Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Christian Urland

Ereignisdiskrete Steuerungssynthese und Codegenerierung am Beispiel einer Fertigungszelle

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Christian Urland

Ereignisdiskrete Steuerungssynthese und Codegenerierung am Beispiel einer Fertigungszelle

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. -Ing. Florian Wenck
Zweitgutachter : Prof. Dr. -Ing. Dipl. -Kfm. Jörg Dahlkemper

Abgegeben am 29. März 2012

Christian Urland

Thema der Masterthesis

Ereignisdiskrete Steuerungssynthese und Codegenerierung am Beispiel einer Fertigungszelle

Stichworte

Steuerungssynthese, Supervisory Control Theory (SCT), ereignisdiskrete Systeme, DESTool

Kurzzusammenfassung

Diese Arbeit beinhaltet die Steuerungssynthese für die Automatisierung einer Fertigungszelle. Dafür wird die Fertigungszelle als ein ereignisdiskretes System modelliert. Die Steuerung wird nach der Supervisory Control Theory synthetisiert und implementiert. Zur Implementierung wird ein Codegenerator entwickelt, der den Code für eine Speicherprogrammierbare Steuerung erzeugt. Letztendlich wird geprüft, ob die geforderte Steuerungsaufgabe erfüllt wird.

Christian Urland

Title of the paper

Supervisor synthesis and code generation of a discrete event manufacturing cell

Keywords

Control synthesis, Supervisory Control Theory (SCT), discrete event systems, DESTool

Abstract

Inside this report the control synthesis of a Supervisor for a manufacturing cell is described. Therefore the manufacturing cell is modeled as a discrete event system. The Supervisory Control Theory is responsible for the synthesis of the controller. For the implementation of the Supervisor a code generator is developed. It generates the program code for a programmable logic controller. Finally it is checked if the required control task is satisfied.

Danksagung

An dieser Stelle möchte ich mich bei Herrn Professor Wenck für die Bereitstellung des Themas und die umfassenden Betreuung bedanken. Ein weiterer Dank geht an Herrn Dahlkemper, der sich als Zweitprüfer zur Verfügung gestellt hat.

Für die Korrektur möchte ich Jürgen Timmann und Kathrin Urland meinen Dank aussprechen, sowie Herrn Huss, der mich stets im Labor unterstützt hat.

Mein persönlicher Dank geht an Selina Timmann, die immer für mich da ist, mich besonders in schwierigen Zeiten unterstützt und mir neuen Mut schenkt.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

A	Adjazenzmatrix
G	Generator
$G_\times$	Produktautomat
K	Formale Spezifikation
$K^{\downarrow C}$	Infimale präfix-abgeschlossene steuerbare Obersprache
$K^{\uparrow C}$	Supremale steuerbare Teilsprache
$L(G)$	Reguläre Sprache von G
$L_m(G)$	Markierendes Verhalten von G
P, $P^{(-1)}$	Natürliche Projektion, inverse natürliche Projektion
S	Supervisor
S/G	Gesteuertes System (G gesteuert von S)
X	Zustandsmenge
X_m	Menge der markierten Zustände
Σ	Ereignisalphabet
Σ^*	Kleene-Hülle
Σ_{CE}	Gemeinsame Ereignisse
Σ_{PE}	Private Ereignisse
Σ_c	Menge der steuerbaren Ereignisse
Σ_{uc}	Menge der nicht steuerbaren Ereignisse
δ	Zustandsübergangsfunktion
ϵ	Sequenz ohne Ereignis
$\mathbb{N}^+$	Menge der nichtnegativen ganzen Zahlen
$\overline{L}$	Präfix-Hülle einer Sprache
σ	Ereignis
k	Diskreter Zähler
t	Kontinuierliche Zeit
x_0	Anfangszustand
$x_{(k)}$	Zustand zum diskreter Zeitpunkt k

A	Ausgang der SPS
AC(G)	Erreichbarer Teil des Generators G
AWL	Anweisungsliste
CoAc(G)	Ko-erreichbarer Teil des Generators G
DB	S7- Datenbaustein
DES	Discrete Event Systems
E	Eingang der SPS
FB	S7- Funktionsbaustein
FC	S7- Funktion
FUP	Funktionsplan
HU	Handling Portal
KOP	Kontaktplan
LA	Lagereinheit
LI	Transporteinheit (Linearachse)
LS	Lichtschranke
R	Robotereinheit
RFID	Radio-frequency identification
SA	Schwenkarm
SCL	Structured Control Language
SCT	Supervisory Control Theory
SPC	Strict Product Composition
STR	Steuerbares Ereignis in der SPS
SYPC	Synchronous Product
Trim(G)	Erreichbarer und ko-erreichbarer Teil des Generators G
WS	Werkstück

1. Einführung und Zielsetzung

1.1. Einführung und Motivation

Grundlage der Automatisierung von Systemen ist der Steuerungsentwurf. Ohne die Programmierung der Automatisierungskomponenten kann keine Anlage ihre Aufgabe erfüllen. In der industriellen Praxis werden Steuerungen intuitiv entworfen und anschließend verifiziert. Die Steuerung wird erst bei der Inbetriebnahme getestet und die Überprüfung der Korrektheit erfolgt empirisch. Beim Entwurfsvorgang werden die Vorgänge Steuerungsentwurf, Test und Nachbesserung solange wiederholt bis das Verhalten empirisch korrekt ist.

Im Gegensatz dazu liefert die Steuerungssynthese ein beweisbar korrektes Ergebnis nach dem ersten Iterationsschritt im Entwicklungsprozess. Es kann direkt aus der synthetisierten Steuerung der Code für eine Speicherprogrammierbare Steuerung extrahiert werden. In dieser Arbeit wird eine solche Steuerungssynthese für ein ereignisdiskretes System anhand einer Anlage durchgeführt.

Dafür werden im zweiten Kapitel in die Grundlagen der ereignisdiskreten Systeme und der Supervisory Control Theory eingeführt. Im dritten Kapitel werden der Aufbau und die Funktion der betrachteten Fertigungszelle beschrieben. Das vierte Kapitel beinhaltet die Entscheidungsfindung zu den verwendeten Beschreibungsformen, Softwaretools und geeigneten strukturellen Ansätzen. Die Strecke der Fertigungszelle wird im fünften Kapitel modelliert und auf dieser Grundlage die Spezifikationen und die Supervisor im sechsten Kapitel entwickelt. Kapitel 7 erläutert die Entwicklung des Codegenerators DES2IEC. Anschließend werden die Realisierung und Inbetriebnahme der Steuerung erläutert und die Betriebsergebnisse präsentiert. Im Abschluss der Arbeit werden die Ergebnisse zusammengefasst, ein Ausblick gegeben und ein Fazit gezogen.

1.2. Aufgabenstellung

In dieser Arbeit ist eine ereignisdiskrete Steuerung für eine Fertigungszelle zur Realisierung eines Stückgutprozesses zu entwerfen. Der Stückgutprozess (siehe Abbildung 1.1) beinhaltet die Lagerung, den Transport und die Sortierung von Werkstücken unterschiedlicher Far-

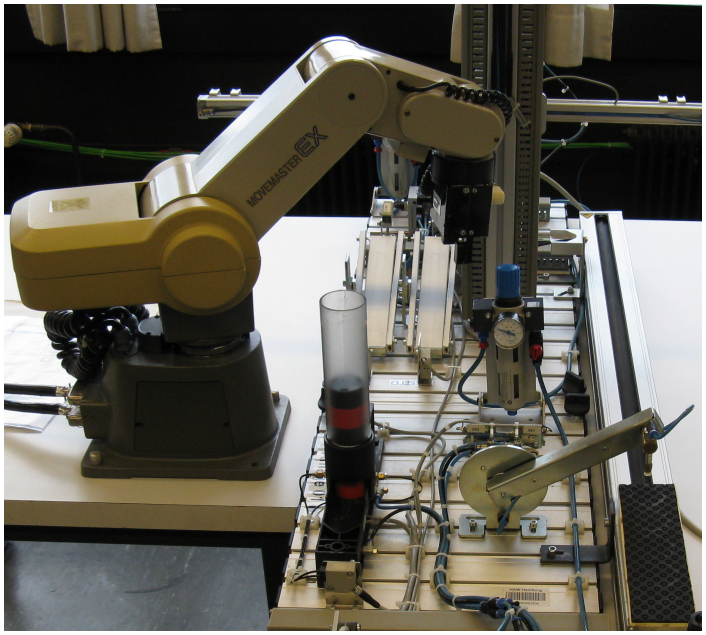


Abbildung 1.1.: Fertigungszelle zur Realisierung eines Stückgutprozesses

ben. Die Funktionalität und Sicherheit der entworfenen Steuerung ist formal zu analysieren und zu verifizieren. Dafür sind geeignete Verfahren und Tools auszuwählen und anzuwenden. Für die Steuerungssynthese ist die von W.M. Wonham entwickelte Supervisory Control Theory (SCT) zu verwenden und eine dem Ansatz entsprechende Modellbildung der Anlage und der Spezifikation durchzuführen. Die Steuerung ist auf einer Speicherprogrammierbaren Steuerung zu implementieren. Abschließend ist zu testen, ob die Automatisierungsaufgabe erfüllt wird.

1.3. Zielsetzung

Ziel dieser Arbeit ist zu zeigen, dass eine erfolgreiche Anwendung der SCT auf ein komplexes gekoppeltes ereignisdiskretes System möglich ist und, dass der Entwurfsprozess durch Formalisierung für derartige Steuerungen einfacher, sicherer und letztendlich kostengünstiger gemacht werden kann. Die Anlage muss die Steuerungsaufgabe korrekt, vollständig und ohne zu blockieren ausführen.

2. Ereignisdiskrete Systeme

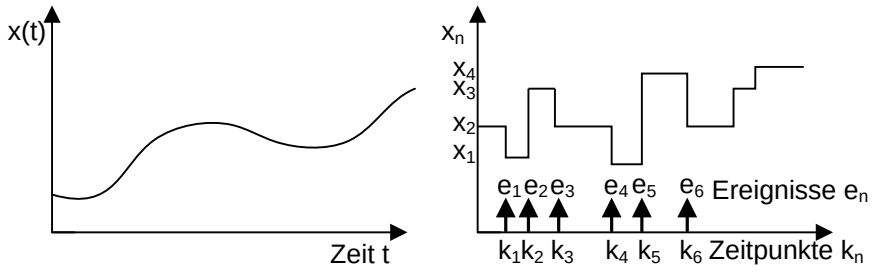
In diesem Kapitel werden die Grundlagen der ereignisdiskreten Systeme (DES, engl. Discrete Event Systems) und der Supervisory Control Theory erläutert. Es werden zunächst allgemein die Systemtheorie von DES und deren Beschreibungsformen vorgestellt. Dazu werden notwendige mathematische Verfahren wie die Komposition und die Erreichbarkeitsanalyse von Automaten erläutert. Am Ende wird die für die Steuerungsimplementierung erforderliche Codeerzeugung für Speicherprogrammierbare Steuerungen erläutert und die verwendeten Softwaretools kurz beschrieben.

2.1. Begriffe und Definitionen

„Ein System beschreibt allgemein die Gesamtheit von Objekten, die sich in einem ganzheitlichen Zusammenhang befinden und durch die Wechselbeziehungen untereinander gegenüber ihrer Umgebung abzugrenzen sind“ [15]. Sie sind sowohl in der Natur, als auch in der vom Menschen konstruierten Welt zu finden. Das Verhalten des Systems kann durch eine Systemtheorie beschrieben und analysiert werden. Diese wird von dem Anwender je nach System und Abstraktion ausgewählt [9], [47].

Systeme werden zur Erklärung komplexer Sachverhalte und Zusammenhänge verwendet. Die Systemtheorie baut dabei auf formale Abstraktionen der Systeme, sogenannte Modelle auf. Diese Modelle bilden, je nach Abstraktion, mehr oder minder das Verhalten des Systems nach. Der Begriff System und die dazu gehörige Systemtheorie sind interdisziplinär und werden in vielen Wissenschaften eingesetzt. In der Technik werden in der Regel dynamische Systeme betrachtet. Die dort auftretenden Signale werden häufig durch physikalische Größen dargestellt. Diese Größen sind abhängig von der kontinuierlichen Zeit t mit $t \in \mathbb{R}$[26] (siehe Abbildung 2.1a).

Die ereignisdiskrete Systemtheorie abstrahiert diese kontinuierlichen Signale und stellt sie durch eine Folge von Ereignissen dar (siehe Abbildung 2.1b). Die Ereignisse werden nicht auf eine kontinuierliche, sondern auf eine diskrete Achse aufgetragen, wofür nur noch die Werte der Menge $\mathbb{N}^+$ der nichtnegativen ganzen Zahlen verwendet werden. Das Auftreten der Ereignisse wird durch den diskreten Zähler k gezählt [26]. Durch das Auftreten eines Ereignisses kann ein Zustandswechsel auftreten. Ein Zustand beschreibt den Status zum



a) Kontinuierliches System b) Ereignisdiskretes System

Abbildung 2.1.: Trajektorien von kontinuierlichen und ereignisdiskreten Systemen

diskreten Zeitpunkt k und wird mit $x_{(k)}$ bezeichnet. Durch die Information, die der Zustand über die Anlage enthält, kann der nächste Zustand ermittelt werden. Durch das Ereignis wird der Zustandswechsel ausgelöst. Das Ereignis tritt dabei asynchron und ohne zeitliche Dauer auf. Ein Zustandsübergang wird somit nicht durch die Zeit, sondern von dem Ereignis selbst verursacht. Der Zustandsübergang ist demnach ereignisgesteuert.

Ein ereignisdiskretes System kann somit als ein dynamisches System definiert werden, dessen Zustände in einem diskreten Zustandsraum zusammengefasst werden können. Die Zustandswechsel werden durch asynchron auftretende Ereignisse ausgelöst.

Sprachentheoretische Grundlagen
Zur Behandlung von ereignisdiskreten Systemen sind Grundlagen der Sprachentheorie notwendig. Es folgt eine Auflistung von Begriffen, Definitionen und Operationen aus der Sprachentheorie. Sie sind aus [9] und [49] entnommen.

- Σ ist eine endliche nichtleere Menge von paarweise verschiedenen Ereignissen. Es wird im weiteren Alphabet genannt.
- Strings sind endliche Sequenzen von Ereignissen (Konkatenation) $\sigma_1\sigma_2\sigma_3 \ldots \sigma_k$.
- ϵ ist eine Sequenz ohne Ereignisse mit $\epsilon \notin \Sigma$ und $\{\epsilon\} \neq \emptyset$.
- Die Kleene-Hülle von Σ ist definiert als $\Sigma^* = \{\epsilon\} \cup \Sigma^+$.
- Die Konkatenation von Strings ist die Abbildung $cat(\epsilon, s) = cat(s, \epsilon) = s$, $s \in \Sigma^*$ und $cat(s, t) = st$, für $s, t \in \Sigma^+$.
- Es sei $s = cat(t, \sigma) = abab$, mit $t = \{aba\}$ und $\sigma = \{b\}$, dann ist t Präfix von σ.
- Die Präfix-Hülle einer Sprache $L \subseteq \Sigma^*$ enthält die Strings aus L und alle ihre Präfixe und ist definiert durch $\overline{L} = \{s \in \Sigma^* \mid \exists t \in \Sigma^* (st \in L)\}$.

