: 5), any subpath p_i, t_i, p_{i+1} in a Value-dependent path π would have $\Omega_{t_i}(p_i,\ p_{i+1}) = \mathtt{I}$. According to the definition the weight of a path is equal to the total change in the color when a token traverses along it. Hence, the only increment in the color of the token while traversing along the path π is the α value associated with each transition and its corresponding output place, that the token would visit along the path. Therefore,

$$W(\pi) = \alpha_{t_0 p_1} + \alpha_{t_1 p_2} + \ldots + \alpha_{t_{n-1} p_n}$$

■

For a Value-independent path, since there exists a subpath p_i, t_i, p_{i+1} in π $\Omega_{t_i}(p_i,\ p_{i+1}) = \emptyset$, not the same relation is possible between the color of the token at the beginning of the path and the color of the token at the end of the path every time. Therefore, the weight of a Value-independent path is undefined.

Length

The length of a path is equal to the total count of nodes (places/transitions) in the path. It is denoted by $L(\pi)$

Example: Given a path $\pi = (p_0, t_0, p_1, t_1, \ldots, t_{n-1}, p_n)$.

$L(\pi) = (2{*}n + 1)$

Master Cycles

Master cycles are *Value-dependent path* cycles with non-zero weight. A master cycle is denoted by $\tilde{\pi}_c$and a set of master cycles is denoted by $\tilde{\Pi}_c$.

Note:

1. To refer a node at k^{th} position in the path, we use the notation $\pi(k)$, where $1 \leqslant k \leqslant L(\pi)$. For example, in the path $\pi_1 = \langle p_1, t_1, p_2 \rangle$, $\pi_1(1)$ is p_1, $\pi_1(2)$ is t_1 and $\pi_1(3)$ is p_2. Also, $1 \leqslant k \leqslant L(\pi)$

2. To refer to the "head" and the "tail" in a path, we use the notation $H()$ and $T()$ respectively. Hence, $H(\pi) = p_1$ and $T(\pi) = p_2$ in the above example.

5.3.2 Intervals revisited

The lower limit of an interval r is denoted by $l(r)$ and the upper limit of the interval is denoted by $u(r)$.

For example, if $r = [\![20, 900]\!]$, then $\quad l(r) = 20, \quad u(r) = 900.$

The lower limit of an interval matrix $\mathcal{I} \in \mathbb{I}^n$ of dimension 'n', is an integer matrix in $\mathbb{Z}^n$ of the same dimension denoted by $l(\mathcal{I})$ and defined as follows.

$$l(\mathcal{I}) = [c_j] \text{ where } c_j = l(\mathcal{I}_j), 1 \leqslant j \leqslant n.$$

Similarly, the upper limit of an interval matrix $\mathcal{I} \in \mathbb{I}^n$ of dimension 'n', is an integer matrix of the same dimension denoted by $u(\mathcal{I})$ and defined as follows.

$$u(\mathcal{I}) = [c_j] \text{ where } c_j = u(\mathcal{I}_j), 1 \leqslant j \leqslant n.$$

DEFINITION 5.1 Given some $\mathcal{I} \in \mathbb{I}^N$, $\upsilon \in \mathbb{Z}^N$, define LEFTSPLIT$(\mathcal{I}, \upsilon)$ and RIGHT-SPLIT$(\mathcal{I}, \upsilon)$ as follows

1. $\textsc{LeftSplit}(\mathcal{I}, \upsilon) = \mathcal{I}'$

   where $\quad \forall\, 1 \leqslant j \leqslant n(\mathcal{I})$,

   $l(\mathcal{I}')_j = l(\mathcal{I}_j), \quad u(\mathcal{I}'_j) = \upsilon_j \qquad \text{if } \upsilon_j \in \mathcal{I}_j$

   $\mathcal{I}'_j = \mathcal{I}_j \qquad \text{if } u(\mathcal{I}_j) < \upsilon_j$

   $\mathcal{I}'_j = \phi \qquad \text{if } l(\mathcal{I}_j) > \upsilon_j$

2. $\textsc{RightSplit}(\mathcal{I}, \upsilon) = \mathcal{I}'$

   where $\quad \forall\, 1 \leqslant j \leqslant n(\mathcal{I})$,

   $l(\mathcal{I}'_j) = \upsilon_j, \quad u(\mathcal{I}')_j = u(\mathcal{I}_j) \qquad \text{if } \upsilon_j \in \mathcal{I}_j$

   $\mathcal{I}'_j = \mathcal{I}_j \qquad \text{if } l(\mathcal{I}_j) > \upsilon_j$

   $\mathcal{I}'_j = \phi \qquad \text{if } u(\mathcal{I}_j) < \upsilon_j$

Note: We generalize the above definition as follows for a set I of intervals, $\mathcal{I} \in \mathbb{I}^n$

- $\textsc{LeftSplit}(I, \upsilon) = \{\, \mathcal{I}' \mid \mathcal{I}' = \textsc{LeftSplit}(\mathcal{I}, \upsilon), \forall\, \mathcal{I} \in I \,\}$

- $\textsc{RightSplit}(I, \upsilon) = \{\, \mathcal{I}' \mid \mathcal{I}' = \textsc{RightSplit}(\mathcal{I}, \upsilon), \forall\, \mathcal{I} \in I \,\}$

Example

Consider the interval $\mathcal{I} = \begin{bmatrix} [\![1, 10]\!] & [\![1, 99]\!] \end{bmatrix}^T$ and
$\upsilon_1 = \begin{bmatrix} 5 & 50 \end{bmatrix}^T$, $\upsilon_2 = \begin{bmatrix} 5 & 100 \end{bmatrix}^T$

- $\textsc{LeftSplit}(\mathcal{I}, \upsilon_1) = \begin{bmatrix} [\![1, 5]\!] & [\![1, 50]\!] \end{bmatrix}^T$

- $\textsc{RightSplit}(\mathcal{I}, \upsilon_1) = \begin{bmatrix} [\![5, 10]\!] & [\![50, 99]\!] \end{bmatrix}^T$

- $\textsc{LeftSplit}(\mathcal{I}, \upsilon_2) = \begin{bmatrix} [\![1, 5]\!] & [\![1, 99]\!] \end{bmatrix}^T$

- $\textsc{RightSplit}(\mathcal{I}, \upsilon_2) = \begin{bmatrix} [\![5, 10]\!] & \phi \end{bmatrix}^T$

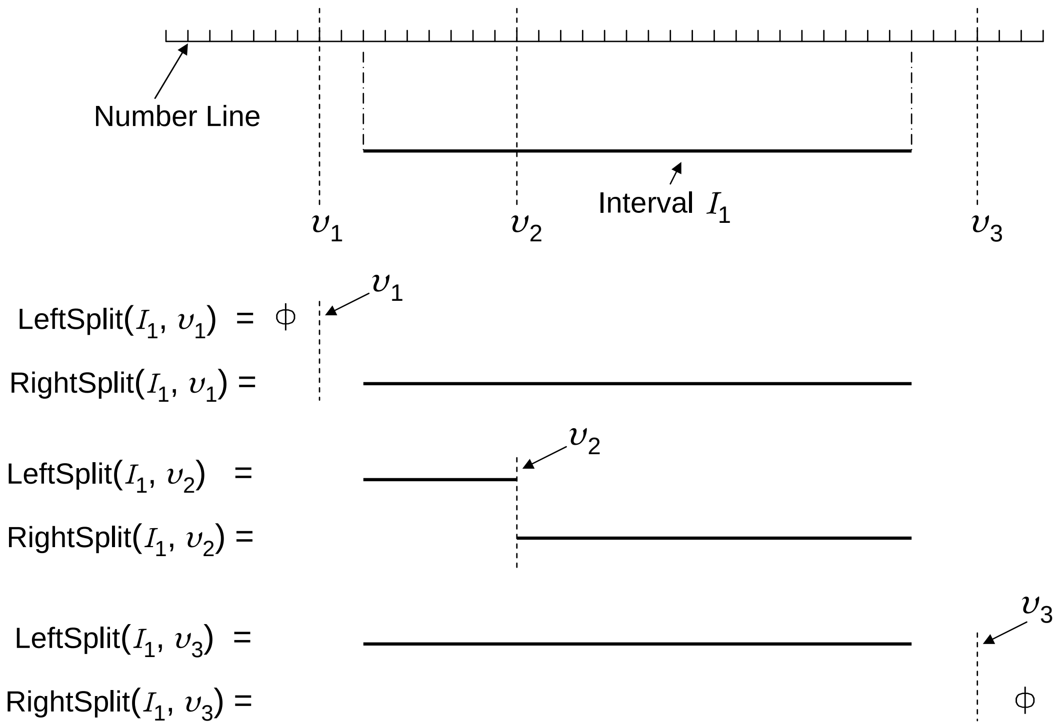

Figure 5.1: Illustration of LeftSplit() and RightSplit()

5.4 Procedure to build an Actionblock

In this section we describe the algorithm that is used to build an action type taskblock given the component model,G_{compo} and target marking υ_{targ}. The algorithm accomplishes this task by

1. *Identifying the target place and target color of the token.*

   The procedure CREATEAB() section:5.4.1 is the top level function that performs this step of identifying the target place and color. It is also responsible to coordinate the next two steps.

2. *Identifying the paths a token with certain color should trace to reach the target place with target color.*

   This step is achieved by three procedures

- DoAdjacentCycles() section:5.4.2
- DoRemoteCycles() section:5.4.3
- DoRemainingColors() section:5.4.4

Each time a path is identified by any of these procedures, the function BuildTB() is called by the procedure that identified it, to accomplish the following step.

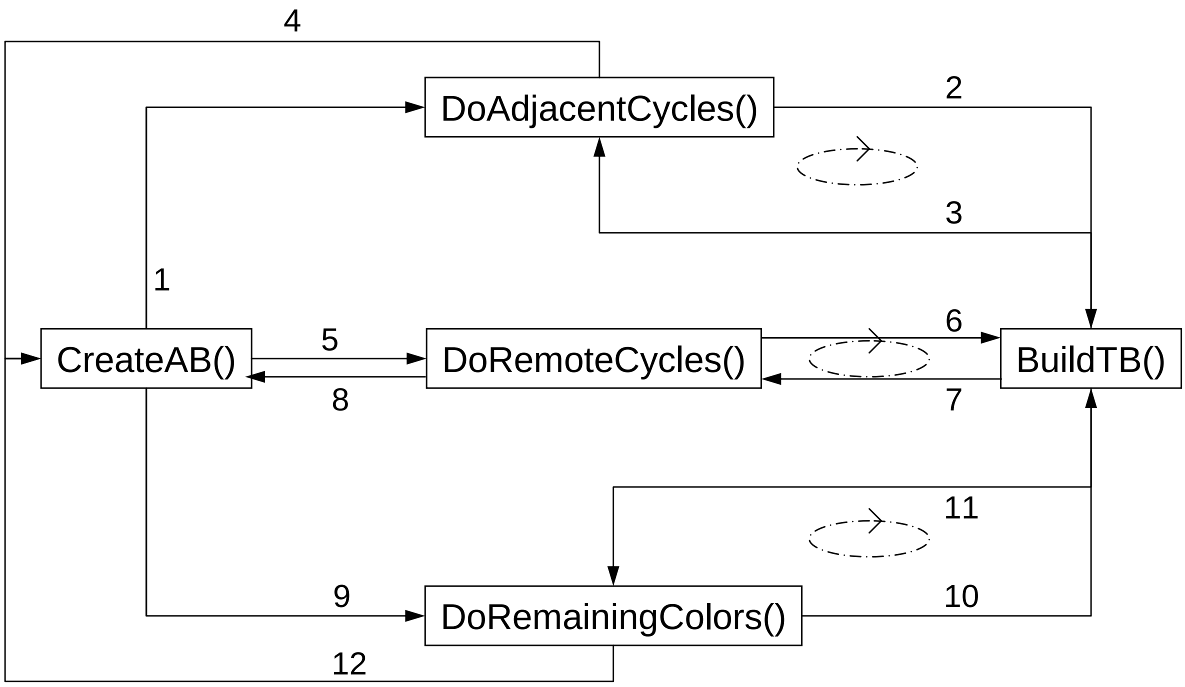

Figure 5.2: Block Diagram of the algorithm

The algorithm is also supported by procedures defined below.