- Eine Sprache ist präfix-geschlossen, wenn $L = \overline{L}$.

2.2. Automaten als Beschreibungsform ereignisdiskreter Systeme

In diesem Abschnitt wird eine mögliche Beschreibungsform für ereignisdiskrete Systeme vorgestellt und erläutert. Die Beschreibung von ereignisdiskreten Systemen durch deterministische Automaten gilt als die einfachste Beschreibungsform. Eine weitere Form sind die Petri-Netze. Die von dem deutschen Mathematiker und Informatiker Carl Adam Petri entwickelten Petri-Netze sind eine Erweiterung der Automaten. Sie werden in dieser Arbeit jedoch nicht weiter behandelt.
Automaten dienen zur abstrakten Darstellung von komplexen ereignisdiskreten Systemen. Durch den Zustand wird der „Status“ eines Systems repräsentiert. Die hier betrachteten deterministischen Automaten werden in der Literatur auch als Standardautomaten [26] bezeichnet. In dieser Arbeit werden sie fortan Automaten bzw. Generatoren genannt. Generatoren zur Beschreibung ereignisdiskreter Systeme wurden durch Ramadge und Wonham in [49] eingeführt. Im Gegensatz zu einem passiven Erkenner erzeugt der Generator aktiv Ereignisse, die zu Zustandsübergängen führen können. Sie werden durch das Quintupel

$$G = (X, \Sigma, \delta, x_0, X_m) \tag{2.1}$$

beschrieben.

Die Menge X stellt die diskrete Zustandsmenge dar. Die Zustände werden in der Regel durchnummeriert, so dass $X = 1, 2, 3, \ldots, N$ ist. Σ gibt die Menge der möglichen Ereignisse an und bildet somit die Ereignismenge bzw. den Ereignisraum. Die Ereignismenge beinhaltet alle Ereignisse $\sigma \in \Sigma$ [26]. Die Zustandsübergangsfunktion wird mit δ bezeichnet. Mit Hilfe dieser Funktion wird das dynamische Verhalten und somit die Zustandsübergänge beschrieben. Sie ist nach [49] wie folgt definiert

$$\delta : X \times \Sigma \to X. \tag{2.2}$$

So ordnet die Funktion δ jedem Zustand $x \in X$ mit jedem Ereignis $\sigma \in \Sigma$ einen Folgezustand zu, so dass gilt

$$x' = \delta(x, \sigma), x' \in X. \tag{2.3}$$

Da die Funktion δ jedem Zustand X einen eindeutigen Nachfolgezustand X' zuordnet, wird der Automat deterministisch genannt. Kann ein Verhalten des Automaten nicht eindeutig vorhergesagt werden, so muss ein nicht deterministischer Automat verwendet werden. In

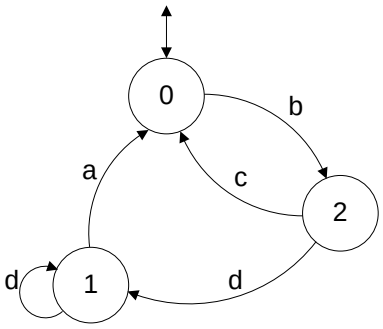


Abbildung 2.2.: Generator mit $X = \{0, 1, 2\}$, $\Sigma = \{a, b, c, d\}$, $x_0 = 0$ und $x_m = 0$

diesem Fall wird anstatt einer Zustandsübergangsfunktion, eine Zustandsübergangsrelation verwendet. Es wird somit eine Menge an möglichen Folgezuständen definiert.

Durch x_0 wird der Anfangszustand beschrieben. Er bezeichnet den Anfangs- bzw. den Initialisierungszustand in dem sich der Automat zum diskreten Zeitpunkt $k = 0$ befindet. Er wird im Graphen durch einen Pfeil auf den Zustand repräsentiert. X_m bezeichnet die Menge an Endzuständen bzw. markierten Zuständen. Es gilt für X_m immer $X_m \subseteq X$. Endzustände werden durch einen Kreis im Zustand oder durch einem vom Zustand weg zeigenden Pfeil repräsentiert.

Adjazenzmatrix
Die Zustandsübergänge können, neben der grafischen Darstellung, auch als Tabelle bzw. Matrix dargestellt werden. Diese Matrix wird als Adjazenzmatrix A bezeichnet. Die Matrix enthält genau $N \times N$ Elemente, wobei N die Anzahl der Zustände ist. Das Element g_{ij} ist genau dann gleich σ, wenn die Übergangsfunktion δ für den Übergang von j nach i definiert ist

$$x_i = \delta(x_j, \sigma)!. \tag{2.4}$$

Ist δ von j nach i nicht definiert

$$x_i = \neg\delta(x_j, \sigma)!, \tag{2.5}$$

so erhält A eine Null. Die Übergangsfunktionen für den Generator in Abbildung 2.2 ergeben sich zu

$$x_2 = \delta\left(x_0, b\right), \tag{2.6}$$

$$x_0 = \delta\left(x_2, c\right), \tag{2.7}$$

$$x_1 = \delta\left(x_2, d\right), \tag{2.8}$$

$$x_0 = \delta\left(x_1, a\right) \text{ und} \tag{2.9}$$

$$x_1 = \delta\left(x_1, d\right). \tag{2.10}$$

Die Schleife am Zustand x_1 wird als „Selfloop“ bezeichnet. Die Adjazenzmatrix des Generators ergibt sich aus den Übergangsfunktionen zu

$$A = \begin{pmatrix} 0 & a & c \\ 0 & d & d \\ b & 0 & 0 \end{pmatrix}. \tag{2.11}$$

Die reguläre Sprache $L(G)$ und somit das Verhalten von G wird durch die Gesamtheit aller Strings, die G von x_0 ausgehend generieren kann, definiert. Es gilt somit

$$L(G) = \{s \in \Sigma^* \mid \delta(x_0, s)!\}. \tag{2.12}$$

Das markierende Verhalten $L_m(G)$ von G enthält die Gesamtheit aller Strings, die von x_0 auf einen markierten Zustand führen

$$L_m(G) = \{s \in L(G) \mid \delta(x_0, s) \in X_m\}. \tag{2.13}$$

2.3. Komposition von Automaten

In diesem Abschnitt werden Operatoren vorgestellt, die zum Zusammenführen von Automaten dienen. Die daraus entstehenden Automaten werden Automatennetze oder Kompositionsautomaten genannt. Aus den Teilmodellen werden demnach durch Kompositionsoperatoren Gesamtmodelle erzeugt. Die Kopplung der Teilautomaten erfolgt über gemeinsame Ereignisse, so dass sie auf diese Ereignisse synchronisiert werden. Die Komposition von Automaten ermöglicht z.B. das Modellieren der Strecke in modularer Form. Die einzelnen Teilsysteme können durch die jeweiligen Kopplungen zusammengeführt werden. In diesem Abschnitt werden die Produktkomposition und die parallele Komposition betrachtet.

Es werden im Folgenden die Operationen für zwei Komponenten vorgestellt, wobei diese auf beliebig viele Komponenten erweiterbar sind. Die Alphabete Σ_1 und Σ_2 zweier Komponenten werden zunächst in gemeinsame und private Ereignisse aufgeteilt, so dass die gemeinsamen Ereignisse

$$\Sigma_{CE} = \Sigma_1 \cap \Sigma_2 \tag{2.14}$$

sind. Die privaten Ereignisse ergeben sich mit Σ_{PE}

$$\Sigma_{PE} = (\Sigma_2 - \Sigma_1) \cup (\Sigma_1 - \Sigma_2). \tag{2.15}$$

Produktkomposition

Die erste betrachtete Kopplung von Automaten ist die Produktkomposition [26]. Sie wird auch Strenge Synchronisation, kurz SPC (Strict Product Composition) [49] genannt. Das Produkt

zweier Automaten

$$G_x = G_1 \times G_2 \tag{2.16}$$

synchronisiert die Komponenten auf ihre gemeinsamen Ereignisse, verbietet dabei alle privaten Ereignisse, so dass in G_x nur gemeinsame Ereignisse von G_1 und G_2 auftreten können. Der resultierende Automat kann nach [26] durch das Quintupel

$$G_x = (X, \Sigma_x, \delta_x, z_0, X_m) \tag{2.17}$$

beschrieben werden. Das Alphabet ergibt sich zu

$$\Sigma_x = \Sigma_1 \cap \Sigma_2. \tag{2.18}$$

Die Zustandsmenge ergibt sich als kartesisches Produkt der Zustandsmengen der Automaten (siehe Gleichung (2.19)). Die Automaten können nur noch gleichzeitig im „totalen Gleichschritt“ [47] schalten.

$$X = X_1 \times X_2. \tag{2.19}$$

Die markierten Endzustände ergeben sich ebenso aus dem kartesischen Produkt der Endzustandsmengen, da nur ein Endzustand erreicht wird, wenn beide Automaten sich in ihren Endzuständen befinden.

$$X_m = X_{m1} \times X_{m2} \tag{2.20}$$

Da im Produktsystem nur noch private Ereignisse vorkommen können, wird die Zustandsübergangsfunktion wie folgt definiert

$$\delta(x, \sigma) = \begin{cases} \delta_1(x_1, \sigma) \times \delta_2(x_2, \sigma), & \text{wenn } \delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \\ \text{nicht definiert}, & \text{sonst.} \end{cases} \tag{2.21}$$

Informell bedeutet die Definition in Gleichung 2.21, dass ein Zustandsübergang für ein Ereignis σ möglich ist, wenn die Zustandsübergangsfunktionen δ_1 und δ_2 für dieses σ in beiden Komponenten definiert sind. Dabei steht das Ausrufezeichen für „definiert“. Ist eine der beiden Zustandsübergangsfunktionen nicht definiert, so kann kein Zustandswechsel erfolgen.

Die Sprache des Produktautomaten ergibt sich, da es keine privaten Ereignisse gibt, ausschließlich aus den gemeinsamen Zeichenketten, die durch die Komponenten generiert bzw. akzeptiert werden [26].

$$L(G_1 \times G_2) = L(G_1) \cap L(G_2) \tag{2.22}$$

Die Produktkomposition wird in technischen Systemen bei Rückkopplungsstrukturen verwendet, die z.B. bei der Zusammenschaltung von Strecke und Steuerungseinrichtung entsteht.

Parallele Komposition

Den zweiten Kompositionsoperator stellt die parallele Komposition dar [26]. Sie wird auch als Synchrones Produkt, kurz SYPC (Synchronous Product) [49] bezeichnet. Der SYPC Opera-

tor synchronisiert, wie der SPC Operator, die Automaten auf die gemeinsamen Ereignisse. Es werden aber, im Gegensatz zu der SPC Operation, die privaten Ereignisse zugelassen, so dass ein asynchrones Schalten der Automaten möglich ist. Der Kompositionsautomat wird nach [26] durch das Quintupel

$$G_{\|} = (X, \Sigma_{\|}, \delta_{\|}, x_0, X_m) \tag{2.23}$$

beschrieben. Das Alphabet des Kompositionsautomaten ergibt sich mit der Vereinigungsmenge der Teilalphabeten der Automaten

$$\Sigma_{\|} = \Sigma_1 \cup \Sigma_2 \tag{2.24}$$

und enthält somit auch die private Ereignisse. Die Zustände und die markierten Zustände ergeben sich (wie beim Produktautomaten) aus dem kartesischen Produkt der beiden Automaten. Die Zustandsübergangsfunktionen sind wie folgt formal definiert

$$\delta(x, \sigma) = \begin{cases} \delta_1(x_1, \sigma) \times \delta_2(x_2, \sigma), & \text{wenn } \delta_1(x_1, \sigma)\,! \wedge \delta_2(x_2, \sigma)\,! \\ \delta_1(x_1, \sigma) \times \{x_2\}, & \text{wenn } \delta_1(x_1, \sigma)\,! \wedge \neg\delta_2(x_2, \sigma)\,! \wedge \sigma \notin (\Sigma_1 \cap \Sigma_2) \\ \{x_1\} \times \delta_2(x_2, \sigma), & \text{wenn } \neg\delta_1(x_1, \sigma)\,! \wedge \delta_2(x_2, \sigma)\,! \wedge \sigma \notin (\Sigma_1 \cap \Sigma_2) \\ \text{nicht definiert}, & \text{sonst.} \end{cases} \tag{2.25}$$

Informell definiert die Gleichung 2.25 einen Zustandswechsel, wenn beide Zustandsübergangsfunktionen δ_1 und δ_2 definiert sind. In diesem Fall können beide Automaten gleichzeitig schalten. Ist nur eine der beiden Zustandsübertragungsfunktionen δ_1 bzw. δ_2 definiert, kann nur ein Automat G_1 bzw. G_2 schalten. Es findet aber, im Gegensatz zu dem SPC Operator, dennoch ein Zustandswechsel statt.

Die Sprache des Kompositionsautomaten kann durch die die natürliche Projektion beschrieben werden. Sie wird vereinfacht durch die Komposition der Sprachen L_1 und L_2 über die Alphabete Σ_1 und Σ_2 definiert [26]

$$L_1 \| L_2 = P^{-1}_{\Sigma_1}(L_1) \cap P^{-1}_{\Sigma_2}(L_2). \tag{2.26}$$

Die natürliche Projektion ist eine Funktion $P_\Sigma : \Sigma^* \to \Sigma_1^*$, die aus einem Alphabet Σ diejenigen Ereignisse entfernt, die $\Sigma_1 \subseteq \Sigma$ nicht enthalten soll. Die natürliche Projektion wird nach [49] wie folgt definiert

$$P_{\Sigma_1}(\epsilon) = \epsilon \tag{2.27}$$

$$P_{\Sigma_1}(\sigma) = \begin{cases} \sigma, & \text{wenn } \sigma \in \Sigma_1 \\ \epsilon, & \text{sonst} \end{cases} \tag{2.28}$$

$$P_{\Sigma_1}(s\sigma) = P_{\Sigma_1}(s) P_{\Sigma_1}(\sigma), s \in \Sigma^*, \sigma \in \Sigma. \tag{2.29}$$

Die inverse Projektion $P_{\Sigma_1}^{-1}$ fügt einem Alphabet Ereignisse aus Σ hinzu, die nicht in Σ_1 stehen und durch die natürliche Projektion wieder entfernt werden. Sie wird durch eine Abbildung

$$P_{\Sigma_1}^{-1} : \Sigma_1^* \to 2^{\Sigma^*} \tag{2.30}$$

beschrieben und wie folgt definiert

$$P_{\Sigma_1}^{-1}(t) = \left\{ s \in \Sigma^* \mid P_{\Sigma_1}(s) = t \right\}. \tag{2.31}$$

Haben die Automaten identische Ereignisalphabete $\Sigma_1 = \Sigma_2$, ergeben SYPC und SPC den selben Automat

$$G_1 \| G_2 = G_1 \times G_2. \tag{2.32}$$

Dieser Spezialfall wird nach [49] „Meet“ genannt. Ein weiterer Spezialfall ergibt sich, wenn keine gemeinsamen Ereignisse existieren, so dass gilt

$$\Sigma_1 \cap \Sigma_2 = \emptyset. \tag{2.33}$$

In diesem Fall bewegen sich beide Automaten absolut asynchron zueinander. Dieser Fall wird nach [49] als „Shuffle“ bezeichnet.

2.4. Analyse ereignisdiskreter Systeme

Die Analyse ereignisdiskreter Systeme beschäftigt sich mit den Fragen nach erreichbaren Zuständen (vom Anfangszustand, sowie von beliebigen Zuständen), aber auch mit einem möglichen Blockieren des Systems. Es werden dafür Situationen analysiert, die das Erreichen von markierten Zuständen für das System unmöglich machen (Deadlock/ Livelock). Mit der Beantwortung dieser Fragen können gewisse Sicherheitsanforderungen des Systems untersucht werden. Es kann festgestellt werden, ob ein System aus jedem beliebigen Zustand in einen sicheren Zustand gefahren werden kann. Auch kann geprüft werden, ob z.B. ein Auftrag in jeder möglichen Situation abgeschlossen werden kann. Im folgenden Abschnitt werden diese Analysekonzepte und deren Operatoren vorgestellt.

2.4.1. Erreichbarkeitsanalyse

Die Erreichbarkeitsanalyse beantwortet die Frage, ob bestimmte Zustände aus einem Anfangszustand $x_0 \in X$ erreicht werden können. Gibt es einen String $s \in \Sigma^*$,so dass $x \in X$ von x_0 erreicht werden kann, heißt dieser Zustand von x_0 aus erreichbar. Es muss somit eine

Funktion $\delta(x_0, s) = x$ definiert sein. Es ergibt sich nach [47] die Menge aller erreichbaren Zustände durch

$$X_{ac} = \{x \in X \mid \exists s \in \Sigma^*(\delta(x_0, s) = x)\}. \tag{2.34}$$

Die nicht erreichbaren Zustände $X - X_{ac}$ können vom Anfangszustand nicht erreicht werden, so dass sie für die Repräsentation des Systems unbedeutend sind. Sie können durch die Ac-Operation entfernt werden, so dass die Zustandsmenge, Zustandsübergangsfunktion und die Menge der markierten Zustände reduziert werden. Σ bleibt jedoch unverändert. Die Ac-Operation wird für einen Generator G nach [47] wie folgt definiert

$$Ac(G) = \{X_{ac}, \Sigma, \delta_{ac}, x_0, X_{ac,m}\} \text{ mit} \tag{2.35}$$

$$X_{ac} = \{x \in X \mid \exists s \in \Sigma^*(\delta(x_0, s) = x)\}, \tag{2.36}$$

$$X_{ac,m} = X_m \cap X_{ac}, \tag{2.37}$$

$$\delta_{ac} = \delta \mid_{A_{ac} \times \Sigma \to X_{ac}}. \tag{2.38}$$

Durch die Notation $\delta|_{A_{ac} \times \Sigma \to X_{ac}}$ wird nach [47] die Einschränkung von δ auf die Menge der erreichbaren Zuständen dargestellt.

Das Konzept der Ko-Erreichbarkeit durchsucht den Generator G auf Zustände von denen es mindestens eine Trajektorie gibt, die von einem beliebigen Zustand auf einen markierten Zustand führt. Ein Zustand $x \in X$ wird als ko-erreichbar bezeichnet, wenn ein String $s \in \Sigma^*$ existiert, so dass $\delta(x, y) \in X_m$ und somit ein markierter Zustand erreicht werden kann. Die Menge der ko-erreichbaren Zustände wird nach [47] definiert durch

$$X_{coac} = \{x \in \Sigma \mid \exists s \in \Sigma^*(\delta(x, s) \in X_m)\}. \tag{2.39}$$

Auch für nicht ko-erreichbare Zustände gibt es eine Reduzierungsoperation. Die CoAc-Operation entfernt alle Zustände, die nicht auf einem Pfad von x_0 zu einem Zustand $x \in X_m$ liegen [9]. Folglich wirkt sich die Operation auf die Sprache des Generators $L(G)$, nicht aber auf das markierende Verhalten $L_m(G)$ aus. Die CoAc-Operation wird nach [47] definiert als

$$CoAc(G) = (X_{coac}, \Sigma, \delta_{coac}, X_m) \text{ mit} \tag{2.40}$$

$$X_{coac} = \{x \in X \mid \exists s \in \Sigma^*(\delta(x, s) \in X_m)\}, \tag{2.41}$$

$$x_{0,coac} = \begin{cases} x_0, & \text{wenn } x_0 \in X_{coac} \\ \text{nicht definiert}, & \text{sonst} \end{cases}, \tag{2.42}$$

$$\delta_{coac} = \delta \mid_{x_{coac} \times \Sigma \to X_{coac}}. \tag{2.43}$$

Auch hier bildet die Notaion $\delta \mid_{x_{coac} \times \Sigma \to X_{coac}}$ die Einschränkung von δ auf die Menge der ko-erreichbaren Zustände.

Beide Operationen nacheinander bilden die Trim-Operation. Der Generator enthält nach der

Anwendung des Trim-Operators weder nicht erreichbare, noch nicht ko-erreichbare Zustände. Die Anzahl der Zustände wird somit auf die wesentliche Anzahl reduziert, in dem die überflüssigen Zustände eliminiert werden.

Die Ermittlung der Erreichbarkeit von Zuständen kann, neben der Berechnung der Erreichbarkeitsmenge, auch algebraisch mittels der Adjazenzmatrix durchgeführt werden [26]. Der Zustand i ist vom Zustand j erreichbar, wenn es eine Zahl k gibt, so dass das ij-te Element der Matrix G^k ungleich Null ist. Die Zahl k repräsentiert dabei die Länge des Pfades von i nach j.

2.4.2. Deadlock, Livelock, Blockierung

Das Blockieren eines Systems beschreibt eine Situation, in der der Generator keinen markierten Zustand mehr erreichen kann. Für das System bedeutet das zum Beispiel, dass ein Auftrag nicht beendet werden kann. Das Blockieren beinhaltet immer einen Live- oder Deadlock. Ein Deadlock entsteht, wenn der Generator einen Zustand $x \notin X_m$ erreicht mit $\neg\delta(x,\sigma)!,\forall \in \Sigma$, in dem er keine Ereignisse mehr generieren kann. Dies hat zur Folge, dass der Generator diesen Zustand also nicht mehr verlassen kann. In diesem Fall gilt nach [47] die echte Inklusion

$$\overline{L_m(G)} \subset L(G). \tag{2.44}$$

In einem Livelock erreicht der Generator eine Zustandsmenge $\widetilde{X} \subseteq X$ mit $x \notin X_m, \forall x \in \widetilde{X}$. Der Generator kann dabei kein Ereignis generieren, mit dem er die Zustandsmenge verlassen kann. Er kann weiter aktiv schalten, aber keinen markierten Zustand erreichen. Auch hier gilt die echte Inklusion 2.44. Ein Generator ist folglich blockierend, wenn die Inklusion 2.44 gilt. Entsprechend ist er nichtblockierend genau dann, wenn

$$\overline{L_m(G)} = L(G). \tag{2.45}$$

2.5. Supervisory Control Theory (SCT)

Die Supervisory Control Theory beinhaltet eine Vielzahl von Methoden zur formalen Steuerungssynthese für ereignisdiskrete Systeme. Gegenwärtig gibt es zahlreiche Publikationen, die die unterschiedlichsten Probleme der Steuerungstechnik mit Hilfe der SCT behandeln. Die SCT behandelt sowohl logische wie auch zeitbewertete ereignisdiskrete Systeme. Die formalen Methoden wurden bis heute so weit erweitert, dass sie sich auf eine Vielzahl von Problemstellungen formal anwenden lassen. Erstmals wurde die SCT formal in der Dissertation [36] von P.J. Ramadge vorgestellt. Sie wurde durch Methoden zum strukturellen Steuerungsentwurf, zur partiellen Beobachtbarkeit und für zeitbewertete Systeme erweitert.

In dieser Arbeit wird ausschließlich Bezug auf die logischen strukturellen, jedoch nicht auf die zeitbewerteten Steuerungsentwürfe genommen. Zusammengetragen wurden die gesamten Erweiterungen von W.M. Wonham in [49].

Während die meisten Publikationen sich mit kleinen, konstruierten Beispielen befassen, wurden bis heute auch größere Anwendungen mit Hilfe der SCT automatisiert [21], [47]. In der Industrie findet sich die SCT bis jetzt eher selten. Dies liegt zum einen daran, dass die Anforderungen bei großen Industrieanlagen bezogen auf den Rechenaufwand sehr hoch sind, zum anderen wird in der Praxis die Verifikation der Synthese vorgezogen [47]. Besonders die Beschreibung ereignisdiskreter Systeme mittels Automaten und Sprachen ist in der Automatisierung, im Gegensatz zur Informatik, überwiegend unbekannt.

Die SCT teilt die Steuerstrecke in Steuerung und Strecke auf und weicht somit von der Definition einer Steuerung der DIN 19226-4 ab [31]. Die Strecke wird als Generator modelliert. Dessen Verhalten wird durch die Steuerung, den Supervisor[1] beeinflusst. Der Supervisor erlaubt und verbietet Ereignisse und steuert dadurch das Verhalten der Strecke. Ereignisse, die von Sensoren kommen oder Rückmeldungen von der Beendigung einer Prozedur darstellen, können nicht vom Supervisor beeinflusst werden. Diese Ereignisse werden als nicht steuerbar bezeichnet. Die Ereignisse, die zum Beispiel Aktoren ansteuern, werden hingegen als steuerbar bezeichnet, da diese durch den Supervisor beeinflusst werden können. Das Ereignisalphabet des Generators wird also in steuerbare Σ_c und nicht steuerbare Σ_{uc} Teilalphabete partitioniert. Die steuerbaren Ereignisse werden bei der grafischen Darstellung des Generators mit einem Querstrich an der Transition gekennzeichnet. Generell können in einem System auch beobachtbare Σ_o und nicht beobachtbare Ereignisse Σ_{uo} auftreten. Dieser Fall wird in dieser Arbeit nicht weiter betrachtet. Es ergibt sich für Σ_c und Σ_{uc} der Zusammenhang

$$\Sigma = \Sigma_c \cup \Sigma_{uc} \text{ mit } \Sigma_c \cap \Sigma_{uc} = \emptyset. \tag{2.46}$$

Die Steuerung beeinflusst die Strecke mittels einer Ereignisfolgenrückführung. Der Supervisor befindet sich in der Rückführung des Steuerkreises. Abbildung 2.3 stellt das Blockschaltbild einer typischen Ereignisfolgenrückführung dar. Der Supervisor S überwacht die von G generierte Ereignisfolge und verbietet diejenigen Ereignisse, die im nächsten Schritt nicht erwünscht sind. $S(s)$ repräsentiert die Steuereingriffe von S. Die Menge der möglichen Steuereingriffe muss immer die nicht steuerbaren Ereignisse enthalten, da diese nicht verboten werden können. Das gesteuerte Verhalten $L(S/G)$ wird so definiert, dass der leere String ϵ immer Teil von $L(S/G)$ ist, so dass $L(S/G)$ nicht leer ist. Desweiteren kann ein auf s folgendes Ereignis σ nur dann erlaubt werden, wenn die Konkatenation von s und σ in S erlaubt werden und in G möglich sind. Nur diese Strings sind im gesteuerten Verhalten erlaubt. Formell können die Bedingungen nach Wonham [49] wie folgt definiert werden:

1. $\epsilon \in L(S/G)$

[1] Der Begriff Supervisor wird für Singular als auch für den Plural verwendet

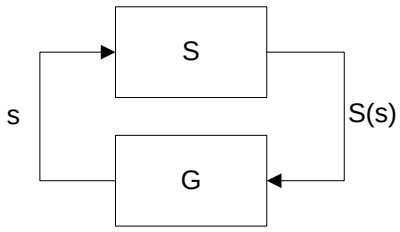


Abbildung 2.3.: Steuerkreis S/G

2. Ist $s \in L(S/G)$, $\sigma \in S(s)$, und $s\sigma \in L(G)$ dann ist $s\sigma \in L(S/G)$
3. Keine weiteren Strings gehören zu L(S/G).

Das markierende Verhalten wird im Steuerkreis auf die erlaubten Strings reduziert, so dass $L_m(S/G) = L(S/G) \cap L_m(G)$ ist. Der Steuerkreis ist nichtblockierend, wenn gilt

$$\overline{L_m(S/G)} = L(S/G). \tag{2.47}$$

Das Verhalten des gesteuerten Systems wird durch eine Spezifikation K festgelegt. Die formale Beschreibungsform für die Spezifikation ist in dieser Arbeit ebenfalls ein Generator. K ist eine Teilmenge von $L(G)$ und repräsentiert das gewünschte Verhalten der Anlage. Die Spezifikation kann die Bedingungen der Steuerbarkeit bezüglich G erfüllen. Damit eine Spezifikation K steuerbar ist, muss sichergestellt werden, dass kein String $s \in \overline{K}$ durch das Auftreten eines nicht steuerbaren Ereignisses $\sigma \in \Sigma_{uc}$ $\overline{K}$ verlässt. Das Verlassen von $\overline{K}$ und somit dem gesteuerten Verhalten $L(S/G)$ bedeutet das Verlassen des geforderten Verhaltens der Strecke. Formal kann die Steuerbarkeit durch den Zusammenhang

$$\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K}. \tag{2.48}$$

überprüft werden.