Definition 5.2 Given a source node n, target node n_s and the component net G_{compo} where $n, n_s \in G_{compo}$, $\Pi_{(n,n_s)}$ is a set of all paths from n to n_s such that;

1. *Path cannot revisit source node except possibly at the end:*

   For any path $\pi \in \Pi_{(n,n_s)}$, for all k such that $1 < k < L(\pi)$ then $\pi(k) \neq n_s$.

2. *Cyclic sub-paths cannot repeat:*

   $\forall\ \pi \in \Pi_{(n,n_s)}$, for any sub-path π' in π such that π' is a cycle, then π' does not occur more than once in π.

   Note: In statement 1 the inequality on k is strict. Thus only when $n = n_s$, $H(\pi) = T(\pi) = n_s$

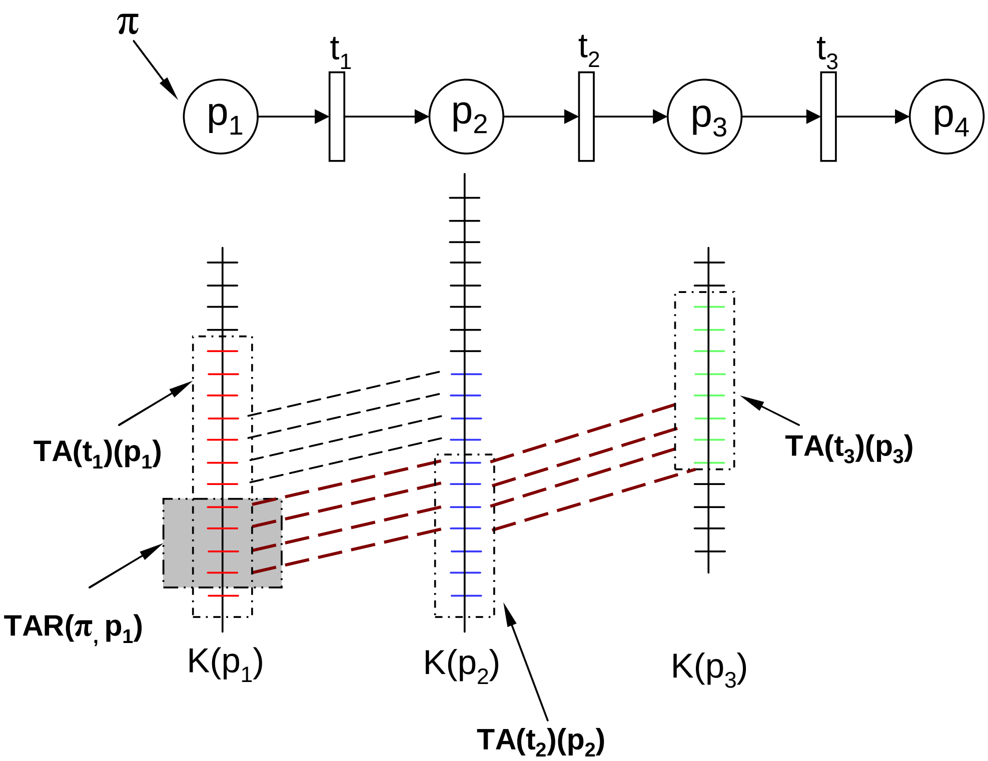

Figure 5.3: Illustration of TAR()

Definition 5.3 Given a path π ={ p₁, t₁, p₂, $\cdots$, t_{n-1}, p_n }, TAR(π, p_1) is the largest interval such that there exist intervals $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_{n-1}$ where,

1. $\mathcal{I}_1$ = TAR(π, p_1)

2. $\forall\ 1 \leqslant i \leqslant (n\text{-}1), \quad \mathcal{I}_i \in TA_G(t_i)(p_i)$

3. $\forall\ 1 \leqslant i \leqslant (n\text{-}1), \quad \mathcal{I}_{i+1} = \Omega_{t_i}(p_i, p_{i+1}) \star \mathcal{I}_i + \alpha_{(t_i)(p_{i+1})}$

Note: The Notation TAR stands for "Token Acceptance Range"

Given a path π, and a place p the procedure $TAR(\pi,\ p)$ generates the token acceptance range $\mathcal{I} \in K(p)$ such that a token at place p with color $k \in \mathcal{I}$ would continuously token enable all the transitions it would visit along π if each of the former transitions fire.

A procedure to determine TAR(π, p) is in the Appendix 6.2

The Token Acceptance Interval function defined as a part of the color condition system, behaves as a guard, by either enabling/disenabling (token enabling) the transition. On the other hand, the procedure $TAR(\pi)$ generates the token acceptance range that would allow the token to flow along a given path.

Example: Consider a path

$$\pi = \{\ p_1, t_a, p_2, t_b, p_3, t_c, p_4\ \}$$

where the token acceptance ranges are

$TA_G(t_a)(p_1) = \begin{bmatrix} [\![1, 1]\!] & [\![1, 99]\!] \end{bmatrix}^T$, $TA_G(t_b)(p_2) = \begin{bmatrix} [\![1, 1]\!] & [\![10, 150]\!] \end{bmatrix}^T$,

$TA_G(t_c)(p_3) = \begin{bmatrix} [\![1, 1]\!] & [\![50, 90]\!] \end{bmatrix}^T$

and the token assignment expression mapping functions are

$$AE_G(t_a)(p_2) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * m(p_1) + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$AE_G(t_b)(p_3) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * m(p_2) + \begin{bmatrix} 0 \\ 2 \end{bmatrix},$$

$$AE_G(t_c)(p_4) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * m(p_3) + \begin{bmatrix} 0 \\ -4 \end{bmatrix}$$

Now consider the colors in the interval $\begin{bmatrix} [\![1, 1]\!] & [\![47, 87]\!] \end{bmatrix}^T$ at place p_1. If t_a is enabled and fires then any token at place p_1 with color $k \in \begin{bmatrix} [\![1, 1]\!] & [\![47, 87]\!] \end{bmatrix}^T$

would result in a new token at place p_2 and also the resultant color will token enable t_b. Similarly, if t_b fires at this stage the resultant new token at place p_3 will token enable t_c.

The procedure *TAR(π, p)* identifies this color interval $\begin{bmatrix} [\![1, 1]\!] & [\![47, 87]\!] \end{bmatrix}^T$ which would allow the token with any color in the identified interval to traverse along the path, uninterruptedly.

DEFINITION 5.4 Given a path π, the procedure SUBCYCLES(π) returns the ordered set of cycles that a token could possibly trace along the path π, in their order of completion.

Example: Consider a path, which is also a cycle

$\pi = \{ p_1, t_a, p_2, t_b, p_3, t_c, p_2, t_d, p_4, t_e, p_5, t_f, p_4, t_g, p_6, t_h, p_4, t_i, p_7, t_j, p_1 \}$

The cycles a token possibly trace (with no cycle within it) along the path are

- $\{ p_2, t_b, p_3, t_c, p_2 \}$
- $\{ p_4, t_e, p_5, t_f, p_4 \}$

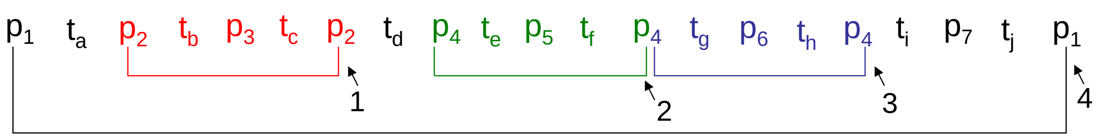

Figure 5.4: Illustration of Subcycle()

The procedure SUBCYCLES(π) extracts all these cycles in the order they appear along the path. In the above example $|\text{SUBCYCLES}(\pi)| = 4$.

DEFINITION 5.5 Given place p and color $k \in K(p)$, the function ISCOLORLEGAL(k, p) returns true if there exists some valid execution of the net from the initial marking such that the place p is marked with color k.

A color *k* at place p is defined to be a legal color if and only if there exists a path from color k_{home} at *home* to *k* at p. The procedure ISCOLORLEGAL(k, p) performs this task.

DEFINITION 5.6 Given paths $\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n$ such that $T(\pi_i) = H(\pi_{i+1})$ $\forall\ 1 \leqslant i \leqslant (n-1)$, we define

$$\text{CONCAT}(\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n) = \pi'_1, \pi'_2, \ldots, \pi'_{n-1}, \pi_n$$

where $\pi'_i = \pi_i - T(\pi_i) \quad \forall\ 1 \leqslant i \leqslant (n-1)$

Note: If $\exists T(\pi_i) \neq H(\pi_{i+1})\ \forall\ 1 \leqslant i \leqslant (n-1)$, *then* CONCAT$(\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ *is undefined*

Example: Consider the following paths,

- $\pi_1 = \{ p_1, t_1, p_2 \}$
- $\pi_2 = \{ p_2, t_4, p_4 \}$
- $\pi_3 = \{ p_4, t_5, p_5 \}$

Since, $T(\pi_i) = H(\pi_{i+1})\ \forall\ 1 \leqslant i \leqslant 2$, CONCAT$(\pi_1, \pi_2, \pi_3) = \{ p_1, t_1, p_2, t_4, p_4, t_5, p_5 \}$

5.4.1 *Procedure: CreateAB()*

CREATEAB() is the top level function. It begins by generating the ordered condition set C_{TB} of the action block being built. The procedure selects a place 'p_{targ}', called the target place from the component net. It identifies the target place by searching for the goal color $k_{targ} \in K(p_{targ})$ which would output the condition matrix υ_{targ}.

CREATEAB() then begins to determine the paths a token with color $k \in K(p_{targ})$, would have to trace to attain the target color, k_{targ}. Note that the paths in the former statement are cycles of p_{targ}. Each and every color $k \in K(p_{targ}) - k_{targ}$, at place p_{targ} would have start and end at p_{targ} to reach the target state. To achieve this, the procedure calls three procedures DOADJACENTCYCLES() 5.4.2, DOREMOTECYCLES() 5.4.3 and DOREMAININGCOLORS() 5.4.4. Once a path is identified, these procedures call BUILDTB() to build the action block for the identified path. At the same time all the colors $k \in K(p_{targ})$ and $k' \in K(p)$ (where p is a place along this identified path) are marked as spotted.

The procedure then calls DOREMOTECYCLES() and DOREMAININGCOLORS(), to identify the paths a token with color $k \in K(p)$ (where $p \in P(G_{compo}) - p_{targ}$) which is not yet marked as spotted in the above stage, would have to trace.

Note: We use the color set $\widetilde{K}_p$ as a GLOBAL variable. Initially, $\widetilde{K}_p = K(p)\ \forall\ p \in P_{G_{compo}}$. In the algorithm all the spotted colors for a place p are removed from $\widetilde{K}_p$, thus leaving all the unspotted colors in it.

Procedure: CREATEAB(υ_{targ})

1. Define G_{TB} with $P_{TB} \Leftarrow \emptyset$, $T_{TB} \Leftarrow \emptyset$, and $C_{TB} \Leftarrow$ BUILDSETC$_{TB}$(υ_{targ}, $C_{G_{compo}}$).
2. **Define** $P_{targ} \Leftarrow \{p_{sys}$ in $G \mid \exists\ k \in K(p_{sys})$ where $V(p_{sys})*k = \upsilon_{targ}\ \}$
3. Choose any one $p_{targ} \in P_{targ}$
4. $k_{targ} \Leftarrow k$, for some k s.t., $V(p_{targ})*k = \upsilon_{targ}$
5. DOADJACENTCYCLES(G_{TB}, p_{targ}, k_{targ})
6. DOREMOTECYCLES(G_{TB}, p_{targ}, p_{targ}, k_{targ})
7. DOREMAININGCOLORS(G_{TB}, p_{targ}, p_{targ}, k_{goal})
8. for each ($p \in (P_{G_{compo}} - \{p_{targ}\})$) {
9. DOREMOTECYCLES(G_{TB}, p, p_{targ}, k_{targ})
10. DOREMAININGCOLORS(G_{TB}, p, p_{targ}, k_{goal})

$\}$

BuildSet$_{\mathrm{TB}}$

Each taskblock generated has an ordered set of conditions C_{TB} associated with it. Every action-block that is created has a specific purpose; to accomplish a certain task. The condition that would activate the action-block to accomplish its assigned task is the activation condition $\textsc{ACon}(\upsilon_{targ})$. The first condition in C_{TB} represents this activation condition. The second condition in C_{TB} is the idle condition $\textsc{IdleCon}(\textsc{ACon}(\upsilon_{targ}))$ followed by the completion condition $\textsc{ComCon}(\textsc{ACon}(\upsilon_{targ}))$, the truth of which notifies that the task of action-block is accomplished.

The remaining conditions that follow the above three conditions in C_{TB} are dependent on $C_{G_{compo}}$ (where G_{compo} is the plant associated with the action-block being built). An input to G_{compo} turns into a *do^{A}Map(C)*, while an output to G_{compo} remains unchanged in C_{TB}.

Definition 5.7 *do^{A}Map(C)*:

$do^A Map(C) = \{do_c^A \mid c \in C\} \subseteq AllC$

These conditions created in C_{TB} are in the same order as they appear in $C_{G_{compo}}$. The following procedure illustrates the creation of C_{TB}.

Procedure: $\textsc{BuildSet}C_{TB}(\upsilon_{targ}, C_{G_{compo}})$

$(C_{TB})_1 \Leftarrow \textsc{ACon}(\upsilon_{targ})$

$(C_{TB})_2 \Leftarrow \textsc{IdleCon}(\textsc{ACon}(\upsilon_{targ}))$

$(C_{TB})_3 \Leftarrow \textsc{ComCon}(\textsc{ACon}(\upsilon_{targ}))$

for $i \Leftarrow 1$ to $|C_{G_{compo}}|$

$\{$

    if ($(C_{G_{compo}})_i \in C_{in}(G_{compo})$) $\quad (C_{TB})_{i+3} \Leftarrow do^A Map((C_{G_{compo}})_i)$

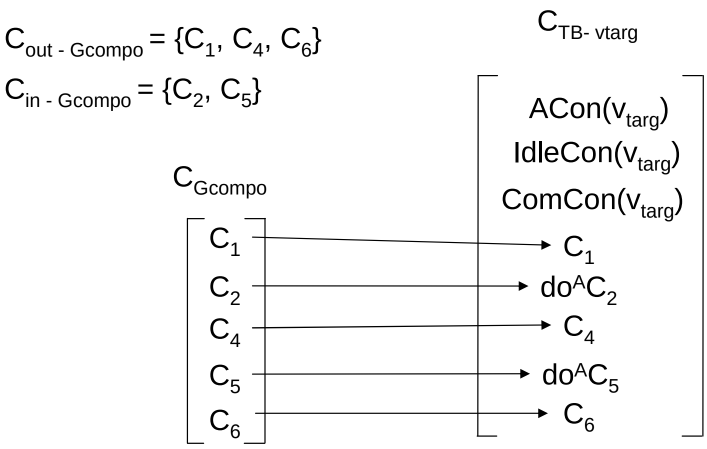

Figure 5.5: BuildSetC_{TB}: Comparison of $C_{G_{compo}}$ and C_{TB}

5.4.2 *Procedure: DoAdjacentCycles()*

Given the target place p_{targ} and the target color k_{goal}, this procedure examines each cycle π_c (s.t., $|\textsc{SubCycles}|(\pi_c) = 1$) which is both a master cycle and cycle of p_{targ} to determine that set of colors I ($I \in K(p_{targ})$ which if allowed to trace the cycle would ultimately either reach the target state or go to a state that is closer to the target state.

The procedure *CreateAB*() calls $\textsc{DoAdjacentCycles}(p_{source},\ k_{targ})$ such that $p_{source} = p_{targ}$ and $k_{targ} = k_{goal}$. The steps from 1 to 3 in the procedure described below identify all the cycles $\Pi_{+/-}$ such that there is no subcycle within any of the identified cycle and are both master cycles as well as cycles of the p_{source}.

Procedure: $\textsc{DoAdjacentCycles}(G_{TB}, p_{source}, k_{targ})$

1. $K_{p_{source}} \Leftarrow K(p_{source}),\ p_{dest} \Leftarrow p_{source}$
2. $\Pi_{cycles} \Leftarrow \{\ \pi \mid \pi \in \Pi_{(p_{source}, p_{sink})},\ |\textsc{SubCycles}(\pi)| = 1\ \}$

3. $\Pi_{+/-} \Leftarrow \{ \pi \mid \pi \in \Pi_{\text{cycles}}, W(\pi) \neq 0 \}$

4. for each $\pi_c \in \Pi_{+/-}$

5. {

6. $\quad K_{\pi_{cur}} \Leftarrow \text{TAR}(\pi_c, p_{source}) \cap K_{p_{source}}$

7. $\quad$ if $(W(\pi_c) > 0)$

8. $\quad\quad I \Leftarrow \text{LEFTSPLIT}(K_{\pi_{cur}}, k_{targ} - W(\pi_c))$

9. $\quad$ if $(W(\pi_c) < 0)$

10. $\quad\quad I \Leftarrow \text{RIGHTSPLIT}(K_{\pi_{cur}}, k_{targ} - W(\pi_c))$

11. $\quad$ BUILDTB$(G_{TB}, \{\pi_c\}, \{I\})$

12. }

Each identified cycle π_c of $\Pi_{+/-}$ is then analyzed. Step 6 extracts the color set $K_{\pi_{cur}}$ of p_{source} such that any color $k \in K_{\pi_{cur}}$ can uninterruptedly cycle through π_c. Once this set $K_{\pi_{cur}}$ is determined the actual set of colors I is detected in either step 8 or 10. Now the procedure is ready to build the task block with the identified color set

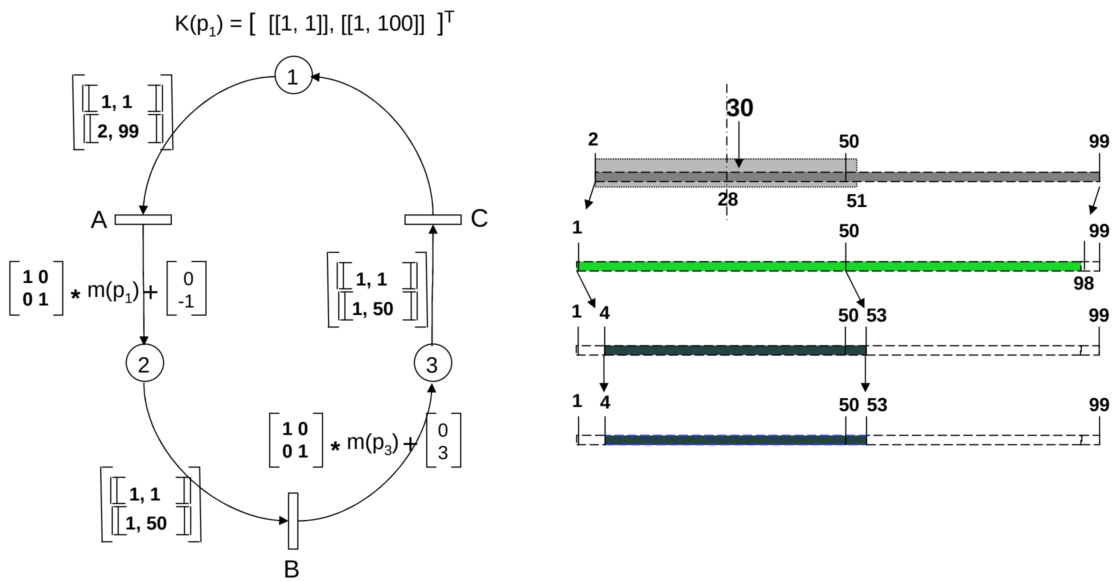

Figure 5.6: Working of the procedure AdjacentCycles()

Note: The algorithm described selects any path from $\Pi_{+/-}$ at random. This does not ensure an optimum solution and achieving an optimum solution is beyond the scope of this thesis. One way to pick a path from $\Pi_{+/-}$ so that the target state is reached sooner is as follows

Choose any $\pi_c \in \Pi_{+/-}$ *s.t.* $|W(\pi_c) / P_{\pi_c}|$ *is greatest where* $P_{\pi_c} \Leftarrow |\{ p \mid p \in \pi_c\}|$

If a cycle is chosen such that the change in the color of the token when it traverses from one place to its next place in the cycle is less than no other cycle, it is obvious that the target state can be reached faster.

5.4.3 *Procedure: DoRemoteCycles()*

Given the source place p_{source} and destination place p_{dest}, the procedure *DoRemoteCycles()* is designed to examine paths Π_{remote} of the following type.

$$\Pi_{\mathrm{remote}} \Leftarrow \{ \pi \mid \pi \in \Pi_{(p_{source}, p_{dest})}, \textsc{SubCycles}(\pi) \cap \tilde{\Pi}_c \neq \phi \}$$

1. $\Pi_{(p_{source}, p_{dest})}$: All paths starting at p_{source} and end at p_{dest}.
2. $\textsc{SubCycles}(\pi) \cap \tilde{\Pi}_c \neq \phi, \pi \in \Pi_{(p_{source}, p_{dest})}$: Paths that contain atleast one master cycle in there subcycles.

Notation

In the algorithm that follows we use the following notation.

- π_{cur}: Current path under consideration. $\pi_{\mathrm{cur}} \in \Pi_{\mathrm{remote}}$.
- π_{mc}: Current master cycle identified by the algorithm along the path π_{cur}.
- $\pi_{\mathrm{pre-mc}}$: The subpath of π along which the token travels to enter π_{mc}.
- $\pi_{\mathrm{post-mc}}$: The subpath of π along which the token travels to exit π_{mc} and reach p_{dest}.

- $\mathcal{I}_A \subseteq K(p_{source})$, $\mathcal{I}_B \subseteq K(p_{com})$, $\mathcal{I}_C \subseteq K(p_{com})$

Note:

1. $T(\pi_{pre-mc}) = H(\pi_{post-mc}) = H(\pi_{mc})$. *Let this place be denoted by* p_{com}.
2. $\textsc{Concat}(\pi_{pre-mc}, \pi_{mc}, \pi_{post-mc}) = \pi_{cur}$

The goal of the present algorithm is to identify the color set ($\mathcal{I}_A \in K(p_{source})$) so that any color $k \in \mathcal{I}_A$ at p_{source} can eventually reach either the destination state or get closer to it.

The procedure examines all the paths with fewer number of subcycles prior to analyzing those with larger number of subcycles. In other words the procedure tries to send the token along a path with fewer subcycles.