Ist die Spezifikation nicht steuerbar, lässt sich aus K die Supremale steuerbare Teilsprache $K^{\uparrow C}$ bzw. die Infimale präfix-abgeschlossene steuerbare Obersprache $K^{\downarrow C}$ bilden. Die Definitionen dafür finden sich in [9], [47] und [49]. Die in [49] beschriebenen Algorithmen zur Berechnung dieser Teilsprachen sind in Tools wie TCT und DESTool bereits implementiert. Die Existenz eines nichtblockierenden Supervisors für eine steuerbare Spezifikation kann mittels der Existenzbedingung für einen nichtblockierenden Supervisor festgestellt werden. Für diese Existenzbedingung, auch $L_m(G)$-Abgeschlossenheit genannt, muss der geschlossene Steuerkreis die Bedingung

$$L_m(S/G) = K \wedge L(S/G) = \overline{K} \tag{2.49}$$

erfüllen. Diese Eigenschaft wird erreicht genau dann wenn

$$K = \overline{K} \cap L_m(G). \tag{2.50}$$

In [9] und [49] sind mehrere Standardprobleme der SCT und ihre Lösungen zusammengefasst. Sie definieren unterschiedliche Ansätze, um einen anwendungsspezifischen Supervisor erstellen zu können. In dieser Arbeit wird nur das Basic Supervisory Control Problem (BSCP) bzw. das BSCP-Nichtblockierend (-NB) betrachtet. Im BSCP werden Strecken mit nicht steuerbaren, aber keine mit nicht beobachtbaren Ereignissen betrachtet. Das BSCP beachtet zunächst nicht die Anforderung nach Nichtblockieren der Steuerung. Das BSCP-NB wird nach [9] und [47] wie folgt formuliert:

BSCP-NB:
Gegeben ist ein Generator G mit einem Ereignisalphabet Σ, $\Sigma_{uc} \subseteq \Sigma$ und eine $L_m(G)$-abgeschlossene Spezifikation $K \subseteq L_m(G)$. Bestimme eine nichtblockierende Steuerung, so dass gilt:

1. $L_m(S/G) \subseteq K$
2. $L_m(S/G)$ „maximal“ ist, also für jede andere nichtblockierende Steuerung S' mit $L_m(S'/G) \subseteq K$ gilt:

$$L_m(S'/G) \subseteq L_m(S/G)). \tag{2.51}$$

Die gesuchte Steuerung ist dann so zu wählen, dass:

$$L(S/G) = \overline{K^{\uparrow C}}. \tag{2.52}$$

Der Supervisor kann in zwei Formen repräsentiert werden. Erstens kann er als Liste seiner Steuereingriffe realisiert werden. Dafür wird für jeden String $s \in L(G)$ die Menge der im nächsten Schritt erlaubten Ereignisse angegeben. Die Steuereingriffe müssen alle erlaubten Ereignisse der abgehenden Transitionen aus dem aktuellen Zustand $\delta(x, s)$ enthalten. Alle nicht steuerbaren Ereignisse können nicht beeinflusst werden und müssen erlaubt werden. Mit der Vereinigung aller in $K^{\uparrow C}$ erlaubten Ereignisse wird sichergestellt, dass die Spezifikation nicht verletzt wird. Formal lassen sich nach [47] die Steuereingriffe S(s) definieren durch

$$S(s) = [\Sigma_{uc} \cap \{\sigma \in \Sigma \mid \delta(\delta(x_0, s), \sigma)!\}] \cup \left\{\sigma \in \Sigma_c \mid s\sigma \in \overline{K^{\uparrow C}}\right\}. \tag{2.53}$$

Die zweite, praktikablere Variante repräsentiert den Supervisor durch einen Akzeptor R. Diese Realisierung wird auch Standardrepräsentation von S genannt. Hierfür wird der Akzeptor $R = \{Y, \Sigma, \zeta, y_0, Y_m\}$ konstruiert, so dass $Y = Y_m$, Σ gleich dem Ereignisalphabet der Strecke und ζ, so dass $L_m(R) = L(R) := \overline{K^{\uparrow C}}$ ist. Das Verhalten des geschlossenen

Steuerkreises lässt sich in diesem Fall mittels der Strengen Synchronisation (SPC) berechnen.

$$L(G \times R) = L(G) \cap L(R) = L(G) \cap \overline{K^{\uparrow C}} = \overline{K^{\uparrow C}} = L(S/G) \tag{2.54}$$

$$L_m(G \times R) = L_m(G) \cap {}_m(R) = \overline{K^{\uparrow C}} \cap {}_m(G) = L(S/G) \cap L_m(G) = Lm(S/G) \tag{2.55}$$

2.6. Strukturelle Ansätze

Die SCT, wie sie im Abschnitt 2.5 vorgestellt wurde, setzt ein unstrukturiertes Gesamtmodell der Strecke voraus. Für dieses Gesamtmodell wird ein Supervisor erstellt, der die gesamte Steuerungsaufgabe übernimmt. Diese Art von Verfahren wird im Allgemeinen als monolithischer Ansatz bezeichnet. In der Realität sind Ansätze dieser Art aufgrund der Modellkomplexität, Steuerungsstruktur und der algorithmischen Komplexität nicht praktikabel. Die nachstehenden Syntheseverfahren bilden einen Ansatz, die Nachteile des monolithischen Ansatzes zu eliminieren.

2.6.1. Modularer Ansatz

Das Konzept des modularen Steuerungsentwurfs verfolgt das Ziel, die Modellkomplexität der resultierenden Supervisor gegenüber dem monolithischen Ansatz zu reduzieren. Die Grundidee ist, die Steueraufgabe auf mehrere Supervisor aufzuteilen. Die einzelnen Supervisor bilden zusammen den modularen Supervisor S_{mod}. In Abbildung 2.4 ist das Schema des Ansatzes für zwei Supervisor dargestellt. Voraussetzung für den Ansatz ist, dass alle Ereignisse global für jeden Supervisor verfügbar sind.

Die Supervisor S_1 und S_2 überwachen den von G generierten String s. Je nach Steuerungsaufgabe verbieten sie die für sich unerwünschten Ereignisse. Diese werden durch ihre Steuereingriffe $S_1(s)$ und $S_2(s)$ festgelegt. Die Konjunktion der Steuereingriffe legt fest, ob ein Ereignis verboten oder erlaubt wird. Ist das Folgeereignis σ in $S_1(s)$ und $S_2(s)$ enthalten, so wird es nicht verhindert. Ist es jedoch in einem der beiden Steuereingriffe nicht enthalten, so wird es verboten. Die Steuereingriffe $S_{mod}(s)$ ergeben sich aus den Schnittmengen der einzelnen Steuereingriffe der Supervisor. Der modulare Supervisor $S_{mod} : L(G) \rightarrow 2^{\Sigma}$ kann allgemein für n Supervisor formal durch

$$S_{mod}(s) = S_1(s) \cap S_2(s) \cap \ldots \cap S_n(s) \tag{2.56}$$

definiert werden.

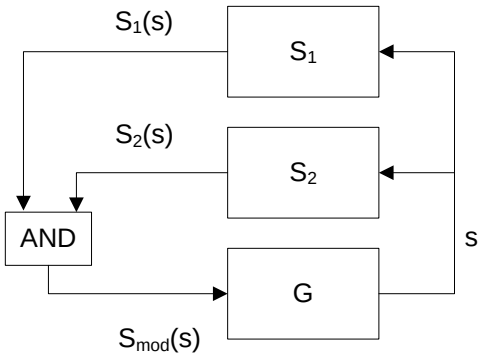


Abbildung 2.4.: Steuerkreis S_{mod}/G des modularen Ansatzes

Aus Gleichung (2.56) folgt nach [9] und [47] direkt die resultierenden Sprachen für den geschlossenen Steuerkreis $L(S_{mod}/G)$ und die markierende Sprache $L_m(S_{mod}/G)$.

$$L(S_{mod}/G) = L(S_1/G) \cap L(S_2/G) \cap \ldots \cap L(S_n/G) \tag{2.57}$$

$$L_m(S_{mod}/G) = L_m(S_1/G) \cap L_m(S_2/G) \cap \ldots \cap L_m(S_n/G). \tag{2.58}$$

Für zwei Spezifikationen K_1 und K_2 erfolgt der Entwurf der Supervisor S_1 und S_2 bezüglich der Komponenten G_1 und G_2 der Strecke G, so dass $L(S_1/G_1) = K_1$ und $L(S_2/G_2) = K_2$. Die Steuerbarkeit bleibt bei Bildung der Schnittmenge von präfix-geschlossenen Sprachen erhalten, somit auch für $K_1 \cap K_2$ [9].

Die Realisierung von S_{mod} soll durch die Schnittmenge der erlaubten Ereignissen von S_1 und S_2 gebildet werden, wie in Abbildung 2.4 dargestellt. Diese Repräsentation wird modulare Realisierung genannt. Vorteil gegenüber des Produktsystems ist der geringere Speicherbedarf der Steuerung. Besitzt S_1 n_1 Zustände und S_2 n_2, ergeben sich insgesamt $n_1 + n_2$ Zustände. Die Produktkomposition $S = S_1 \times S_2$ hingegen würde, bei gleichen Ereignisalphabeten von S_1 und S_2, $n_1 \cdot n_2$ Zustände ergeben. Die Berechnung der supremalen steuerbaren Teilsprache kann bei präfix-geschlossenen Sprachen für jede Spezifikation getrennt berechnet werden.

$$(K_1 \cap K_2)^{\uparrow C} = K_1^{\uparrow C} \cap K_2^{\uparrow C}. \tag{2.59}$$

Dadurch wird die algorithmische Komplexität der Synthese reduziert [50]. Das Modular Supervisory Control Problem (MSCP) definiert, dass aus den Spezifikationen $K = K_1 \cap K_2 \ldots \cap K_n$, mit $K_i = \overline{K_i}, i = 1, \ldots n$ ein modularer Supervisor S_{mod} bestimmt werden soll, so dass

$$L(S_{mod}/G) = K^{\uparrow C} \tag{2.60}$$

gilt. Der modulare Supervisor soll also so entworfen werden, dass er mit dem Verhalten eines monolithischen Supervisors identisch ist. Zur Lösung des MSCP lassen sich Aufgrund der

präfix-abgeschlossenen Teilspezifikationen die einzelnen Supervisor S_i unabhängig voneinander entwerfen, so dass

$$L(S_i/G) = K_i^{\uparrow C}, \text{ für } i = 1, \ldots n \tag{2.61}$$

gilt. Es lassen sich dadurch einzelne nichtblockierende Supervisor synthetisieren. Die Überprüfung auf das Nichtblockieren der einzelnen Supervisor $\overline{L_m(S_1/G)} = L(S_1/G)$ und $\overline{L_m(S_2/G)} = L(S_2/G)$ erfüllen nicht zwingend die Aussage $\overline{L_m(S_1 \cap S_2/G)} = L(S_1 \cap S_2/G)$. Zwei nichtblockierende Supervisor ergeben in der Konjunktion also nicht unbedingt einen nichtblockierenden Supervisor S_{mod} [50]. Ein modularer Supervisor muss nach [21] und [50] zusätzlich die Bedingung

$$\overline{L_m(S_1/G) \cap L_m(S_2/G)} = L(S_1/G) \cap L(S_2/G) \tag{2.62}$$

erfüllen, um nichtblockierend zu sein.

Es ergibt sich, durch die Implementierung der einzelnen Supervisor und der Konjunktion der einzelnen Supervisor, in der Regel eine geringer Modellkomplexität der Steuerung als bei einem monolithischen Entwurf. Auch ist die Steuerung flexibler hinsichtlich Änderungen an der Steuerungsaufgabe. Es muss lediglich die betroffene Spezifikation geändert und neu analysiert werden. Der Rest der Steuerung bleibt unverändert.

Die Steuerung muss nicht zwingend auf einer Hardware realisiert werden. Sie kann genauso bei n Spezifikationen auf bis zu n Steuerungen verteilt werden. Jedoch müssen dafür sämtliche Ereignisse global verfügbar sein, was einen erheblichen Verkabelungsaufwand verursachen kann. Auch können nicht alle Spezifikationen zerlegt werden, wie es der Ansatz fordert. Ein weiterer großer Nachteil besteht darin, dass weiterhin ein strukturloses Gesamtmodell der Strecke notwendig ist, um z.B. das Nichtblockieren einer Steuerung sicherstellen zu können.

2.6.2. Lokal - modularer Ansatz

Der lokal-modulare Ansatz ist eine Erweiterung des modularen Ansatzes mit dem Ziel, ein strukturiertes Modell von G und S zu erhalten. Dabei wird die Komplexität der Synthese des Supervisors reduziert. Dafür sollen möglichst wenig Komponenten des Streckenmodells G für den Steuerungsentwurf verwendet werden. Die Grundidee ist dabei, nur Komponenten zusammenzufassen, die zur Einhaltung einer Spezifikation notwendig sind. Der lokalmodulare Ansatz ist besonders für Systeme geeignet, die viele Nebenläufigkeiten enthalten. In Abbildung 2.5 ist die Architektur des Ansatzes für zwei Supervisor schematisch dargestellt. Jeder Supervisor steuert nur einen bestimmten Teil des Gesamtsystems. Dafür wird

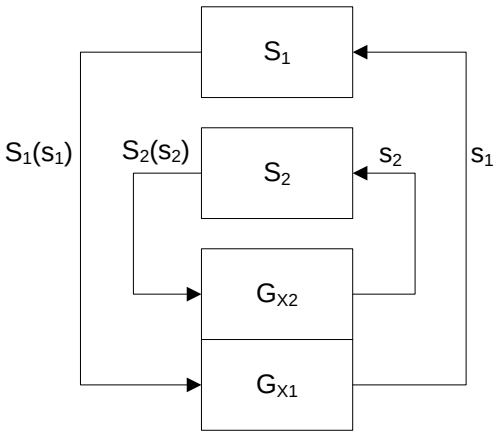


Abbildung 2.5.: Architektur des lokal-modularen Ansatzes

zunächst das Kompositionssystem aus allen Teilmodellen des Systems gebildet. Daraus ergeben sich die Strecke G und das Ereignisalphabet Σ. Aus diesem Kompositionssystem wird ein äquivalentes Produktsystem gebildet. Dafür wird das Kompositionssystem in asynchrone Komponenten partitioniert, so dass das feinste Produktsystem entsteht.

Die durch Spezifikationen synchronisierten asynchrone Komponenten werden mittels des SYPC-Operators zu lokalen Systemen zusammengefasst. Als letztes werden die gegebenen Spezifikationen bezüglich der lokalen Systeme, des eingeschränkten Systems und des Gesamtsystems ausgedrückt. Die Definitionen und Gleichungen zum Kompositionssystem, (feinsten) Produktsystem und Lokalen System, sowie die Anpassung der Spezifikation und die daraus folgenden Supervisor sind in den Arbeiten von de Queiroz und Cury [33] [34] und [35] zu finden. Das Werk von Wenck [47] enthält, neben den Definitionen, auch ein Beispiel zum besseren Verständnis der Anpassung des Systemmodells und der Spezifikationen.

Dem Werk ist auch Abbildung 2.6 entnommen und erweitert. Das Venn-Diagramm zeigt die Aufspaltung der Ereignisalphabete in ihre feinsten Produktsysteme. Es zeigt ein System, bestehend aus fünf Komponenten G_i' mit $i = 1 \ldots 5$. Die inhärent asynchronen Komponenten bilden die Generatoren $G_1 = G_1'$, $G_2 = G_2' \| G_3'$, $G_3 = G_4'$ und $G_4 = G_5'$. Sie sind asynchron zueinander, da sie keine gemeinsamen Ereignisse in ihren Ereignisalphabeten Σ_i(mit $i = 1 \ldots 4$) haben. Die Generatoren G_1, G_2 und G_2, G_3 werden durch eine Spezifikation zwangsweise synchronisiert und bilden die Generatoren $G_{X1} = G_1 \| G_2$ und $G_{X2} = G_2 \| G_3$ und die dazugehörigen Ereignisalphabete Σ_{X1} und Σ_{X2}. G_4 wird durch keine Spezifikation synchronisiert, so dass sich das eingeschränkte System G_e aus der parallelen Komposition mit G_1, G_2 und G_3 ergibt. Die Abbildung 2.6 veranschaulicht die Aufspaltung des Systems auf die einzelnen Komponenten. Es zeigt aber auch, dass bei Systemen, die keine asynchronen Komponenten enthalten, sich als eingeschränktes System wieder das Gesamtmodell des Systems ergibt. Dies ist auch der Fall, wenn alle feinsten Produktsysteme durch

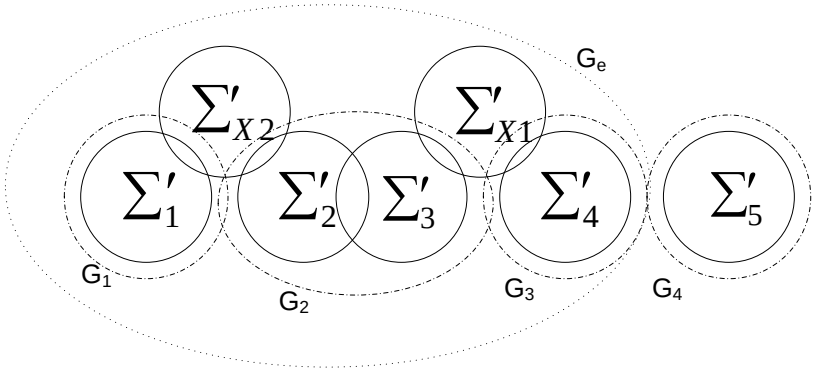


Abbildung 2.6.: Venn-Diagramm der Ereignisalphabete

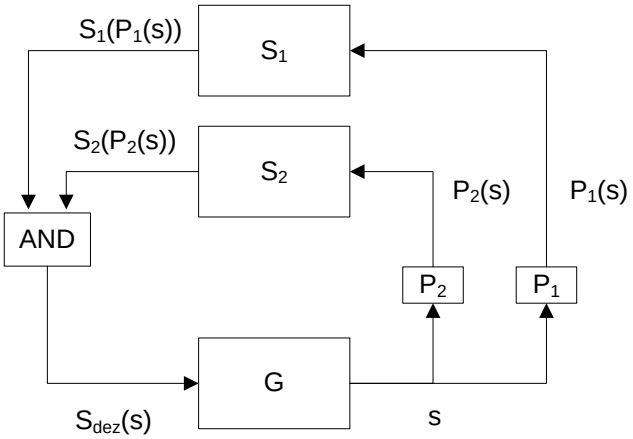


Abbildung 2.7.: Architektur des dezentralen Ansatzes

Spezifikationen synchronisiert werden. In diesem Fall ergibt sich keine Reduzierung der Modellkomplexität gegenüber dem modularen Ansatz.

2.6.3. Dezentraler Ansatz

Der dezentrale Ansatz stellt theoretisch betrachtet eine Erweiterung des modularen Ansatzes auf partielle Beobachtbarkeit dar. Viele verteilte Systeme besitzen Einschränkungen bezüglich der Verfügbarkeit sämtlicher Ereignisse. Es können z.B. durch räumliche Trennung nicht alle Sensorsignale an jeder Steuerung verfügbar sein. Die eingeschränkte Verfügbarkeit von Ereignissen wird durch die natürliche Projektion (siehe Abschnitt 2.3) repräsentiert. Durch die natürliche Projektion werden die für die Steuerung nicht verfügbaren Ereignisse aus dem Ereignisalphabet der Strecke entfernt. Der dezentrale Ansatz wurde von Lin und Wonham in [22] und [23] formalisiert.

Es werden für den dezentralen Ansatz zwei Steuerungsproblemklassen unterschieden. Zunächst das lokale dezentrale Steuerungsproblem, das die Steuerungsaufgabe auf einzelne Supervisor aufteilt. Jeder Supervisor muss eigenständig seine Steuerungsaufgabe erfüllen. Das Steuerungsproblem ist gelöst, wenn jeder Supervisor seine lokale Spezifikation isoliert erfüllt. Kann die Steuerungsaufgabe nicht in lokale Spezifikationen aufgeteilt werden, so muss ein globales dezentrales Steuerungsproblem gelöst werden. In diesem Fall wird die Spezifikation als globale Spezifikation gegeben, die von den einzelnen Supervisor gemeinsam erfüllt werden muss.

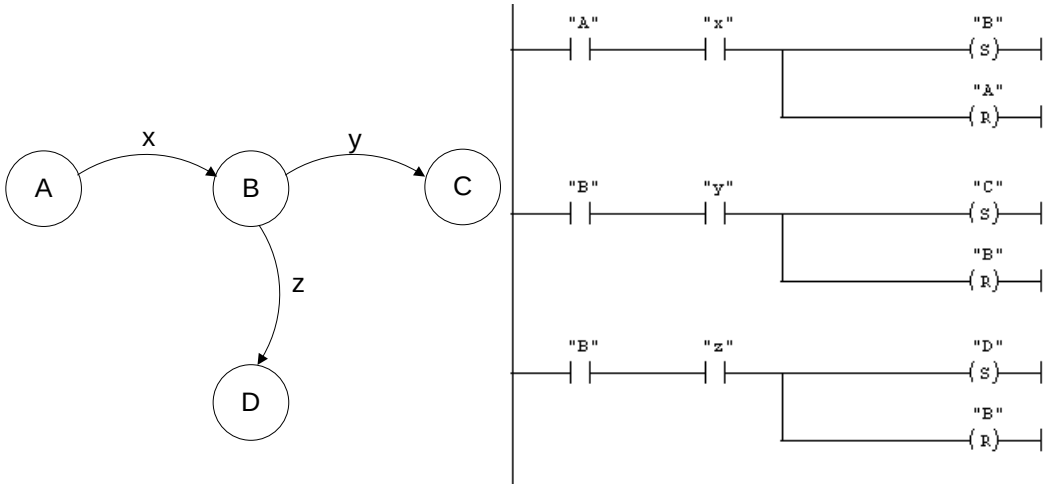
2.7. Codegenerierung

Die Codegenerierung ist eine entschiedene Phase im Entstehungsprozess der Steuerung. In der Industrie wird häufig eine Speicherprogrammierbare Steuerung (SPS) zum Ausführen des Programmcodes eingesetzt. Balemi et al. entwickelten in ihrer Arbeit [5] einen Ansatz zur Implementierung eines ereignisdiskreten Supervisor auf einem Multiprozessorsystem. Leduc stellt in seiner Arbeit [21] eine Methode vor, in der die Generatoren zunächst in getaktete Moore Automaten umgewandelt werden und anschließend der SPS-Code erzeugt wird. Fabian und Hellgren erweiterten den Ansatz in [14] von Balemi et al. auf eine allgemeine Methode zur Umwandlung eines Generators in ein Ladder Logic Diagram (LDD)[2]. Dabei werden zwei Verfahren vorgestellt. Das Grundprinzip beider Verfahren ist identisch. Die Zustände werden von Bitvariablen (Merkerbits) repräsentiert. Das Setzen dieser Bits bedeutet das Aktivieren des Zustandes. Durch das Schalten einer Transition, hervorgerufen durch das Auftreten eines Ereignisses, wird die aktuelle Zustandsvariable zurück und die nächste Zustandsvariable gesetzt. Abbildung 2.8 zeigt die Umsetzung eines Generators in AWL. Er enthält zunächst ausschließlich nicht steuerbare Ereignisse.

Jeder Zustand wird durch ein Merkerbit der SPS repräsentiert. In diesem Beispiel tragen die Merker die symbolischen Namen A, B und C. Die Ereignisse werden durch x, y und z dargestellt. Ist der Zustand A aktiv (der Merker gesetzt) und das Ereignis x tritt auf, wird der Merker des Zustandes B gesetzt und der des Zustandes A zurück gesetzt. Im Kontaktplan wird die Abfrage der Ereignisse und der Zustände durch einen Schließer, in AWL durch eine Konjunktion, realisiert.

Die steuerbaren Ereignisse können unterschiedlich behandelt werden. In der Arbeit von Fabian und Hellgen [14] werden zwei Varianten vorgestellt. Sie unterscheiden sich im Zeitpunkt des Setzens des Ereignisses. Die steuerbaren Ereignisse müssen von der Steuerung forciert

[2] Im deutschen wird LLD als Kontaktplan (KOP) bezeichnet, so auch in der Siemens Simatic Programmiersprache STEP7



a) Einfacher Generator b) Erzeugter Kontaktplan

Abbildung 2.8.: Einfacher Generator und erzeugter Kontaktplan

und so explizit im Programmcode gesetzt werden. In Abbildung 2.9 sind die zwei Möglichkeiten von Fabian und Hellgren dargestellt. In Abbildung 2.9b werden die Ereignisse am Ende, nach dem Setzen der Zustände erzeugt. In Abbildung 2.9c werden die Ereignisse schon beim Zustandswechsel erzeugt. Die Priorität der Transitionen kann durch die Reihenfolge der Abfragen der Ereignisse im Code festgelegt werden.

Auch die Reihenfolge der Ereignisabfragen ist entscheidend. In Abbildung 2.9c müssen unbedingt die nicht steuerbaren Ereignisse als erstes abgefragt werden. Der Fall, in dem erst die steuerbaren Ereignisse abgefragt werden, hätte die Konsequenz, dass die nicht steuerbaren Ereignisse ignoriert werden. Würde also in Abbildung 2.9 erst das Ereignis y abgefragt und aus dem Zustand B der Zustand C gesetzt werden, würde das Auftreten von z nicht mehr berücksichtigt werden. Dies hätte zur Folge, dass das Ereignis z verboten wird, was nach der SCT für nicht steuerbare Ereignisse unmöglich ist.

Ein weiteres Problem stellt das Setzen von Ausgängen dar. Sollen die Ausgänge direkt aus dem Automaten angesprochen werden, so muss sichergestellt werden, dass das Ereignis solange aktiv ist, bis z.B. das Relais geschaltet oder Motor die gewünschte Lage erreicht hat. Werden Ausgänge der Steuerung als steuerbare Ereignisse definiert, so müssen sie aus der Steuerung gesetzt werden. Außer das Ereignis wird z.B. durch einen Schalter erzeugt, der manuell betätigt werden kann. Im Beispiel aus Abbildung 2.9 werden die steuerbaren Ereignisse nur für die Zeit eines Programmzyklusses „true“ gesetzt. Daraus ergeben sich, je nach Art des Ausgangs, folgende Probleme:

1. Bei selbsthaltenden Ausgängen bleibt der Ausgang dauerhaft gesetzt und wird ohne das Eingreifen der Steuerung nicht mehr zurückgesetzt.

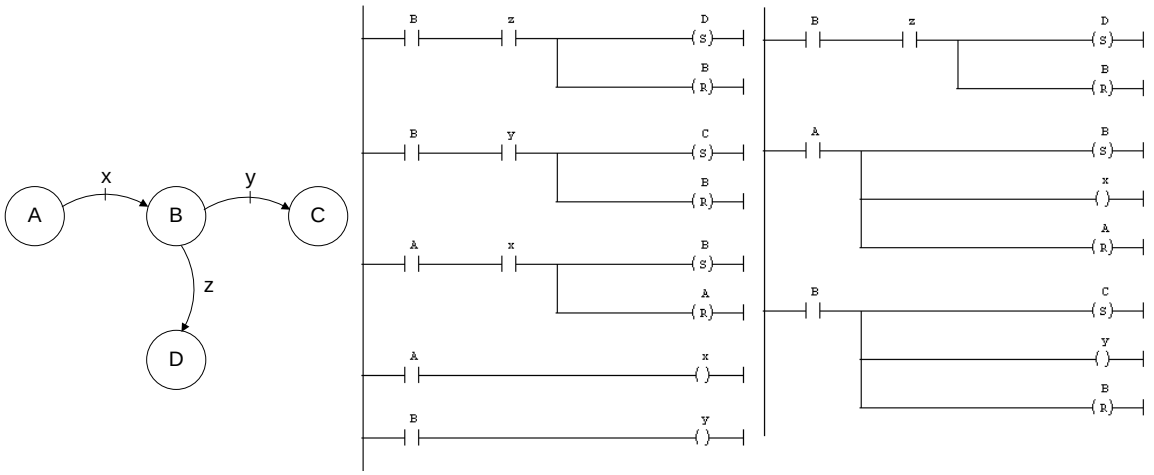


Abbildung 2.9.: Automat mit steuerbaren Ereignissen

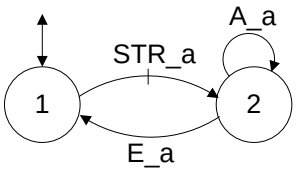


Abbildung 2.10.: Beispiel Ausgänge setzen nach Uzam [45]

2. Bei nicht selbsthaltenden Ausgängen wird der Ausgang für die Zeit von einem Programmzyklus gesetzt und danach wieder zurückgesetzt.

Diese Probleme werden in der Literatur häufig umgangen, in dem die Ausgänge in unterlagerten Steuerungen gesetzt werden. So können auch Zeitabhängigkeiten mit verarbeitet werden, ohne dass zeitbewertete Automaten verwendet werden müssen. Andere Lösungsansätze werden in [21] und in [45] vorgestellt. Uzam et al. definieren in [45] dafür sogenannte Aktionen, die für die Ausgänge der SPS stehen. Diese Aktionen werden als nicht steuerbare Ereignisse deklariert und in den Zuständen, in den die Ausgänge geschaltet werden sollen, als Selfloops an den Zustand gesetzt. So wird der Ausgang so lange aktiv gesetzt, bis der Zustand wieder verlassen wird. Die Aktionsereignisse dürfen ausschließlich als Selfloops an Zustände gesetzt werden, nicht aber als Zustandsübergang verwendet werden. Diese Methode setzt immer eine Rückmeldung der abgeschlossenen Aktion voraus, wie etwa eine Meldung, das eine Position erreicht oder das Ventil geschlossen wurde. Ist dies nicht vorhanden, so wird ein Ereignis benötigt, durch das auf die Beendigung des Vorgangs geschlossen werden kann.

Abbildung 2.10 zeigt einen Beispielgenerator, im dem durch das steuerbare Ereignis „STR_a“ ein Ausgang gesetzt werden soll. Der Ausgang „A_a“ wird als Selfloop an den Zustand 2 gesetzt und bleibt solange aktiv, bis die Rückmeldung „E_a“ erzeugt wird.

2.8. Entwicklungswerkzeuge

In diesem Abschnitt werden die in dieser Arbeit verwendeten Entwicklungswerkzeuge kurz thematisiert. Im ersten Teil wird das Entwicklungstool Simatic STEP 7 von Siemens beschrieben. Darauf hin wird im zweiten Abschnitt ein Tool zur Modellierung, Simulation und Verifikation von ereignisdiskreten Systemen vorgestellt.

2.8.1. Simatic STEP 7

Die Simatic STEP 7 Software dient zur Verwaltung, Konfiguration und Programmierung der Siemens S7 Komponenten. Sie beinhaltet Editoren zur Konfiguration der Siemens Hardware und zum Programmieren der S7 CPUs. Dabei werden die Sprachen KOP, AWL, FUP, S7-SCL und S7- Graph, die teilweise von der DIN-19226-4 [31] abweichen, unterstützt. Zusätzlich bietet STEP 7 durch die S7-PLCSIM die Möglichkeit, das Verhalten der Steuerung in einer virtuellen SPS zu simulieren.

2.8.2. Entwicklungswerkzeug DESTool

Für die Anwendung der SCT sind Entwicklungswerkzeuge zur Erstellung und Untersuchung ereignisdiskreter Systeme unabdingbar. In diesem Abschnitt wird das frei erhältliche Werkzeug DESTool vorgestellt, das die Modellierung, den Steuerungsentwurf und die Verifikation erleichtert [29]. DESTool ist ein an der Universität Erlangen entwickeltes Tool. Es bietet eine graphische Bedieneroberfläche, in der sich die Automaten per „Drag and Drop“ erstellen lassen. Durch eine große Anzahl an Funktionen können viele der bekannten Operationen an den Automaten realisiert und so die gesamte Synthese der Supervisor durchgeführt werden. Darüber hinaus bietet es die Möglichkeit, das Verhalten der gesteuerten Strecke mit dem Supervisor zu simulieren.