Procedure: $\textsc{DoRemoteCycles}(G_{TB}, p_{source}, p_{dest}, k_{targ})$

1. *cycmax* $\Leftarrow \max\{|\textsc{SubCycles}(\pi)|$, such that $\pi \in \Pi_{(p_{source}, p_{dest})}\}$
2. *sccount* $\Leftarrow 2$, $\Pi_{MC} \Leftarrow \tilde{\Pi}_c$
3. while (*sccount* $\leqslant$ *cycmax*)
4. {
5. for each $\pi_{cur} \in \Pi_{remote}$ s.t. $|\textsc{SubCycles}(\pi_{cur})| =$ *sccount*
6. {
7. Define υ such that,
   $\upsilon \Leftarrow k_{targ} - (\, W(\pi_c)\,)$
8. if $(W(\pi_{mc}) > 0)$ {
9. $\mathcal{I}_1 \Leftarrow \textsc{LeftSplit}(TAR(\pi_{pre-mc}, p_{source}), \upsilon\,)$
10. $\mathcal{I}_B \Leftarrow [\mathcal{I}_1 + W(\pi_{pre-mc})] \cap TAR(\pi_{mc}, p_{com})$
11. if $(\mathcal{I}_B = \phi)$
    break;
12. else {

if $(\mathrm{k}_{\mathrm{targ}} - W(\pi_{\mathrm{post-mc}})) \in \mathrm{TAR}(\pi_{\mathrm{post-mc}}, \mathrm{p}_{\mathrm{com}})$
    $\mathrm{k}_{\mathrm{rem-targ}} \Leftarrow \mathrm{k}_{\mathrm{targ}} - W(\pi_{\mathrm{post-mc}})$
else if $(u(\mathrm{TAR}(\pi_{\mathrm{post-mc}}, \mathrm{p}_{\mathrm{com}})) < \mathrm{k}_{\mathrm{targ}} - W(\pi_{\mathrm{post-mc}}))$
    $\mathrm{k}_{\mathrm{rem-targ}} \Leftarrow u(\mathrm{TAR}(\pi_{\mathrm{post-mc}}, \mathrm{p}_{\mathrm{com}}))$
else   break;
$\mathcal{I}_{\mathrm{C}} \Leftarrow [\![\, \mathrm{k}_{\mathrm{rem-targ}} - W(\pi_{\mathrm{mc}}) + 1, \mathrm{k}_{\mathrm{rem-targ}} \,]\!]$
if $\mathcal{I}_{\mathrm{C}} \notin \mathrm{TAR}(\pi_{\mathrm{post-mc}}, \mathrm{p}_{\mathrm{com}})$
    break;
if $(\ (\mathcal{I}_{\mathrm{C}} - W(\pi_{\mathrm{mc}})) \notin \mathrm{TAR}(\pi_{\mathrm{mc}}, \mathrm{p}_{\mathrm{com}})\ )$
    break;
$\mathcal{I}_{\mathrm{B}} \Leftarrow \mathcal{I}_{\mathrm{B}} \cap \textsc{LeftSplit}(\mathcal{I}_{\mathrm{B}}, u((\mathcal{I}_{\mathrm{C}} - W(\pi_{\mathrm{mc}}))))$
if $\mathcal{I}_{\mathrm{B}} = \phi$
    break;
$\mathcal{I}_{\mathrm{A}} \Leftarrow \mathcal{I}_{\mathrm{B}} - W(\pi_{\mathrm{pre-mc}})$
}
}
if $(W(\pi_{\mathrm{mc}}) < 0)$ {
$\mathcal{I}_1 \Leftarrow \textsc{RightSplit}(\mathrm{TAR}(\pi_{\mathrm{pre-mc}}, p_{source}), \upsilon)$
$\mathcal{I}_{\mathrm{B}} \Leftarrow [\mathcal{I}_1 + W(\pi_{\mathrm{pre-mc}})] \cap \mathrm{TAR}(\pi_{\mathrm{mc}}, \mathrm{p}_{\mathrm{com}})$
if $(\mathcal{I}_{\mathrm{B}} = \phi)$
    break;
else {
if $(\mathrm{k}_{\mathrm{targ}} - W(\pi_{\mathrm{post-mc}})) \in \mathrm{TAR}(\pi_{\mathrm{post-mc}}, \mathrm{p}_{\mathrm{com}})$
    $\mathrm{k}_{\mathrm{rem-targ}} \Leftarrow \mathrm{k}_{\mathrm{targ}} - W(\pi_{\mathrm{post-mc}})$
else if $(l(\mathrm{TAR}(\pi_{\mathrm{post-mc}}, \mathrm{p}_{\mathrm{com}})) > \mathrm{k}_{\mathrm{targ}} - W(\pi_{\mathrm{post-mc}}))$
    $\mathrm{k}_{\mathrm{rem-targ}} \Leftarrow l(\mathrm{TAR}(\pi_{\mathrm{post-mc}}, \mathrm{p}_{\mathrm{com}}))$
else   break;

32. $\mathcal{I}_C \Leftarrow [\![\ k_{rem-targ}, k_{rem-targ} - W(\pi_{mc})$ - 1 $]\!]$

33. if $\mathcal{I}_C \notin$ TAR($\pi_{post-mc}$, p_{com})
    break;

34. if (($\mathcal{I}_C - W(\pi_{mc})$) $\notin$ TAR(π_{mc}, p_{com}))
    break;

35. $\mathcal{I}_B \Leftarrow \mathcal{I}_B \cap$ RIGHTSPLIT($\mathcal{I}_B$, $l((\mathcal{I}_C - W(\pi_{mc})))$)

36. if $\mathcal{I}_B = \phi$
    break;

37. }

38. }

39. BUILDTB(G_{TB}, $\{\pi_{pre-mc}, \pi_{mc}, \pi_{post-mc}\}$, $\{\mathcal{I}_A, \mathcal{I}_B, \mathcal{I}_C\}$)

40. }

41. $sccount \Leftarrow sccount + 1$.

42. }

Example

Consider the color petrinet in figure 5.7 which shows a part of a plant. The above algorithm is designed to analyze paths that resemble the color petri net shown in the figure 5.7. The purpose of this example is to walk the reader through the algorithm and hence it should be noted that this example does not depict any practical system. In the present example

1. $p_{source} = p_1$, $\quad p_{dest} : p_1$.
2. $\pi_{cur} = \{p_1, t_A, p_2, t_B, p_3, t_C, p_4, t_D, p_2, t_B, p_3, t_E, p_1\}$
3. $\pi_{mc} = \{p_2, t_B, p_3, t_C, p_4, t_D, p_2\}$
4. $\pi_{pre-mc} = \{p_1, t_A, p_2\}$
5. $\pi_{post-mc} = \{p_2, t_B, p_3, t_E, p_1\}$

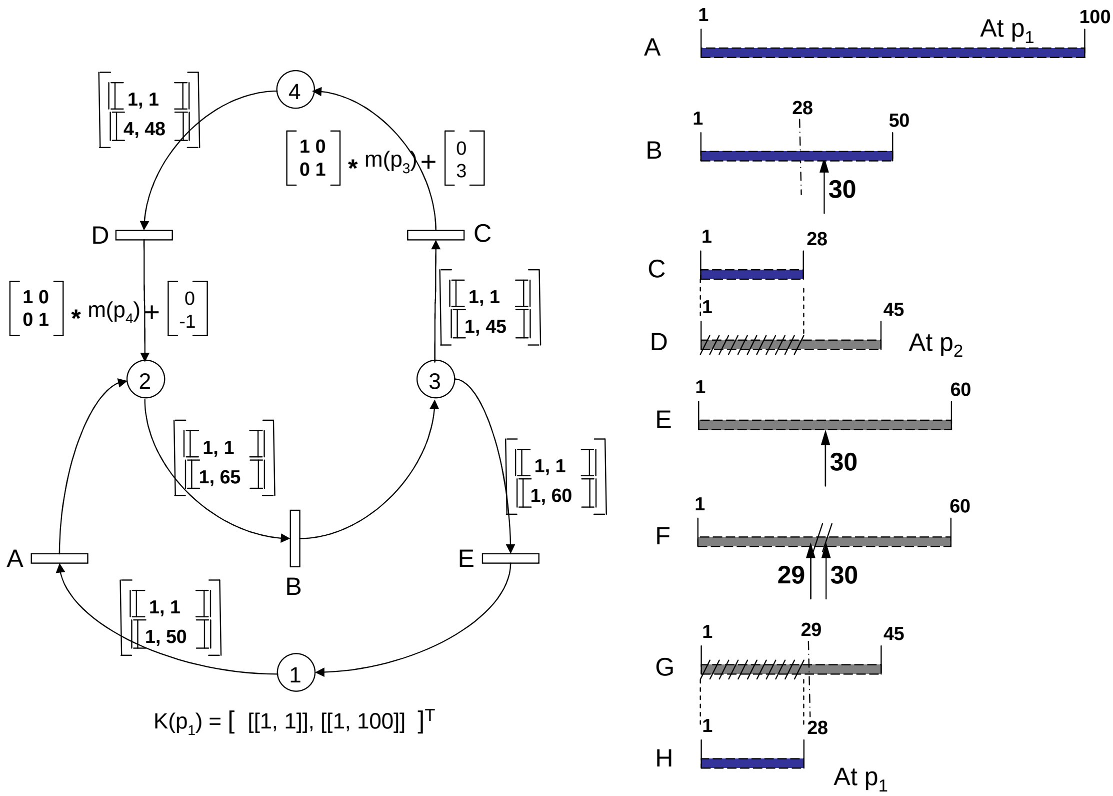

Figure 5.7: Working of the procedure DoRemoteCycles()

6. $k_{targ} = \begin{pmatrix} 1 & 30 \end{pmatrix}^T$

7. $W(\pi_{mc}) = \begin{pmatrix} 0 & 2 \end{pmatrix}^T$ $W(\pi_{pre-mc}) = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ $W(\pi_{post-mc}) = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$

Notice that $\text{CONCAT}(\pi_{pre-mc}, \pi_{mc}, \pi_{post-mc}) = \pi_{cur}$ and $T(\pi_{pre-mc}) = H(\pi_{post-mc})$ $= H(\pi_{mc}) = p_{com} = p_2$. The place p_2 (p_{com}) is significant because of the fact that it is common to all the paths.

The right side of figure 5.7, show those steps that the algorithm follows to determine $\mathcal{I}_A$, $\mathcal{I}_B$ and $\mathcal{I}_C$.

1. 'A' denotes the color set of p_1.

2. 'B' shows line 9 of the algorithm. While the entire number line represents the Token Acceptance Range of p_1 along π_{pre-mc}, it is left split along υ. The

resultant is shown as 'C' ($\mathcal{I}_1$)

3. 'D' depicts line 10. The hatched line is the projection of $\mathcal{I}_1$ on the Target Acceptance Range of p_2 along π_{mc}.

4. 'E' represents the Token Acceptance Range of p_2 along $\pi_{post-mc}$ and the number denotes $k_{rem-targ}$.

5. 'F' shows $\mathcal{I}_C$.

6. 'G' depicts the line 19 and 'H' is the projection on $\mathcal{I}_1$ i.e., on the TAR shown in 'B'.

The weight of the master cycle π_{mc} is a positive value 2. Hence the lines 8 to 23 are applicable to this example. The goal of the algorithm is to determine the subset $\mathcal{I}_A$ of the color set of p_1 (p_{source}) which can reach the destination place $p_1(p_{dest})$ such that any $k \in \mathcal{I}_A$ would transform to either k_{targ} or get closer to k_{targ} after traversing through the path π_{cur} before reaching the destination place p_1. The minimum possible change in the color would be the weight of π_{cur} itself since the token would have to traverse it atleast once. So, if a token with color ($k_{targ} - W(\pi_{cur})$) or a color value lesser than ($k_{targ} - W(\pi_{cur})$) begins at p_{source}, it would reach p_{dest} well with in k_{targ}. Hence these values are filtered out at line 9. Line 10 determines the probable color set $\mathcal{I}_B$ that can reach $p_2(p_{com})$ from p_{source} and can traverse through π_{mc} from p_2 atleast once. Line 16 determines the actual color set $\mathcal{I}_C$ of p_2 that should trace the path $\pi_{post-mc}$ to achieve the goal. Notice that if, any color $k \in \mathcal{I}_C$ determined in step 16 traces π_{mc} atleast once before traversing $\pi_{post-mc}$, then the color of the resultant token at p_2 (p_{com}) would either be unable to traverse $\pi_{post-mc}$ or if it is successful in traversing $\pi_{post-mc}$, the resultant color value of the token at p_{dest} would be greater than k_{targ} which is not desirable. Hence $\mathcal{I}_C$ becomes the desired interval at p_2 that should traverse $\pi_{post-mc}$. Line 17 ensures that every color reaching p_2 has an exit path along $\pi_{post-mc}$. Line 19 ensures that only those colors at p_2 which can eventually fall in $\mathcal{I}_C$ are assigned to $\mathcal{I}_B$. Based on $\mathcal{I}_B$, $\mathcal{I}_A$ is determined.

The logic flows in a similar manner if $W(\pi_{mc})$ is a negative.

5.4.4 *Procedure: DoRemainingColors()*

Non-Overlapping cycles property (NOC):

In a system $\mathcal{G}$, any cycle $\pi_c \in \mathcal{G}$ satisfies the following property.
For any $\pi_i, \pi_j \in \textsc{SubCycles}(\pi_c)$ where $1 \leqslant i, j \leqslant |\textsc{SubCycles}(\pi_c)|$ such that if $|\pi_i \cap \pi_j| > 1$ then either

- $\pi_i \subseteq \pi_j$ or
- $\pi_j \subseteq \pi_i$

The procedure discussed in this section, determines the paths a token with color that is not yet marked as spotted in either DoAdjacentCycles() or DoRemoteCycles() would trace. In this procedure, a path is determined for each individual unspotted color unlike in the above procedures where a path is determined for a color set. The paths that are analyzed in this procedure are those where the subcycles within the path are more than one. In the previous procedures the master cycles that were analyzed were one at a time. In the current procedure, a combination of master cycles are analyzed to determine the path a color should traverse to reach the destination state. Also note that in this procedure, the procedure determines those path which will lead all the colors the final destination unlike in previous procedures where the paths determined either allow the token with a color to reach the destination state or get closer to the destination state.

In the current procedure, an equation is constructed based on the path being considered. The task is to find out if there exists a solution such that if a token with a color k (line: 5) starts at the beginning of the path π, (3) under consideration and traverses along the subcycles of π for n_i (line: 7) times before reaching the end of the path, will it attain the destination state. We require that the total change in the color value before the token traverses the entire path should be $k_{targ} - k$. Hence the constructed equation is equated to this value. Also, note that when a token traverses along the path π the total change in the token would not only include the values of n_i times the weight of the subcycle but also the weights of intermediary paths defined in line 11.

The equation can be solved using mathematical software like MATLAB, Solver in Excel etc., Once the solution is determined, the procedure checks for the validity of solution in the lines from 14 to 19. After the validity is checked, BUILDTB() is called to construct the part of the taskblock for the determined color and path.

Procedure: DOREMAININGCOLORS(G_{TB}, p_{source}, p_{dest}, K_{goal})

1. if ($p_{source} = p_{dest}$)

   $\Pi_{cycles} \Leftarrow \{ \pi \in \Pi_{(p_{source}, p_{sink})}$ s.t. $|$SUBCYCLES$(\pi)| > 2 \}$

2. if ($p_{source} \neq p_{dest}$)

   $\Pi_{cycles} \Leftarrow \{ \pi \in \Pi_{(p_{source}, p_{sink})}$ s.t. $|$SUBCYCLES$(\pi)| > 1 \}$

3. for each $\pi \in \Pi_{cycles}$

   {

4. $\quad \Pi \Leftarrow$ SUBCYCLES(π)

5. $\quad$ for each $k \in K(p_{source})$

6. $\quad$ {

7. $\quad\quad$ Define equation eq with variables n_i, $1 \leqslant i \leqslant |\Pi|$ s.t.,

$$\sum_{\forall 1 \leqslant i \leqslant |\Pi|, \pi'_i \in \Pi} n_i * W(\pi'_i) = k_{goal} - k - (W(\pi) - \sum_{\forall \pi' \in \Pi} W(\pi'))$$

8. $\quad\quad$ solve the equation eq for the variables n_i.

9. $\quad\quad$ If solution for n_i exists,

   $\quad\quad$ for any i, j if $\pi'_j \in$ SUBCYCLES(π'_j)

   $\quad\quad$ then $n_j \geqslant n_i$

   $\quad\quad$ {

$N \Leftarrow |\textsc{SubCycles}(\Pi)|$

For π with ordered subcycle set $\pi'_0, \pi'_1, \cdots, \pi'_N$
define intermediary paths $\pi''_0, \pi''_1, \cdots, \pi''_N$ such that
$\pi = \textsc{Concat}(\pi''_0, \pi'_1, \pi''_1, \pi'_2, \pi''_2, \cdots, \pi'_N, \pi''_N)$

Assign $\mathcal{I}(\pi''_0) \Leftarrow k$

feasibleFLAG = TRUE

for i = 1 to N

{

$\quad\mathcal{I}(\pi'_i) \Leftarrow \mathcal{I}(\pi''_{i-1}) + W(\pi''_{i-1})$

$\quad\mathcal{I}(\pi''_i) \Leftarrow \mathcal{I}(\pi'_i) + n_i * W(\pi''_{i-1})$

$\quad$if $\mathcal{I}(\pi'_i) \notin K_{H(\pi'_i)}$ or $\mathcal{I}(\pi''_i) \notin K_{H(\pi''_i)}$ then *feasibleFLAG* = FALSE

}

if (*feasibleFLAG* $\neq$ FALSE) then
$\textsc{BuildTB}(\{\pi''_0, \pi'_1, \pi''_1, \pi'_2, \cdots, \pi'_N, \pi''_N\}, \{\mathcal{I}_{\pi''_0}, \mathcal{I}_{\pi'_1}\ \mathcal{I}_{\pi''_1}, \cdots, \mathcal{I}_{\pi'_N}, \mathcal{I}_{\pi''_N}\})$

}

}

}

5.4.5 *Procedure: BuildTB()*

The procedure BUILDTB() builds the actual action type taskblock for a given component model. Every time a procedure identifies a color set and its corresponding path, it calls BUILDTB().

In the procedure $\textsc{BuildTB}(G_{TB}, \Pi, I_{set}, \upsilon_{targ})$, G_{TB} is the net being built and υ_{targ} is the target condition matrix. Π is an ordered set of paths, and I_{set} is a corresponding ordered set of color intervals. For example, let $\pi_1, \pi_2, \cdots, \pi_k \in \Pi$ i.e., $|\Pi| = k$. If $|\Pi| = k$, then $|I_{set}| = k$ and the procedure is designed to allow the colors $I_{set}(j)$ (which belongs to $K(p)$, where $p = H(\pi_j)$, $1 \leqslant j \leqslant k$), traverse along the path $\pi(j)$.

BUILDTB() creates the following places and transitions in the taskblock.

- p_{idle}: The place that is marked when the taskblock is idle.
- p_{cmpl}: The place that will be marked when the task is completed and the component is outputting goal condition matrix.
- p_t: The place that is created in the taskblock from the transition t in the component. There exists one and only one place corresponding to each transition in the component.
- $t_p^{\#}$: One of the # transitions that is created in the taskblock from the place p in the component. There can exist more than one transition corresponding to a place in the component. # represents a positive integer.
- $t_{cmpl}^{\#}$: One of the # transitions in taskblock which would fire when the task is complete.

When the procedure is called for the first time, BUILDTB() creates two places p_{idle}, p_{cmpl} and the transition t_{cmpl}^{1} in the taskblock.

Procedure: BUILDTB(Π, I_{set})

1. if ($P_{TB} == \emptyset$) {
2. **Create** place p_{idle} in P_{TB}
   with $V(p_{idle})_{\text{IDLECON}(\text{ACON}(\upsilon_{targ}))} \Leftarrow 1$
3. **Create** place p_{cmpl} in P_{TB}
   with $V(p_{cmpl})_{\text{COMCON}(\text{ACON}(\upsilon_{targ}))} \Leftarrow 1$
4. **Create** transition t_{cmpl}^{1} in T_{TB} from p_{idle} to p_{cmpl}
   with $CA_{TB}(t_{cmpl}^{1})_c \Leftarrow [\![z, z]\!]$,
   $\forall$ c $\in C_{TB} \cap$ *basis*(k_{targ}), z is value of c in k_{targ}

}

5. For each $\pi \in \Pi$and for each $t \in (\pi \cap T_{\mathrm{G_{compo}}})$, {

6. if (p_t doesnot yet exist in $P_{\mathrm{G_{TB}}}$) {

7. $C \Leftarrow \{ c \mid \forall c, \text{s.t. } \mathrm{CA}_{\mathrm{G_{compo}}}(t)_c = [\![1, 1]\!]\} \cup$
$\forall \{c' \mid \forall c', \text{s.t. } \mathrm{CA}_{\mathrm{G_{compo}}}(\mathrm{t}')_{\mathrm{c}'} = [\![0, 0]\!]$
where $\mathrm{t}' \in ({}^{(\mathrm{p})}\mathrm{t})^{(\mathrm{t})}, \mathrm{t} \neq \mathrm{t}' \}$

8. $C \Leftarrow \{ c \mid \forall c, \text{s.t. } \mathrm{CA}_{\mathrm{G_{compo}}}(t)_c = [\![0, 0]\!]\} \cup$
$\forall \{c' \mid \forall c', \text{s.t. } \mathrm{CA}_{\mathrm{G_{compo}}}(\mathrm{t}')_{\mathrm{c}'} = [\![1, 1]\!]$
where $\mathrm{t}' \in ({}^{(\mathrm{p})}\mathrm{t})^{(\mathrm{t})}, \mathrm{t} \neq \mathrm{t}' \}$

9. **Create** place p_t in $P_{\mathrm{G_{TB}}}$ such that
$\mathrm{V}(\mathrm{p}_t)_{\mathrm{do}^{\wedge}\mathrm{Map(c)}} \Leftarrow 1,$
$\mathrm{V}(\mathrm{p}_t)_{\mathrm{do}^{\wedge}\mathrm{Map(c')}} \Leftarrow 0 \quad \forall \mathrm{c} \in C, \mathrm{c}' \in C'$

10. }

11. }

12. for each $\mathrm{t} \in {}^{(\mathrm{t})}\mathrm{p}_{\mathrm{targ}}$ {

13. if ($\mathrm{p_t} \in P_{\mathrm{G_{TB}}}$) {

14. if $\exists\, \mathrm{t}^{\#}_{\mathrm{cmpl}} \in \mathrm{T_{TB}}$ then x = GetMax(#)+1, else x = 1.

15. if there exists no transition from $\mathrm{p_t}$ to $\mathrm{p_{cmpl}}$ then

16. **Create** transition $\mathrm{t}^{\mathrm{x}}_{\mathrm{cmpl}}$ in $\mathrm{T_{TB}}$ from $\mathrm{p_t}$ to $\mathrm{p_{cmpl}}$
with $\mathrm{CA_{TB}}(\mathrm{t}^{\mathrm{x}}_{\mathrm{cmpl}})_{\mathrm{c}} \Leftarrow [\![\mathrm{z}, \mathrm{z}]\!]$,
$\forall \mathrm{c} \in \mathrm{C_{TB}} \cap \mathit{basis}(\mathrm{k_{targ}})$, z is value of c in $\mathrm{k_{targ}}$

17. }

18. }

19. $\mathrm{P_{visited}} \Leftarrow$ NULL

20. for each $\pi \in \Pi$ {

21. for each $[p \in (\pi \cap P_{\mathrm{G_{compo}}}) - \mathrm{P_{visited}}]$ {

22. $\mathrm{T_{i/p}} \Leftarrow \emptyset, \mathrm{T_{o/p}} \Leftarrow \emptyset$

23. for each $\pi' \in \Pi$) and for each ($p' \in (\pi' \cap P_{\mathrm{G_{compo}}})$, {

24. if ($p = p'$)

25. $\mathrm{T}_{\mathrm{i/p}} \Leftarrow \mathrm{T}_{\mathrm{i/p}} + t$ where t appears immediately before p' in π'

26. $\mathrm{T}_{\mathrm{o/p}} \Leftarrow \mathrm{T}_{\mathrm{o/p}} + t$ where t appears immediately after p' in π'

27. }

28. for each $t \in \mathrm{T}_{\mathrm{i/p}}$ {

29. for each $t' \in \mathrm{T}_{\mathrm{o/p}}$ {

30. $\Pi'' \Leftarrow \{ \pi'' \mid p$ and t are consecutive in $\pi'' \}$

31. if $(|\Pi''| = 1) \quad \pi_1 \Leftarrow \pi''$ where $\pi'' \in \Pi''$

32. if $(|\Pi''| > 1) \quad \pi_1 \Leftarrow \pi''$ such that t immediately precedes p in π''.

33. for each $\mathcal{I} \in I$ where $I \in I_{set}$ and I corresponds to π_1

34. {

35. if $\exists\ \mathrm{t}_p^{\#} \in \mathrm{T}_{\mathrm{TB}}$ then x = GetMax(#)+1, else x = 1.