DESTool verwendet quelloffene C++ Bibliotheken, wie zum Beispiel LibFAUDES [30]. Diese Bibliothek für ereignisdiskrete Systeme beinhaltet Datenstrukturen und Algorithmen zur Bearbeitung von endlichen Automaten und reguläre Sprachen.

In Tabelle 2.1 sind die in dieser Arbeit verwendeten Funktionen von DESTool aufgelistet. Im Skriptfenster können über das Menü, neben anderen, die sogenannten „CoreFaudes“ und

Funktion	Beschreibung
isAccessible	Prüft, ob G nur erreichbare Zustände besitzt
isCoaccessible	Prüft, ob G nur ko- erreichbare Zustände besitzt
isDeterministic	Prüft, ob G deterministisch ist
isNonblocking	Prüft, ob zwei Generatoren nichtblockierend sind
isTrim	Prüft, ob G „überflüssige“ Zustände besitzt
Parallel	Führt die SYPC-Operation aus
Product	Führt die SPC-Operation aus
SelfLoop	Führt die inverse Projektion durch
isPrefixClosed	Prüft G auf Präfix-Geschlossenheit
isControllable	Prüft Spezifikation auf Steuerkeit bzgl. G
SupCon	Berechnet supremale steuerbare Teilsprache
SupReduce	Berechnet reduzierten Supervisor

Tabelle 2.1.: verwendete DESTool Operationen

„Synthesis“ Operationen eingefügt werden. Dabei beinhaltet „CoreFaudes“ Operationen auf den Generator selbst (z.B. SYPC, SPC und TRIM). „Synthesis“ hingegen bietet die Menge an Operatoren, die zur SCT gehören, wie z.B. Prüfung der Steuerbarkeit und der Beobachtbarkeit, aber auch Algorithmen zur Berechnung der supremalen steuerbaren Teilsprache und der Supervisor Reduzierung.

3. Beschreibung der Fertigungszelle

In dieser Arbeit wird eine Fertigungszelle zur Realisierung eines Stückgutprozesses automatisiert. Der Prozess dient zur Sortierung von Werkstücken in unterschiedlicher Farbe und Qualität. Die Anlage besteht aus einer Lagereinheit mit Schwenkarm, einer Transporteinrichtung, einem Handling Portal, sowie einer Robotereinheit. Diese Komponenten und die automatisierungstechnische Hardware werden in diesem Kapitel vorgestellt und beschrieben.

3.1. Nomenklatur und Anlagenschema

Die Namen der Ereignisse werden wie folgt zusammengesetzt:

Ereignisklasse_Komponente_Aktion.

Die Ereignisklassen und Komponenten werden in Tabelle 3.1 erläutert. Die Aktion bezieht sich auf den zu erreichenden Status. Sie wird durch eine Position (vor, zurück, Lager etc.) oder einer anderen Eigenschaft (an, aus, schwarz etc.) beschrieben.

Abbildung 3.1 zeigt das Anlagenschema in Form eines Blockschaltbildes. Es zeigt, in welcher Weise die Anlagenkomponenten miteinander verschaltet sind. Die Pfeile repräsentieren dabei den Fluss der Werkstücke. Das Lager kann durch den Ausschieber ein Werkstück in die Warteposition vor dem Lager schieben. Von dort aus kann der Schwenkarm das Werkstück entnehmen und es auf den Transportschlitten positionieren. Er kann auch Werkstücke vom Transportschlitten entnehmen und diese in die Warteposition setzen. Der Transportschlitten kann zwischen den Haltepositionen am Schwenkarm und am Handling Portal bidirektional bewegt werden, so dass sowohl Werkstücke vom Lager zum Handling Portal, als auch vom Handling Portal zum Lager transportiert werden können. Auch das Handling Portal kann prinzipiell Werkstücke auf den Transportschlitten setzen bzw. vom Transportschlitten entnehmen. Die vom Transportschlitten entnommenen Werkstücke können auf die Rutschen gelegt werden und der Roboter kann die Werkstücke in das Lager zurückführen, auf die Akzeptabel- und Ausschusspositionen legen oder zur RFID Station bringen.

Hardwareklasse	Beschreibung
A	Beschreibt einen Ausgang der SPS, Ereignisse dieser Klasse sind generell nicht steuerbar
E	Beschreibt einen Eingang der SPS, Ereignisse dieser Klasse sind generell nicht steuerbar
STR	Beschreibt steuerbare Ereignisse, wie zum Beispiel Steuerbefehle
WS	Beschreibt Ereignisse, die durch die RFID Station erzeugt werden
Komponenten	**Beschreibung**
LA	Lagereinheit
LA_AS	Ausschieber der Lagereinheit
SA	Schwenkarm
LI	Transporteinheit (Linearachse)
HU	Handling Unit
R	Robotereinheit
LS	Lichtschranke

Tabelle 3.1.: Ereignis-Notation

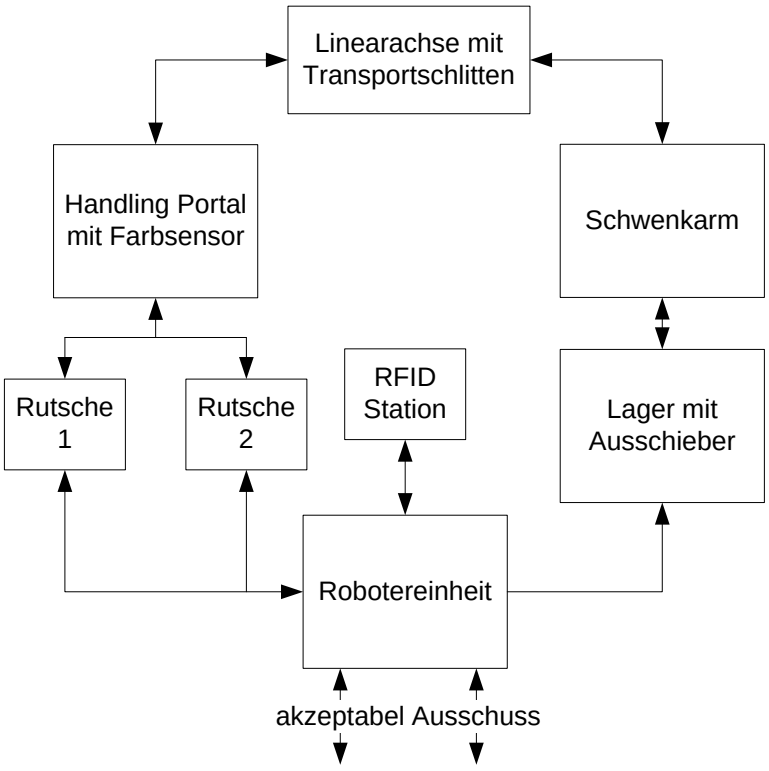


Abbildung 3.1.: Anlagenschema der Fertigungszelle

3.2. Aufbau der Fertigungszelle

In diesem Abschnitt werden die einzelnen Komponenten der Fertigungszelle beschrieben. Zunächst werden die FESTO-Anlagenkomponenten, anschließend die Robotereinheit, Transporteinheit und die automatisierungstechnische Hardware vorgestellt.

3.2.1. Lagereinheit mit Schwenkarm

Die Lagereinheit besteht aus einem Stapelmagazin, einem Ausschieber und aus einem Schwenkarm, auch Umsetzer genannt. Das Stapelmagazin stellt das Lager für die Werkstücke dar und beinhaltet folgende Sensoren:

- Endschalter des Ausschiebers „eingefahren“,
- Endschalter des Ausschiebers „ausgefahren“ und
- Lichtschranke „Lager leer.“

Der Schwenkarm entnimmt dem Lager die Werkstücke und setzt diese auf die Transporteinrichtung. Dafür besitzt er eine Vakuumeinheit, mit der er die Werkstücke ansaugen und somit festhalten kann. Er verfügt über folgende Sensoren:

- Endschalter an der Position Lagereinheit,
- Endschalter an der Position Transporteinrichtung und
- Werkstückerkennung an der Vakuumeinheit

Das Ausfahren des Ausschiebers kann aktiv durch das Ansteuern des dazugehörigen Ventils gesteuert werden. Das Einfahren erfolgt durch eine Feder, sobald das Ventil wieder geschlossen wurde. Der Schwenkarm kann durch Öffnen der jeweiligen Ventile in die Position „Lagereinheit“ bzw. „Transporteinrichtung“ gefahren werden. Zum Ansaugen eines Werkstücks am Aufnehmer des Schwenkarms wird ein Vakuum erzeugt. Das Lösen des Werkstücks wird durch Druckluft realisiert. Abbildung 3.2 skizziert die Komponenten Lager und Ausschieber.

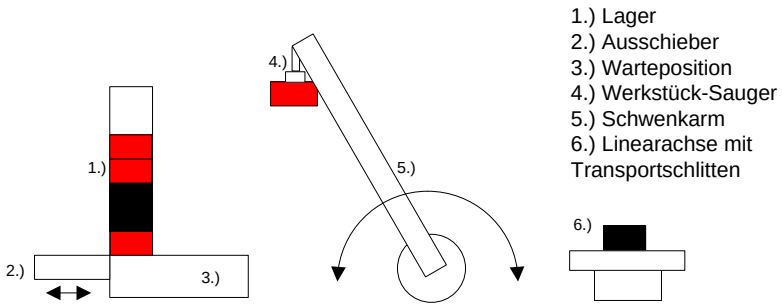


Abbildung 3.2.: Lagereinheit mit Ausschieber und Schwenkarm

3.2.2. Transporteinrichtung

Die Transporteinrichtung dient zum Transportieren der Werkstücke vom Lager zum Handling Portal. Sie besteht aus einer isel-Doppelspureinheit, Schrittmotor und einem Kugelgewindetrieb mit Transportschlitten. Die Doppelspureinheit besteht aus einem Aluminiumprofil mit den Maßen $1000mm \times 92mm \times 70mm$ (L/B/H). Der Schrittmotorantrieb ermöglicht eine Wiederholungsgenauigkeit von $\pm 0,01mm$, so dass eine Positionsreproduzierbarkeit sichergestellt werden kann. Der Transportschlitten hat eine Länge von 125mm und bietet Platz für fünf Werkstücke. Der Schrittmotor wird durch die Positionierbaugruppe FM 353 angesteuert (siehe Abschnitt 3.2.6). Abbildung 3.3 zeigt eine Zeichnung der Transporteinrichtung.

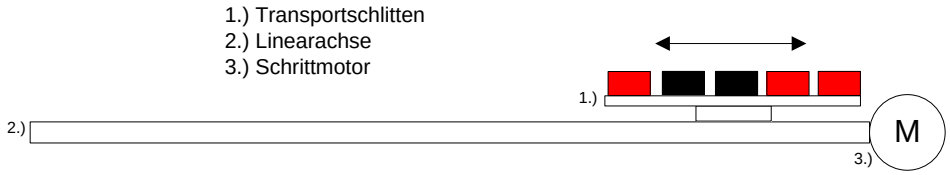


Abbildung 3.3.: Transporteinheit

3.2.3. Handling Portal

Das Handling Portal ist eine zwei-Achs-Handhabungseinrichtung, die den Greifer in horizontale und vertikale Richtung bewegen kann. Es können die Positionen „Transporteinrichtung“, „Rutsche 1“ und „Rutsche 2“ durch Positionsschalter detektiert werden. Dabei lässt sich der Greifer senken, öffnen und schließen. Das Handling Portal verfügt über folgende Sensoren:

- Positionsschalter der Position „Transporteinrichtung“,

- Positionsschalter der Position „Rutsche 1“,
- Positionsschalter der Position „Rutsche 2“,
- Positionsschalter der Position „Greifer unten“,
- Positionsschalter der Position “Greifer oben“ und
- Farbsensor zur Erkennung von roten Werkstücken im Greifer.

Der Farbsensor im Greifer detektiert rote Werkstücke, schwarze jedoch nicht. So ist es möglich, schwarze und rote Werkstücke zu unterscheiden. Der Greifer ist im Ruhezustand geschlossen und in der Position „oben“. Er kann durch das Ansteuern des entsprechenden Ventils geöffnet bzw. abgesenkt werden. Durch eine horizontale Achse kann der Greifer pneumatisch in Richtung Transporteinheit bzw. in Richtung der Rutschen gefahren werden. Abbildung 3.4 zeigt eine Skizze des Handling Portals.

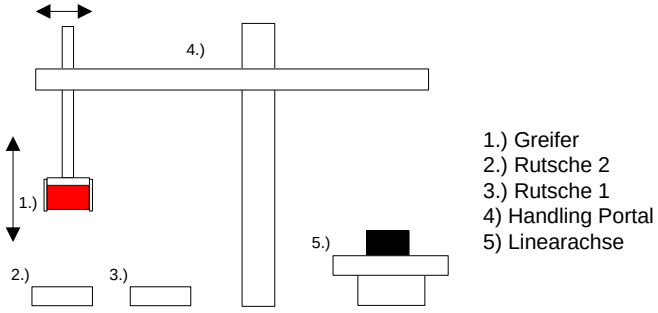
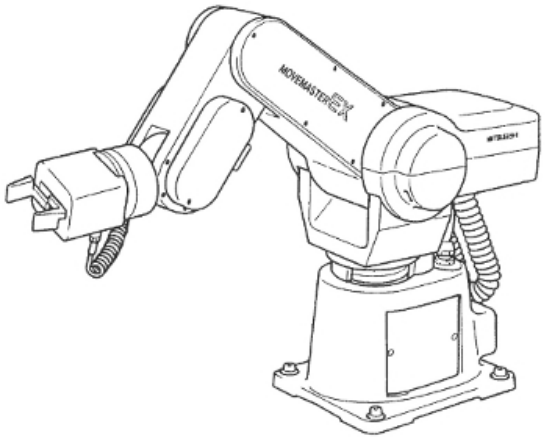


Abbildung 3.4.: Handling Portal

3.2.4. Robotereinheit

Die Robotereinheit besteht aus einem Vertikal-Roboter mit 5 Freiheitsgraden, einer Drive Unit zum Steuern des Roboters und einer motorgesteuerten Hand. Die Drive Unit ist über die serielle Schnittstelle RS-232 mit dem Steuerungsrechner verbunden. Dieser Steuerungsrechner besitzt eine Profinet-Anbindung, über die er mit der SPS kommuniziert. Die Programmierung des Roboters erfolgt über den Steuerungsrechner mit Hilfe von Visual Basic 6.0. Durch die Profinet-Anbindung des Steuerungsrechners wird ermöglicht, das Verhalten des Roboters anhand von Merkerbits aus der SPS zu steuern.



Quelle: http://hmt.fh-duesseldorf.de/hmt/images/3/33/Movemaster_EX_300px.jpg

Abbildung 3.5.: Roboter als Skizze

Eine Nestfahrt bildet den Initialisierungszustand des Roboters, um einen definierten Startzustand zu erlangen. Dabei werden alle Endschalten des Roboters langsam angefahren. Ohne diese Initialisierung kann keine Verbindung mir der SPS aufgebaut werden.