36. **Create** t_p^x in T_{TB} from p_t to $\mathrm{p}_{t'}$ with
$\mathrm{CA}(\mathrm{t}_p^x)_{\mathrm{c}} \Leftarrow [\![1, 1]\!]\ \forall \mathrm{c}$, s.t. $\mathrm{c} \in (\mathrm{C}_{\mathrm{B}})$, $\mathrm{V}_1(p)_{\mathrm{c}} \Leftarrow 1$;
$\mathrm{CA}(\mathrm{t}_p^x)_{\mathrm{c}} \Leftarrow [\![0, 0]\!]\ \forall \mathrm{c}$, s.t. $\mathrm{c} \in (\mathrm{C}_{\mathrm{B}})$, $\mathrm{V}_1(p)_{\mathrm{c}} \Leftarrow 0$;
$\mathrm{CA}(\mathrm{t}_p^x)_{\mathrm{index}} \Leftarrow \mathcal{I}(2) + [W(\pi_1')](2)$ where
π_1' is the subpath in π_1 such that $H(\pi_1') = H(\pi_1)$, $T(\pi_1') = p$
if $\mathrm{V}_{\mathrm{j}}(p)_{\mathrm{index}} = 1$, for any j.

37. }

38. **Create** t_p^{x+1} in T_{TB} from $\mathrm{p}_{\mathrm{idle}}$ to $\mathrm{p}_{t'}$ with $\mathrm{CA}(\mathrm{t}_p^{x+1}) = \mathrm{CA}(\mathrm{t}_p^x)$

39. }

40. }

41. }

42. $\mathrm{P}_{\mathrm{visited}} \Leftarrow \mathrm{P}_{\mathrm{visited}} + p$

43. }

The procedure from line 5 to line 11 creates a places p_t. The visibility matrix associated with the place created is such that the presence of a token in that place results in a marking which makes the conditions of the taskblock C_{TB} that are associated with the input conditions of its corresponding transition t in the plant model true and the conditions associated with its sibling transitions(and not associated with the transition t) false. In other words the presence of a token in this place p_t in the taskblock will eventually enable the corresponding transition t of the plant (At the same time the sibling transitions are disabled).

Visibility Matrix

In the above procedure, the visibility matrix associated with a place p has all its values equal to 'dc', unless specified otherwise. For example, when a place p is created then the associated visibility matrix is $V(p) = \begin{pmatrix} dc & dc & \cdot & \cdot & dc \end{pmatrix}^T$ unless specified otherwise. Since the size and values in the visibility matrix correspond to C_{TB}, the notation $V(p)_c$ would mean that the row corresponding to condition $c \in C_{TB}$ is being referred.

Finally, the procedure creates the transitions of the taskblock. This can been seen in the lines from 20 to 43 of the procedure. The total number of transitions after the execution of the last function call to BUILDTB() is equal to the place's number of input transitions times the number of output transitions. This is based on the fact that this number is equal to the total number of firing sequences possible involving the place's input transitions and output transitions.

5.4.6 Example

Consider the system model described in the example section 2.2.1 of chapter 2. There is an additional sensor LS that goes high when the motor moves to left. The model of the plant, modelled using the color condition system framework is shown in the figure 5.8. The condition set C_G of the model and the token are

$$C_G = \{HS, LS, M_F, M_B, index\},$$

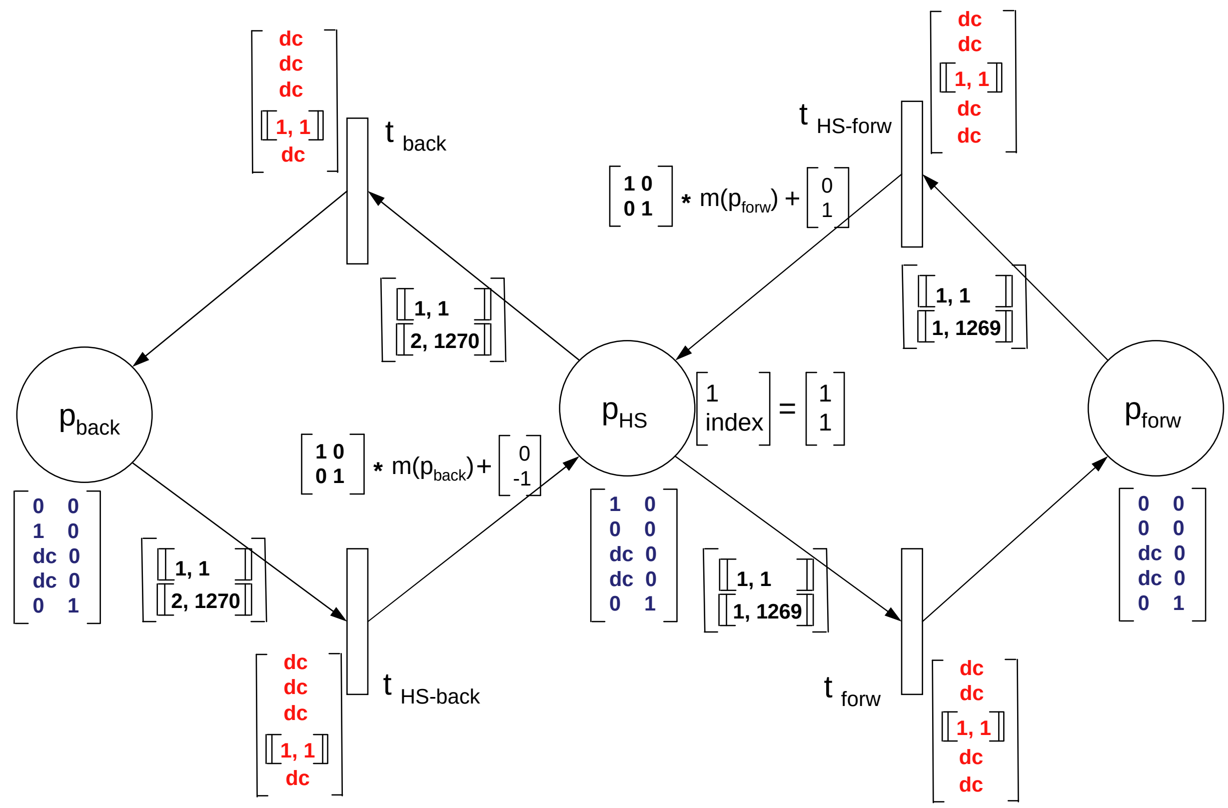

Figure 5.8: Model of a Mercedes Benz Passenger Seat

$$\text{Color Matrix, } K_{token} = \begin{pmatrix} 1 & index \end{pmatrix}^T$$

Let us say that the specification specifies the target state as

$$v_{targ} = \begin{pmatrix} 1 & 0 & 0 & 0 & 720 \end{pmatrix}^T$$

which would give the target color as $k_{targ} = \begin{pmatrix} 1 & 720 \end{pmatrix}^T$. The action block generated to achieve this specification from the procedures described in the previous sections for is shown in the figure 5.9. The following points give a brief description of the steps involved in the synthesis of this taskblock.

1. *CreateAB*() identifies the target place. In the current example, the target place set P_{targ} is $P_{targ} = \{ p_{HS} \}$. Firstly it generates the condition set C_{TB} which corresponds to the taskblock. All the output conditions associated with the plant are copied to C_{TB} while the input conditions are transformed to do^A

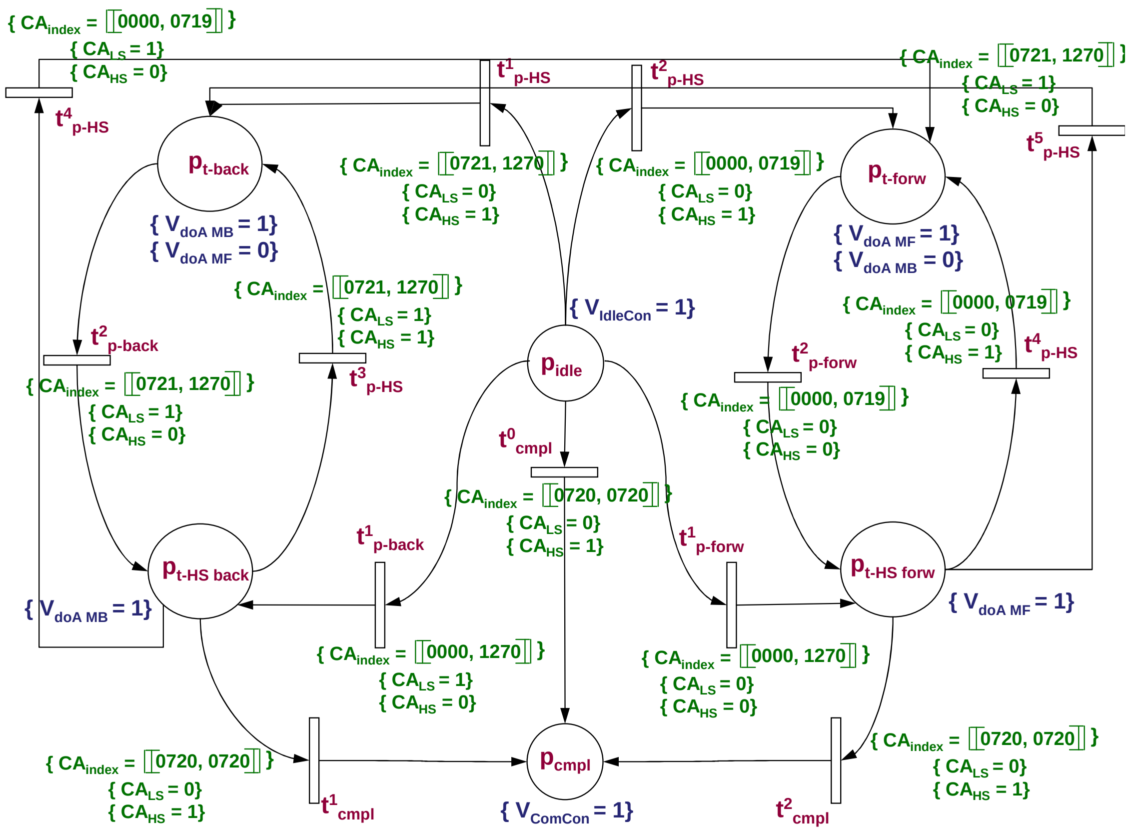

Figure 5.9: Actionblock of model for Mercedes Benz Passenger Seat

conditions.

2. *CreateAB*() makes a function call to *DoAdjacentCycles*() with p_{targ} being p_{HS}.

3. The paths identified by DoAdjacentCycles() in step 2 of the algorithm would be

   - $\Pi_{cycles} = \{ \{ p_{HS}, t_{back}, p_{back}, t_{HS-back}, p_{HS} \},$
     $\{ p_{HS}, t_{forw}, p_{forw}, t_{HS-forw}, p_{HS} \} \}$

   where the first path in Π_{cycles} is negative weighted master cycle and the second path is a positive weighted master cycle.

4. DoAdjacentCycles() analyzes both the above paths. The first path in Π_{cycles}, since being negative weighted yields $I = \begin{pmatrix} \llbracket 1, 1 \rrbracket & \llbracket 721, 1270 \rrbracket \end{pmatrix}^T$ and

the second path would yield $I = \left([\![1, 1]\!] \quad [\![1, 719]\!] \right)^T$

5. BUILDTB() is called by DOADJACENTCYCLES() for each of the above identified paths and I.

6. BUILDTB() checks for the existence of places in the taskblock being generated and since the function is being called for the first time, there exist no places in the taskblock. Hence, BUILDTB() creates the places p_{idle}, p_{cmpl} and also the transition t^1_{cmpl} which connects p_{idle} and p_{cmpl} with the corresponding visibility matrices and condition acceptance intervals respectively.

7. The function BUILDTB() creates exactly one place in the taskblock corresponding to a transition in the plant model. Hence, the function searches for a place in the taskblock which corresponds to the transition it finds, along the paths it is analyzing. If not already created, the function creates its corresponding place and its visibility matrix. For example, the place p_{t-back} in the taskblock corresponds to the transition t_{back} which is along the first path in Π_{cycles} shown in step 3. The process is repeated for each place of the entire path shown. Notice in the generated taskblock (5.9) that the visibility matrix of p_{t-back} has $do^A ML$ true and $do^A MR$ false, since the condition 'ML' corresponds to t_{back} and the condition 'MR' corresponds to t_{forw}, its sibling transition.

8. The function also creates several transitions in the taskblock corresponding to a single place in the plant as discussed in the final paragraph of section 5.4.5. Corresponding to the place p_{HS}, it can be seen in the figure 5.9 that there are four transitions in the taskblock. The function DOADJACENTCYCLES() returns the control back to CREATEAB().

9. The absence of any remote cycles and since all the colors of the place p_{targ}, p_{back} and p_{forw} have been analyzed, the function calls for the DOREMOTECY-

CLES() and DOREMAININGCOLORS() are ignored by CREATEAB().

These are the steps involved in building an actiontype taskblock.

Chapter 6

Conclusion and Future work

6.1 Conclusion

In the current thesis, condition systems were reintroduced and a summary of its results were stated. The concept of color of a token and modeling framework of color condition systems was defined. The primary contribution has been development of algorithms to generate the action type task blocks. The algorithms described in chapter 5 generate the taskblock given the plant modeled in color condition system framework defined in the chapter 3 under the System Structure Assumption(SSA) and Non-Overlapping Cycles assumption (NOC) defined in chapter 5.

6.2 Future Work

The goal of the thesis was to build a foundation for color condition systems. Since the current topic of research is still in the earlier stages there is a lot of scope to optimize the algorithms developed in this document. Also the assumptions made in this thesis can be relaxed to allow the modeler to model a broader range of complex systems. The non-overlapping cycles assumption made in this document is not a requirement. An important direction would be to extend this algorithms to non-overlapping cycles. Also, the algorithms described in this thesis work for those tokens whose colors are two-dimensional. Hence, research may be pursued to work for larger dimensioned color matrices. This will introduce more information carrying capabilities in the net.

Development of techniques to build state observers for systems with unidentifiable states and to counteract forbidden states would be another important direction future research could pursue.

Appendix

Algorithm to determine possible paths between two places

In this section we show a procedure that explores the plant $\mathrm{G}_{\mathrm{compo}}$ and generates the set $\Pi_{(n,n_s)}$ containing all the paths from node n to n_s. This is description of a possible implementation of definition 5.2 defined in chapter 5. An example is used to illustrate the working of the procedure.

Given a source node n, target node n_s and the component net $\mathrm{G}_{\mathrm{compo}}$ such that n, $n_s \in \mathrm{G}_{\mathrm{compo}}$, the procedure SEEKPATH() generates all the possible set of paths $\Pi_{(n,n_s)}$ from n to n_s by exploring the plant $\mathrm{G}_{\mathrm{compo}}$

The procedure starts at the node n, selects one of the possible output nodes of n' (child nodes) and explores as far as possible along each branch before backtracking. A node is a parent node to all its output nodes, called child nodes and since the procedure starts at n, we call it as the root node. If the node n_s is found, the branch is appended to the set $\Pi_{(n,n_s)}$ and the algorithm backtracks to the corresponding parent node to investigate the other possible branches. If the branch being investigated encounters a cycle that was already visited in the branch, the procedure stops exploring the branch further and backtracks to choose an alternative branch. The search stops when all the paths get exhausted and there are no more branches left to be investigated.

In the following procedure which is a recursive procedure the function GETOPTRANSITIONS(n, $\mathrm{G}_{\mathrm{compo}}$) in the line 11 retrieves all the output transitions of the place n in the plant $\mathrm{G}_{\mathrm{compo}}$. Similarly function GETOPPLACES(n, $\mathrm{G}_{\mathrm{compo}}$) in the line 12 retrieves all the output places of the transition 'n' in the plant $\mathrm{G}_{\mathrm{compo}}$. The function on the line 17 ensures that the branch under consideration will not search further since there exists a cycle that has been visited twice in it.

Procedure: SEEKPATH(n, n_s, π, $\Pi_{(n,n_s)}$)

1. **Assign** $N \Leftarrow$ NULL
2. if (($n = n_s$) AND ($n \in \pi$))
3. {
4. $\pi \Leftarrow \pi + n$
5. $\Pi_{(n,n_s)} \Leftarrow \Pi_{(n,n_s)} + \pi$
6.   RETURN($\Pi_{(n,n_s)}$)
7. }
8. else
9. {
10. $\pi \Leftarrow \pi + n$
11.   if ($n \in P_{G_{compo}}$) $N \Leftarrow$ GETOPTRANSITIONS(n, G_{compo})
12.   if ($n \in T_{G_{compo}}$) $N \Leftarrow$ GETOPPLACES(n, G_{compo})
13.   for each $n' \in N$
14.   {
15.   if (($n' \in T_{G_{compo}}$) And
            (Number of times n' was already visited in $\pi > 2$))
16.   {
17.   if (DOESANYCYCLERECUR(π) = TRUE)
18.   RETURN($\Pi_{(n,n_s)}$)
19.   }
20. $\Pi_{(n,n_s)} \Leftarrow$ SEEKPATH(n', n_s, π, $\Pi_{(n,n_s)}$)
21.   }

22. RETURN($\Pi_{(n,n_s)}$)

23. }

Theorem .1 SEEKPATH() generates the set $\Pi_{(n,n_s)}$.

Proof: The procedure explores along a branch until

1. The destination node n_s is located (at line 6).

   This return statement would be true if the condition at line 2 is satisfied. SEEKPATH() is a recursive procedure and when it is called by itself n is replaced by n', where n' is an output node of n (see line 13). $n = n_s$, implies that the destination node is located. Hence SEEKPATH() should stop exploring further.

   When the procedure is called for the first time, π and Π are empty. If the task is to determine all the cycles of a node n, then n and n_s are same. In this case $n = n_s$, would not imply that the destination node has been located. Hence, the second condition $n \in \pi$ at line 2 ensures that further exploration of the branch is not halted.

2. No further exploration is possible. At line 22.

   The task of procedure SEEKPATH() would have been accomplished when all the paths along the output nodes n' of n are explored. The lines from 13 to 21 ensure that all the output nodes of n have been explored. So further exploration is halted at line 22.

3. The existence of a subcycle π' (where $\mathrm{n_s} \notin \pi'$) in a path π more than once would never terminate the procedure SEEKPATH() since it searches for the destination node n_s with out any success forever. The procedure avoids this condition at line 17.

   Since the system is a state graph (SSA 5.2), for any transitions t, t' ${}^{(p)}t = {}^{(p)}t'$ and $|{}^{(p)}t| = 1$. If a transition t is visited twice along a path, then according to the definition of cycle (5.3.1) the subpath π' between the first occurrence of t to the subsequent occurrence of t represents a cycle. Any subsequent occurrence of t again in the path π might result in the repetition of the subcycle π'. In order to assert if a repetition exists

- Let the transition t occur m ($m > 2$) times along the path π.
- Let the sub paths between any two subsequent occurrences of t be π_1, π_2 $\cdots$ π_{m-1}.
- Assume that the same subcycle π' repeats in π_1 and π_{m-l}.

If the above assumption that the same subcycle π' repeats in π_1 and π_{m-l} is true, then if any transition $t'' \in \pi_1$ implies $t'' \in \pi_{m-1}$ or vice-versa. Hence, the set of transitions appearing in π_1 is a subset of π_{m-1} or viceversa.

In the flowchart of the sub-procedure DOESANYCYCLERECUR() 1 the the loop at bottom right extracts the transition sets with transitions that would be found between any two subsequent occurrences of t. These sets are compared between each other to determine if any one is a subset of the other in the top left of the flowchart. If comparison is successful, a repetition of a subcycle is asserted and the sub-procedure returns TRUE. The procedure SEEKPATH() then terminates any further investigation along the branch (line 18).

Hence, $\forall\ \pi \in \Pi_{(n,n_s)}$, there exists no sub-cycle π' along the path π that is visited twice in it.

From the above, we see that the procedure explores along every output node of n. Hence, there exists no path from n to n' in G_{compo} that remains unexamined i.e., there exists no path π' from 'n' to 'n_s' in the net G_{compo} which does not belong to the set $\Pi_{(n,n_s)}$.

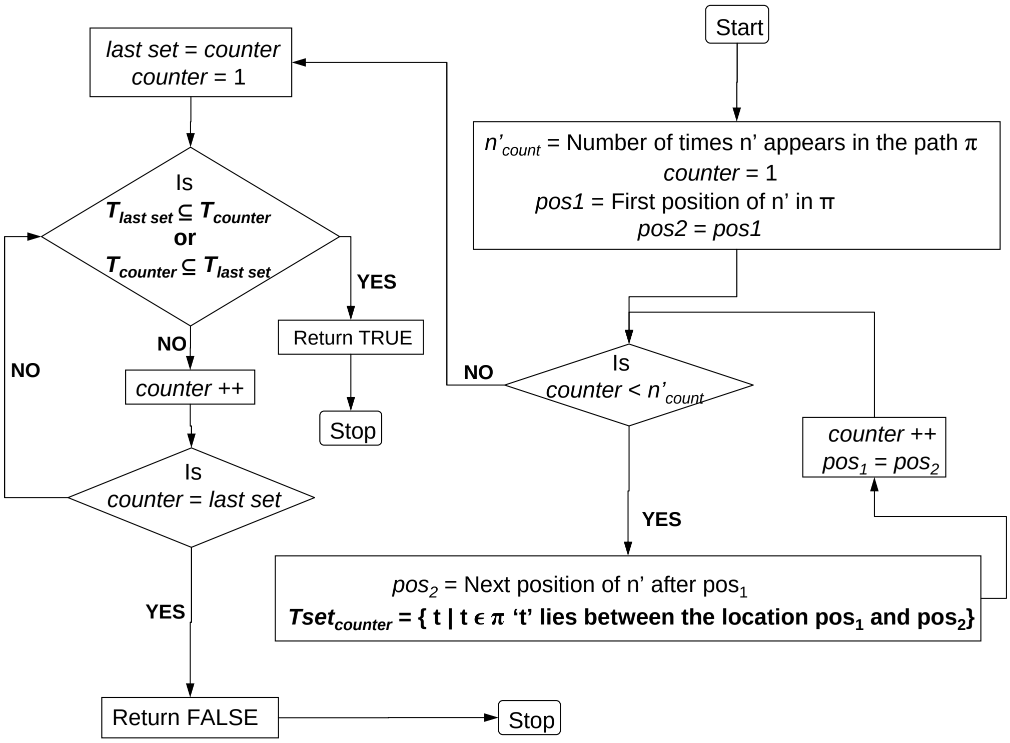

Figure 1: DoesAnyCycleRecur(n′)

Example

Consider the example shown in the figure 2. We use this example only to illustrate the various paths a token can trace between any given pair of places. Let us determine all possible paths, the token could trace between place P_2 and P_2 (i.e., all the possible cycles of P_2) with the help of procedure SEEKPATH(). The figure 3 shows the entire working of the procedure indicating the branches it would search and the conditions that would terminate the search process along a branch. The procedure starts at P_2 and since the search is to find place P_2, the branches

- P_2, T_2, P_1, T_1, P_2
- P_2, T_3, P_3, T_7, P_5, T_6, P_2
- P_2, T_3, P_3, T_7, P_5, T_5, P_4, T_4, P_3, T_7, P_5, T_6, P_2
- P_2, T_3, P_3, T_7, P_5, T_5, P_4, T_8, P_6, T_9, P_4, T_4, P_3, T_7, P_5, T_6, P_2

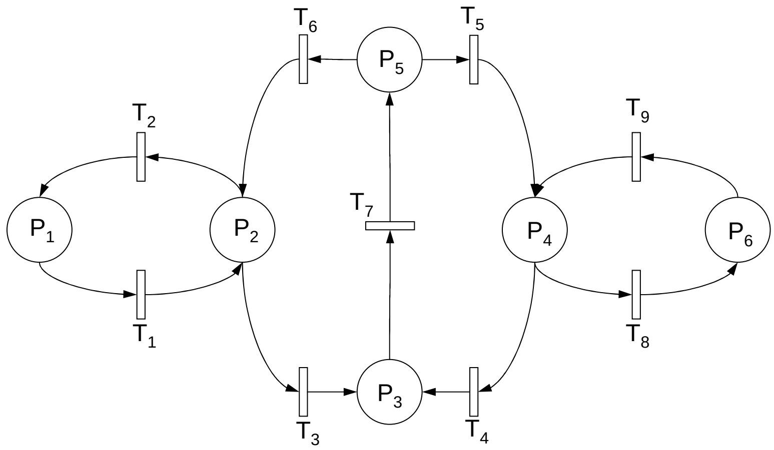

Figure 2: Example to illustrate the working of the procedure SeekPath()

terminate as soon as P_2 is found. Now let us consider the branch

$$P_2, T_3, P_3, T_7, P_5, T_5, P_4, T_4, P_3, T_7, P_5, T_5, P_4, T_4, P_3, T_7.$$

We observe that T_7 repeated itself third time (> 2) in the above branch. If allowed to trace the above branch, the token after visiting T_7 for the first time would visit T_5 and T_4 before visiting T_7 for the second time. We also observe that the token would visit the same set of transitions (i.e., traces the same path) again before T_7 is visited for the third time. Since the algorithm has already identified this path, further exploration of the branch is not necessary. Therefore the procedure stops investigating the branch further.