Über eine von Crescent Software Inc. vorgefertigte Methode können Strings direkt an die Drive Unit gesendet werden. Die Drive Unit übernimmt die Lageregelung der Motoren, so dass die gesendeten Positionen angefahren werden. Dadurch kann jede beliebige Position im Arbeitsbereich des Roboters erreicht werden. Die Software sendet einen Punkt, der auf der gewünschten Trajektorie liegt und wartet solange, bis dieser angefahren wurde. Danach wird der nächste Punkt gesendet. Das Programm zur Ansteuerung des Roboters wird in Abschnitt 8.1 weiter erläutert.

Der Roboter kann durch die Steuersoftware auf dem PC sechs Trajektorien abfahren:

- Werkstück von der Rutsche 1 zur RFID Station transportieren,
- Werkstück von der Rutsche 2 zur RFID Station transportieren,
- Werkstück von RFID Station auf die Position „akzeptabel“ setzen,
- Werkstück von RFID Station auf die Position „Ausschuss“ legen,
- Werkstück von RFID Station in das Lager rückführen und
- Ruheposition einnehmen.

3.2.5. RFID Station

Zur Identifizierung und Markierung der Werkstücke wird ein RFID Identifikationssystem von Siemens verwendet. Jedes Werkstück ist mit einem SIMATIC RF320T Transponder versehen. Dieser ermöglicht das Abspeichern einer Datenmenge von 20 Byte. Ausgelesen und beschrieben werden die Transponder mit Hilfe der Schreib- und Leseeinheit SIMATIC RF340R. Der RF340T gestattet das Einlesen und Beschreiben der Transponder in einem maximalen Abstand von 60mm. Die Schreib- und Leseeinheit ist durch die Kommunikationseinheit SIMATIC RF180C mit dem Profinet und somit auch mit der SPS verbunden. Weitere Informationen können dem Systemhandbuch [40] entnommen werden. Die konkrete Implementierung auf der SPS ist der Beschreibung des FB45 [44] zu entnehmen.

3.2.6. Automatisierungstechnische Hardware

Das Zentrum der automatisierungstechnischen Hardware bildet die Siemens SIMATIC S7-300 Station. Als CPU wird eine CPU 315-2 PN/DP mit folgenden Eigenschaften verwendet:

- Datenbereich Merker: 2048 Byte
- Maximale Anzahl der Bausteine (FCs, FBs, DBs): 1024
- PN Schnittstelle (z.B. Profinet)
- DP Schnittselle (Dezentrale Peripherie, Profibus).

Ergänzt wird die S7-300 Station durch zwei digitale Ein- und Ausgabebaugruppen. Sie verfügen über jeweils 16 Ein- und 16 Ausgängen. Sie belegen die Ausgangsbytes (AB) 4,5,12 und 13.

Die Kommunikation mit dem S7 Programmiergerät, der Robotereinheit und der Visualisierungseinrichtung erfolgt über dem echtzeitfähigen Feldbus Profinet. Die Komponenten werden durch einen SCALANCE X208 Switch in einer Sterntopologie verbunden.

Zur Ansteuerung des Schrittmotors der Transporteinrichtung wird die Positionierbaugruppe FM 353 verwendet. Die mikroprozessorgesteuerte Positionierbaugruppe dient zur Ansteuerung und Positionierung von Schrittmotoren und arbeitet selbstständig. Sie kann über das Anwenderprogramm in STEP7 gesteuert werden. Die FM 353 muss im SIMATIC Manager konfiguriert und parametriert werden. Weitere Informationen hierzu und zur Programmierung sind in [39] zu finden. Die Positionierbaugruppe wird direkt mit dem Isel Antriebsmodul verbunden.

4. Konzeption

In diesem Kapitel erfolgen Abwägungen und Entscheidungen bezüglich der Beschreibungsformen, der strukturellen Ansätze und der Entwicklungstools. In Abschnitt 4.1 werden die möglichen Beschreibungsformen in Bezug auf die Fertigungszelle miteinander verglichen und bewertet. Im Abschnitt 4.2 werden die in 2.6 vorgestellten strukturellen Ansätze gegenübergestellt und schließlich ein Ansatz ausgewählt, der auf die Fertigungszelle angewendet wird. Im letzten Teil des Kapitels werden Entwicklungstools durch allgemeine Kriterien verglichen und entschieden, mit welchem Tool die Fertigungszelle modelliert wird.

4.1. Auswahl der Beschreibungsform

Grundsätzlich werden zwei Varianten von Beschreibungsformen für ereignisdiskrete Systeme unterschieden. Im Abschnitt 2.2 wurden die Automaten als mögliche Beschreibungsform vorgestellt. Eine weitere Variante sind die Petri-Netze, die als eine Erweiterung der Automaten gesehen werden können. Beide Beschreibungsformen bieten Vor- und Nachteile. Zur Entscheidung, welche Beschreibungsform geeignet ist, ergeben sich Faktoren, die abzuwägen sind. Die Vergleichskriterien sind in Tabelle 4.1 als Entscheidungsmatrix aufgelistet. Ein wichtiges Kriterium ist die Modellkomplexität. Petri-Netze eignen sich besonders gut zur Darstellung von parallelen Prozessen. Die Automaten werden bei parallelen Prozessen äußerst komplex, so dass sie nicht gut dafür geeignet sind. Bei sequentiellen Prozessen unterscheiden sich die beiden Beschreibungsformen nicht.

Weitere wichtige Faktoren sind (neben denen, die von der Anlage abhängen) die formalen Faktoren. Dazu zählen die verfügbaren Tools zur Implementierung und Synthese der Supervisor und die Analysierbarkeit der Modelle bzw. der Supervisor. Besonders für die Synthese der Supervisor gibt es für Automaten eine Vielzahl an Tools, in denen die Synthese und Supervisor Reduzierung implementiert sind. Auch die Analysierbarkeit ist für Automaten deutlich einfacher. Die Erreichbarkeitsanalyse der Petri-Netze beinhaltet die Umwandlung der Netze in einen Erreichbarkeitsgrafen. Die Codegenerierung ist für Automaten in den üblicherweise verwendeten Tools in der Regel nicht implementiert. Für Petri-Netze gibt es hingegen diverse Tools, die diese Aufgabe übernehmen.

		Automaten	**Petri-Netze**
Vergleichskriterien	Faktor	Bewertung	Bewertung
Formale Analysierbarkeit	5	5 (++)	3 (o)
Modellkomplexität bei parallelen Prozessen	2	2 (-)	5 (++)
Modellkomplexität bei sequentiellen Prozessen	5	5 (++)	5 (++)
Tools zur automatischen SCT Synthese	5	5 (++)	2 (-)
kostenlose Entwicklungstools	4	5 (++)	1 (- -)
Tools zur automatischen Codegenerierung	3	3 (o)	5 (++)
Ausgänge an SPS im Modell definiert	1	3 (o)	4 (+)
Gesamte Punkte		**111**	**83**

Tabelle 4.1.: Entscheidungsmatrix Beschreibungsform

Zur Entscheidung, welche Beschreibungsform verwendet werden kann, müssen die Vergleichskriterien gewichtet werden. Bewertet werden die Kriterien mit dem Faktor 5 für „sehr wichtig“ bis 1 „unwichtig“. Die Gewichtungen „++“ für „sehr gut“ bekommen einen Faktor 5, „+“ den Faktor 4 bis „- -“ den Faktor 1. Die formale Analysierbarkeit erhält den höchsten Faktor, genauso wie die Modellkomplexität bei sequentiellen Prozessen und die automatische SCT Synthese.

Die formale Analysierbarkeit, sowie die automatische SCT Synthese, sind essentiell für die effektive Anwendung der SCT auf reelle, komplexe Systeme. Die Modellkomplexität bei parallelen Prozessen ist für die Fertigungszelle nicht relevant, da wenig parallele Prozesse ablaufen bzw. die Parallelität der Prozesse nicht zwingend erforderlich ist. Daher erhält dieses Kriterium in dieser Entscheidung die Gewichtung 2.[1]

Tools zur automatischen Codegenerierung sind für die Anwendung der SCT auf reelle Systeme zwar hilfreich, aber der Code kann auch manuell erzeugt werden. Desweiteren kann, bei bestimmten Entwicklungswerkzeugen (siehe Abschnitt 2.8.2) ein Codegenerator erstellt werden. Mit diesen Faktoren ergibt sich die gewichtete Entscheidungsmatrix aus Tabelle 4.1.

Nach der Gewichtung der Kriterien steht die Entscheidung für Automaten als Beschreibungsform für die Fertigungszelle fest.

[1] In anderen Anwendungen kann dieses Kriterium durchaus eine sehr hohe Gewichtung erhalten, da die Komplexität der Automaten nicht mehr umsetzbar ist.

4.2. Auswahl des strukturellen Ansatzes

Zur Auswahl des strukturellen Ansatzes stehen insgesamt vier Ansätze zur Verfügung. Sie wurden im Abschnitt 2.6 vorgestellt und beschrieben. Zunächst erfolgt ein Vergleich der Ansätze. Als Vergleichskriterien werden acht Streckeneigenschaften verwendet, die in Tabelle 4.2 aufgeführt sind. Verglichen werden der modulare Ansatz, der lokal-modulare Ansatz, der dezentrale Ansatz und der monolithische Ansatz.

Der modulare Ansatz ist besonders geeignet, wenn alle Ereignisse global zur Verfügung stehen und die Spezifikationen gut partitionierbar sind. Nachteil des Ansatzes besteht darin, dass zum Test auf Nichtblockieren der Steuerung ein Gesamtmodell der Strecke verfügbar sein muss. Der lokal-modulare Ansatz unterscheidet sich vom modularen Ansatz hauptsächlich darin, dass kein Gesamtmodell der Strecke erforderlich ist und die Supervisor in strukturierter Form vorliegen. Dadurch sinkt die algorithmische Komplexität bei der Supervisor Synthese und der Modellanalyse. Dies ist aber nur der Fall, wenn die Strecke viele asynchrone Komponenten enthält. Die Implementierung wird aufwendiger, da nach [35] neben den modularen Supervisor auch das Produktsystem implementiert wird. Dadurch steigt auch der Bedarf an Speicherplatz auf der SPS.

Der dezentrale Ansatz ist dem modularen Ansatz sehr ähnlich. Der große Unterschied besteht darin, dass dieser Ansatz seine Vorteile bei verteilten Steuerungen zeigt, bei denen nicht das gesamte Ereignisalphabet global in jeder Steuerung zur Verfügung steht. Der monolithische Ansatz zeigt für komplexere Systeme keine Vorteile. Die Strecke und der Supervisor weisen eine hohe Modellkomplexität auf, so dass eine Erstellung nahezu unmöglich ist.

Die Vergleichskriterien werden in Tabelle 4.2 mit Faktoren gewichtet. Sie werden auf die in Kapitel 3 beschriebene Fertigungszelle bezogen und von 5 „trifft voll zu“ bis 1 „trifft nicht zu“ gewertet. Die Anlage besitzt eine einzige SPS, in der alle Ereignisse vorhanden sind, so dass das Kriterium „alle Ereignisse vorhanden“ mit einer 5 bewertet wird. Die Strecke ist gut in ihre Komponenten strukturierbar, so dass dieses Kriterium mit einer 4 bewertet wird. Die gleiche Aussage gilt für die Spezifikationen.

Mit Hilfe eines Venn-Diagramms kann überprüft werden, ob die Anlage asynchrone Prozesse enthält. Dafür werden aus den einzelnen Teilmodellen der Generatoren aus Kapitel 5 die Kompositionssysteme gebildet und die Ereignisalphabete zusammengefasst:

$$G_1' = G_1 \text{ mit, } \Sigma_{G_1}' = \Sigma_{G_1} \tag{4.1}$$

$$G_2' = G_{20} \| G_{21} \| G_{22} \text{ mit, } \Sigma_{G_2}' = \Sigma_{G_{20}} \cup \Sigma_{G_{21}} \cup \Sigma_{G_{22}} \tag{4.2}$$

$$G_3' = G_3 \text{ mit, } \Sigma_{G_3}' = \Sigma_{G_3} \tag{4.3}$$

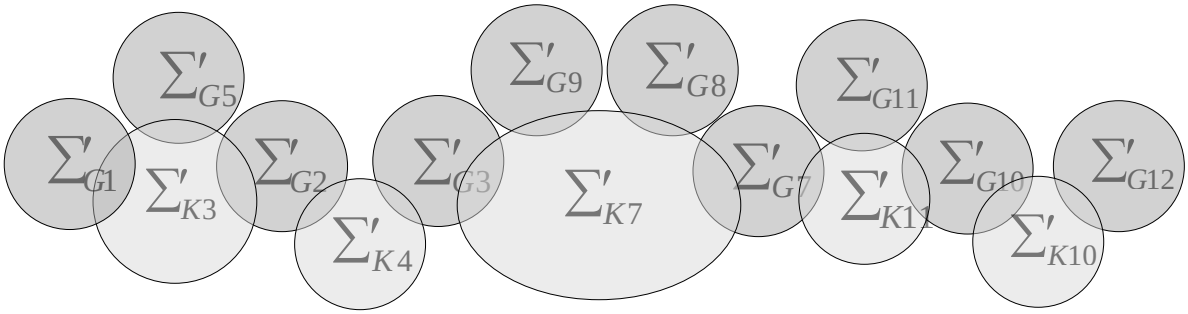


Abbildung 4.1.: Venn-Diagramm der Fertigungszelle

$$G'_5 = G_5 \text{ mit, } \Sigma'_{G_5} = \Sigma_{G_5} \tag{4.4}$$

$$G'_7 = G_{71} \| G_{72} \| G_{73} \text{ mit, } \Sigma'_{G_7} = \Sigma_{G_{71}} \cup \Sigma_{G_{72}} \cup \Sigma_{G_{73}} \tag{4.5}$$

$$G'_8 = G_8 \text{ mit, } \Sigma'_{G_8} = \Sigma_{G_8} \tag{4.6}$$

$$G'_9 = G_9 \text{ mit, } \Sigma'_{G_9} = \Sigma_{G_9} \tag{4.7}$$

$$G'_{10} = G_{10} \text{ mit, } \Sigma'_{G_10} = \Sigma_{G_10} \tag{4.8}$$

$$G'_{11} = G_{111} \| G_{112} \| G_{113} \text{ mit, } \Sigma'_{G_{11}} = \Sigma_{G_{111}} \cup \Sigma_{G_{112}} \cup \Sigma_{G_{11}3} \tag{4.9}$$

$$G'_{12} = G_{121} \| G_{122} \| G_{123} \text{ mit, } \Sigma'_{G_{12}} = \Sigma_{G_{121}} \cup \Sigma_{G_{12}2} \cup \Sigma_{G_{123}} \tag{4.10}$$

Aus den Spezifikationen der Komponenten aus Kapitel 6.2 werden die Ereignisalphabete Σ'_{Kx} gebildet. Im Venn-Diagramm lassen sich die durch die Synchronisation entstandenen Schnittmengen der Ereignisalphabete der Streckenkomponenten erkennen. Daraus ist abzuleiten, dass das eingeschränkte System wieder das Kompositionssystem ergibt. Demnach gibt es keine asynchronen Komponenten, so dass das Kriterium „wenig asynchrone Prozesse“ mit „trifft voll zu“ also einer 5 bewertet wird.