The appearance of T_7 the third time would trigger the sub-function *DoesAnyCycleRecur*() (refer the line 15 in the algorithm). In the flowchart shown in figure 1, the loop at bottom right extracts the transition sets containing those transitions which the token would visit between two successive occurrences of T_7. In the current branch two such transition sets are possible. They are

- $\text{Tset}_1 = \{ T_5, T_4 \}$
- $\text{Tset}_2 = \{ T_5, T_4 \}$

The loop at the top left compares these sets to determine if either of them is a subset to the other. If the comparison is true, then according to the lemma **??** we

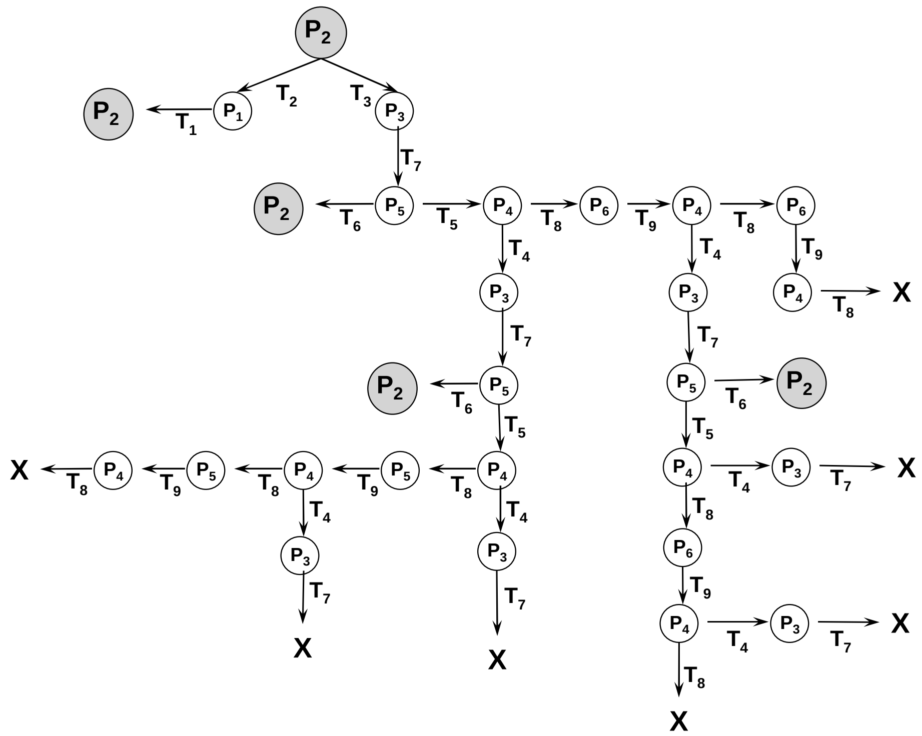

Figure 3: SeekPath Tree

have a cycle repeating itself in the path (as seen in this example) which makes *DoesAnyCycleRecur*() TRUE. Hence, the algorithm SEEKPATH() stops exploring the branch further. (refer line 17 in the algorithm).

In the branch,

P_2, T_3, P_3, T_7, P_5, T_5, P_4, T_8, P_6, T_9, P_4, T_4, P_3, T_7, P_5, T_5, P_4, T_8, P_6, T_9, P_4, T_8 the transition T_8 repeats itself more than twice. Hence, the transition sets considered in this branch are $Tset_1 = \{ T_9, T_4, T_7, T_5 \}$ and $Tset_2 = \{ T_9 \}$ where $Tset_2$ is a subset of $Tset_1$. Similarly is the case with the other branches shown in the figure.

Target acceptance range

In this section we present a flow chart which captures the token acceptance range for a path π from a place located at position n. This is implementation of the definition 5.3 in chapter 5.

The flow chart is self-explanatory. It starts to look at each and every place starting from the place located at position n. Let this place be p_n. It captures the target acceptance interval of the place with respect to its output transition and projects it back to place P_n based on the token assignment functions existing between place p_n and the place under consideration. Every time the algorithm performs this task it would only consider the intersection of the token acceptance interval of P_n with respect to its O/P transition and the projected token acceptance interval, as the future token acceptance interval of P_n with respect to its O/P transition. Once all the places along the path are examined then the token acceptance interval of of P_n with respect to its O/P transition is the required TAR.

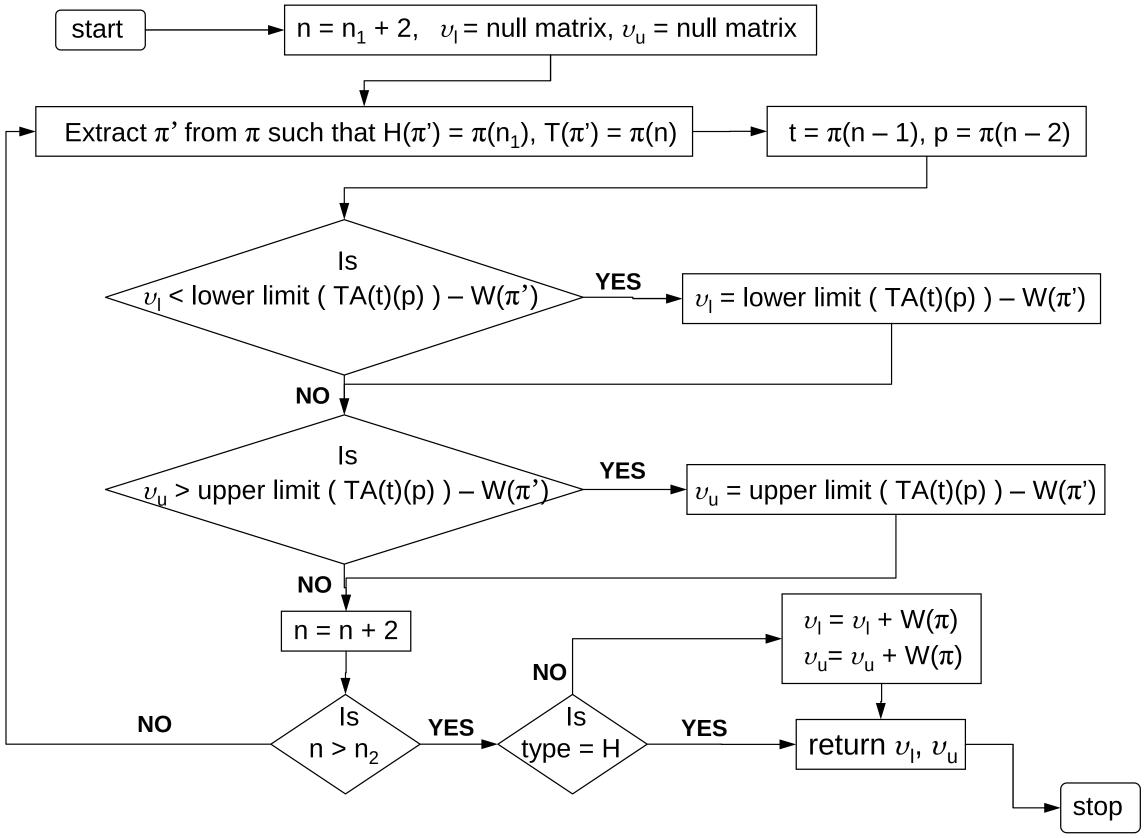

Figure 4: Flow chart of TAR()

Index

Bibliography

[Holl00] Lawrence E. Holloway, Xiaoyi Guan, Ranganathan Sundaravadivelu, Jeff Ashley, Jr. Automated Synthesis and Composition of Taskblocks for Control of Manufacturing Systems. IEEE TRANSACTIONS ON SYSTEMS AND CYBERNETICS-PART B: CYBERNETICS, 30(5), October 2000.

[Holl01] Lawrence E. Holloway, Jeff Ashley, Jr. An equivalent LTL/Kripke Structure for the condition sequence/condition system model 7TH WORKSHOP ON DISCRETE EVENT SYSTEMS (WODES2004), FRANCE, September 2004.

[Holl02] Lawrence E. Holloway, R. Sunderavadivelu. Task Blocks for Synthesis of Controllers for Condition Systems. 1998 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNATICS , San Diego, October 1998.

[Holl03] Lawrence E. Holloway, Xiaoyi Guan, Ranganathan Sundaravadivelu, Jeff Ashley, Jr. State Observer Synthesis for a Class of Condition Systems. 5TH WORKSHOP ON DISCRETE EVENT SYSTEMS (WODES2000), BELGIUM, August 2000.

[Holl04] Lawrence E. Holloway, Jeff Ashley, Jr. Elaborative Orderings of Condition Languages. IN PROCEEDINGS OF 1998 IEEE CONFERENCE ON DECISION AND CONTROL, TAMPA , December 1998.

[Jen98] Kurt Jensen Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use, VOL-I EATCS MONOGRAPHS ON THEORETICAL COMPUTER SCIENCE, SPRINGER-VERLAG, 1992.

[Jen98] Kurt Jensen Coloured Petri Nets, A High-level Language for System Design and Analysis. ADVANCES IN PETRI NETS, LECTURE NOTES IN COMPUTER SCIENCE VOL 483, SPRINGER-VERLAG, 1991.

[Zoh01] Zohar Manna, Amir Pnueli. The Temporal Logic of Reactive and Concurrent Systems, Specification.

[Cain95] P. E. Caines, Y. J. Wei The Hierarchical Lattices of a Finite Machine. SYSTEMS & CONTROL LETTERS 25, PAGES 257 – 263, 1995.

[Cain97] Peter E. Caines, Vineet Gupta, Gang Shen The Hierarchichal Control of ST-Finite State machines SYSTEM & CONTROL LETTERS 32, PAGES 185-192, 1997.

[Gries00] M. Gries Modeling a Memory Subsystem with Petri Nets HARDWARE DESIGN AND PETRI NETS, KLUWER ACADEMIC PUBLISHERS, March 2000.

[Dwy95] Matthew B. Dwyer, Lori A. Clarke, Kari A.Nies A Compact Petri Net Representation for Concurrent Programs. IN PROCEEDINGS OF THE 17TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, PAGES 147–157, April 1995

[URL] www.automotive-uml.com/mc/challenge/specification

Vita

Date and place of birth

April - 19 - 1979, Andhra Pradesh, India.

Educational institutions attended and degrees awarded

Bachelor of Technology, Valluripalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, JNTU, Hyderabad, INDIA.

Major: Mechanical Engineering.

Professional Positions held

1. October 2004 - Present

   RCI Engineer, Paoli, Inc.,

2. August 2001 - October 2004

   Research Assistant, UK Center for Manufacturing, University of Kentucky.

Honors

1. August 2001. Awarded a full scholarship (a Research Assistantship) by the Department of Manufacturing Engineering, University of Kentucky, to pursue Master's Degree.