Der Aufwand der Implementierung ist nicht unerheblich bei der Implementierung, ist aber kein Kriterium, das die Entscheidung sehr stark beeinflussen soll. Alle Varianten der Implementierung sind durchführbar. Daher wird dieses Kriterium mit einer 3 bewertet. Für die Implementierung ist jedoch relevant, dass die Modellkomplexität des Supervisors so gering wie möglich ist. Aus diesem Grund wird dieses Kriterium mit einer 5 bewertet. Ebenso wird die algorithmische Komplexität bewertet, da diese im Entwurfs- und Verifikationsprozess eine große Rolle spielt.

Weniger relevant hingegen ist das benötigte Gesamtmodell der Strecke. Das Kompositionsmodell dieser Anlage ist mit den heutigen PCs handhabbar. Auch der benötigte Speicherplatz ist mit der verwendeten CPU 315-2 PN/ DP kein signifikanter Faktor.

Die Tabelle 4.2 zeigt die Entscheidungsmatrix aus der hervorgeht, dass der monolithische

Streckeneigenschaften	Faktor	Modularer Ansatz	Lokal-modularer Ansatz	dezentraler Ansatz	monolit. Ansatz
Alle Ereignisse sind vorhanden	5	5 (++)	4 (+)	1 (- -)	5 (++)
Strecke gut strukturierbar	4	4 (+)	5 (++)	4 (+)	2 (-)
Spezifikationen gut strukturierbar	4	5 (++)	5 (++)	5 (++)	2 (-)
Wenig asynchrone Prozesse	5	5 (++)	1 (- -)	5 (++)	2 (-)
Aufwand der Implemetierung	3	5 (++)	2 (-)	4 (+)	1 (- -)
Modellkomplexität des Supervisor	5	4 (+)	5 (++)	4 (+)	1 (- -)
Benötigter Speicherplatz auf der SPS	2	4 (+)	3 (o)	4 (+)	1 (- -)
Gesamtmodell der Strecke notwendig	1	2 (-)	5 (++)	2 (-)	2(-)
Algorithmische Komplexität	5	3 (o)	5 (++)	3 (o)	1 (- -)
Gesamte Punkte		146	132	122	68

Tabelle 4.2.: Entscheidungsmatrix der strukturellen Ansätze

und der dezentrale Ansatz für diese Anlage nicht geeignet sind. Sie erreichen nur 68 bzw. 122 Punkte von 160 maximal möglichen Punkten. Der modulare Ansatz bietet mit 146 von 160 Punkten den geeigneten Ansatz, wobei der lokal-modulare nur knapp dahinter liegt.

4.3. Auswahl des Entwicklungswerkzeugs

Zur praktischen Anwendung der SCT auf reelle Systeme sind Entwicklungswerkzeuge zur Modellierung, Steuerungssynthese und Analyse des Steuerkreises unabdingbar. Verglichen werden drei frei erhältliche Tools, die allesamt zur Steuerungssynthese nach der SCT entwickelt wurden. Sie werden in Tabelle 4.3 durch ihre Benutzerfreundlichkeit, Funktionsumfang und Kosten mit „+“, „o“, und „-“ bewertet und verglichen. Zur Entscheidungsfindung werden die einzelnen Faktoren mit den Werten von 1 bis 3 nach der Wichtigkeit bewertet.

Das erste Tool ist TCT. Das in Kanada an der Universität in Toronto entwickelte Tool ist das

Vergleichskriterien	Gewichtung	TCT	DESTool	Supremica
Übersichtlichkeit	2	1 (-)	3 (+)	2 (o)
Bedienbarkeit	2	1 (-)	3 (+)	2 (o)
Analysefunktionen	3	3 (+)	3 (+)	2 (o)
Codegenerierung	1	1 (-)	1 (-)	3 (+)
Freeware	3	3 (+)	3 (+)	3 (+)
Testfunktionen	1	1 (-)	3 (+)	3 (+)
Projektfiles lesbar	3	3 (+)	3 (+)	2 (o)
Dokomentation	2	3 (+)	3 (+)	3 (+)
Gesamte Punkte		39	49	41

Tabelle 4.3.: Entscheidungsmatrix Software Tools

wohl bekannteste Tool für die SCT Steuerungssynthese und findet in Publikationen wie z.B. [34], [45], [49] und [47] Anwendung. Es gilt als das Original SCT Tool [3]. Die Bedienoberfläche besteht aus einer Eingabekonsole, dessen Bedienung durch Eingabe von Befehlen erfolgt. Die Übersichtlichkeit und die Bedienbarkeit sind im Vergleich zu den grafischen Tools nachteilig. TCT verfügt über einen sehr umfangreichen Pool an Analysefunktionen. Durch die vielen Publikationen und Dokumentationen erhält man einen schnellen Einstieg in das Programm.

DESTool wurde in Abschnitt 2.8.2 vorgestellt. Es bietet eine grafische Bedieneroberfläche, in der sich die Automaten per „drag and drop“ modellieren lassen. Das Programm lässt sich intuitiv benutzen. Es bietet, neben den benötigten Operationen, viele weitere nützliche Analysefunktionen zur Untersuchung der Streckenmodelle und der Supervisor.

Supremica bietet neben einer grafischen Bedieneroberfläche einen Codegenerator, der es ermöglicht, ANSI C und IEC 61131 Code zu erzeugen. Die Bedienbarkeit ist weniger intuitiv und die Analysefunktionen beschränken sich auf die wesentlichen Funktionen (wie z.B. SYPC, SPC, Supervisor Synthese).

Die Tabelle 4.3 zeigt die gewichtete Entscheidungsmatrix. Im Vergleich der Anforderungen an die Entwicklungssoftware liegt DESTool vor TCT und Supremica. Es wird zur Modellierung der Strecke und zur Steuerungssynthese DESTool verwendet.

5. Modellierung der Strecke

In diesem Kapitel wird die Modellierung der Fertigungszelle durchgeführt. Zu Beginn des Kapitels werden allgemeine Annahmen zur Modellbildung getroffen. Diese vereinfachen die Modellbildung, die im Abschnitt 5.2 realisiert wird. Dabei wird die Anlage in einzelne Komponenten eingeteilt. Abschließend wird das gesamte Streckenmodell der Anlage mittels der entsprechenden Kompositionsoperatoren berechnet.

5.1. Annahmen zur Modellbildung und zum Steuerungsentwurf

Zur Reduzierung der Komplexität der Modellbildung und dem anschließenden Steuerungsentwurf werden die nachfolgenden Annahmen getroffen:

- Alle Hardwarekomponenten arbeiten korrekt, so dass nur der fehlerfreie Betrieb berücksichtigt wird.
- Sensoren werden ausschließlich von den Werkstücken ausgelöst. Auch Endschalter können nur durch die dazugehörigen Komponenten ausgelöst werden. Ein manuelles Betätigen wird ausgeschlossen.
- Die Hardware befindet sich nach dem Einschalten in einem definierten Zustand. Dies erfolgt durch eine Initialisierungsprozedur beim Start der Anlage. Dieser Zustand bildet den Initialisierungszustand der Komponenten.
- Die Kapazität des Lagers kann nicht überschritten werden. Auch die Plätze „Ausschuss“ und „akzeptabel“ haben eine unbegrenzte Kapazität.
- Die Rutschen, der Transportschlitten und das Handling Portal sind zur Initialisierung nicht mit Werkstücken belegt.

Name	Lichtschranke	S.on	S.off
G2_2	Sensor Schwenkarm	E_SA_KTeil	E_SA_Teil
G5	Lager	E_LA_KTeil	E_LA_Teil
G8	Farbsensor	E_HU_Greifer_rot	E_HU_Greifer_sw
G9	Entnahme	E_LS_Entnahme	E_LS_NEntnahme

Tabelle 5.1.: Instanzen der Lichtschranken nach dem Sensor Modell a

Name	Signal/Lichtschranke	S.on
G6_1	Werkstück ist akzeptabel	WS_akzeptabel
G6_2	Werkstück ist Ausschuss	WS_Ausschuss
G6_3	Werkstück ist reparabel	WS_reparable
G11_1	Rutsche 1	E_LS_Rutsche_1
G11_2	Rutsche 2	E_LS_Rutsche_2

Tabelle 5.2.: Instanzen der Lichtschranken nach dem Sensor Modell b

5.2. Komponentenmodelle

5.2.1. Sensorik

Die Fertigungszelle bietet sechs Sensoren, die losgelöst von den Komponentenmodellen agieren. Die Lichtschranken im Lager sowie an der Entnahme-Position und der Farbsensor im Handling Portal-Greifer lassen sich mit dem generischen Modell in Abbildung 5.1 Modell a) nachbilden. Die Lichtschranke liefert immer dann ein Signal (S.on), wenn das Lager leer ist, ein Werkstück die Entnahme-Position erreicht hat oder ein rotes Werkstück gegriffen wurde. Durch Negation der Signale (S.on) lassen sich die Gegenereignisse (S.off) erzeugen. Die Instanzen dieser generischen Modelle sind in Tabelle 5.1 zusammengefasst.

Die Lichtschranken an den Rutschen zum Detektieren der Werkstücke lassen sich nach dem generischen Generator in Abbildung 5.1 Modell b) nachbilden. Sie liefern immer dann ein Signal, wenn ein Werkstück auf der jeweiligen Rutsche liegt. Ebenso werden die Ereignisse, die durch die RFID-Station erzeugt werden, nachgebildet. Der Zustand muss zwingend markiert sein, da sonst nach einer Zusammenführung der Generatoren mit anderen ein leerer Generator entsteht. Die Instanzen dieser generischen Modelle sind in Tabelle 5.2 zusammengefasst.

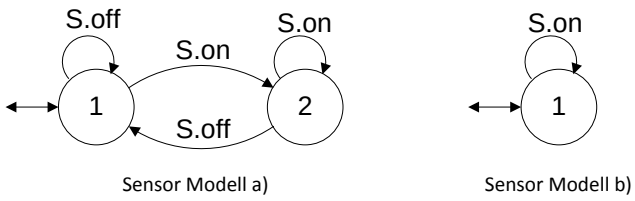


Abbildung 5.1.: Generische Modelle der Sensoren

5.2.2. Lagereinheit mit Ausschieber

Die Lagereinheit besteht aus einem Ausschieber und dem Lager selbst. Der Ausschieber wird durch das Ansteuern des Ausgangssignals „A_LA_AS_vor“ ausgefahren und ein Werkstück wird aus dem Lager herausgeschoben. Wird der Ausgang zurückgesetzt, so fährt der Ausschieber wieder in seine Ausgangsposition zurück. Hat der Ausschieber seine Endposition erreicht, so wird das Ereignis „E_LA_AS_vor“ generiert. Befindet sich der Ausschieber in seiner Ausgangsposition wird das Ereignis „E_LA_AS_zu“ generiert. Es ergeben sich drei Zustände:

1. Ausschieber ist eingefahren,
2. Ausschieber wird ausgefahren und
3. Ausschieber ist ausgefahren.

Abbildung 5.2 bildet den Ausschieber als Generator nach. Der Zustandswechsel zwischen Zustand 1 und 2 wird durch das steuerbare Ereignis „STR_LA_AS_vor“ erzwungen. Im Zustand 2 muss der Ausgang zum Ausfahren des Zustands gesetzt werden. Nach [45] werden diese Ausgänge als Aktionen definiert, die sich als nichtsteuerbare Ereignisse an den jeweiligen Zustand als Selfloop abbilden lassen. Der Zustandswechsel zwischen Zustand 2 und 3 erfolgt durch die Rückmeldung „E_LA_AS_vor“. Der Ausgang wird nicht mehr gesetzt, so dass der Ausschieber wieder zurück fährt. Es erfolgt, durch das Ereignis „E_LA_AS_zu“ der Zustandswechsel von Zustand 3 zum Initialisierungszustand 1.

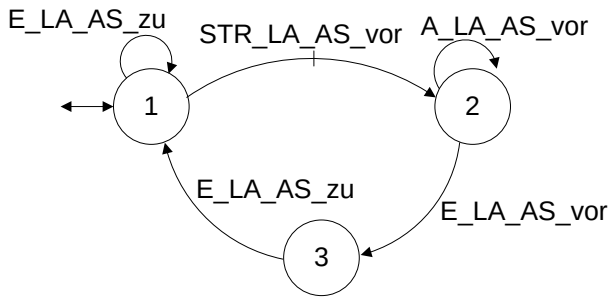


Abbildung 5.2.: Generatormodell für den Ausschieber

5.2.3. Schwenkarm mit Vakuumeinheit

Der Schwenkarm mit der Vakuumeinheit lässt sich in drei unabhängige Komponenten einteilen. So lassen sich die Bewegung des Arms, die Rückmeldung des Saugers für ein vorhandenes Werkstück und die Ansteuerung der Vakuumeinheit getrennt voneinander modellieren.

Der Schwenkarm kann drei Positionen anfahren: „Lager“, „Linearachse“ und „Mitte“. Da nur „Lager“ und „Linearachse“ Positionsendschalter besitzen, muss für die Position „Mitte“ eine unterlagerte, zeitabhängige Steuerung implementiert werden. Durch die jeweiligen steuerbaren Ereignisse „STR_SA_Position“ können die einzelnen Positionen angefahren werden. Die unterlagerte Steuerung gibt das jeweilige Rückmeldeereignis „E_SA_Position“ aus, wenn die Position erreicht wurde. Eine Änderung der Positionen ist nur möglich, wenn der Schwenkarm zuvor seine Position erreicht hat. Das Verhalten des Schwenkarms wird durch den Generator in Abbildung 5.3 nachgebildet.

Der Sensor zur Detektion eines Werkstücks am Schwenkarmsauger liefert eine binäre Rückmeldung. Wird ein Werkstück detektiert, so generiert der Sensor das Ereignis „E_SA_Teil“. Sobald kein Werkstück vorhanden ist, generiert er „E_SA_kTeil“. Es ergeben sich daraus zwei Zustände, die der Sensor einnehmen kann. Ist der Zustand „Werkstück vorhanden“ aktiv, so kann das Signal „E_SA_Teil“ erzeugt werden, ohne dass ein Zustandswechsel stattfindet. Ebenso kann im Zustand „kein Werkstück vorhanden“ das Ereignis „E_SA_Teil“ generiert werden, ohne dass sich der Zustand ändert (siehe Abschnitt 5.2.1).

Das Vakuum zum Ansaugen, sowie die Druckluft zum Abstoßen der Werkstücke sind im Initialisierungszustand nicht aktiviert. Durch das Ereignis „STR_SA_VAK_ein“ kann das Vakuum aktiviert werden. Eine unterlagerte Schrittkette öffnet das Ventil für $500ms$, schließt es anschließend wieder und meldet „E_SA_VAK_aus“ zurück. Der Initialisierungszustand wird wieder erreicht. Ebenso wird mit der Druckluft verfahren. Das Ereig-

nis „STR_SA_VAK_gegen“ startet die unterlagerte Schrittkette. Diese meldet nach $500ms$ „E_SA_VAK_gaus“ zurück.

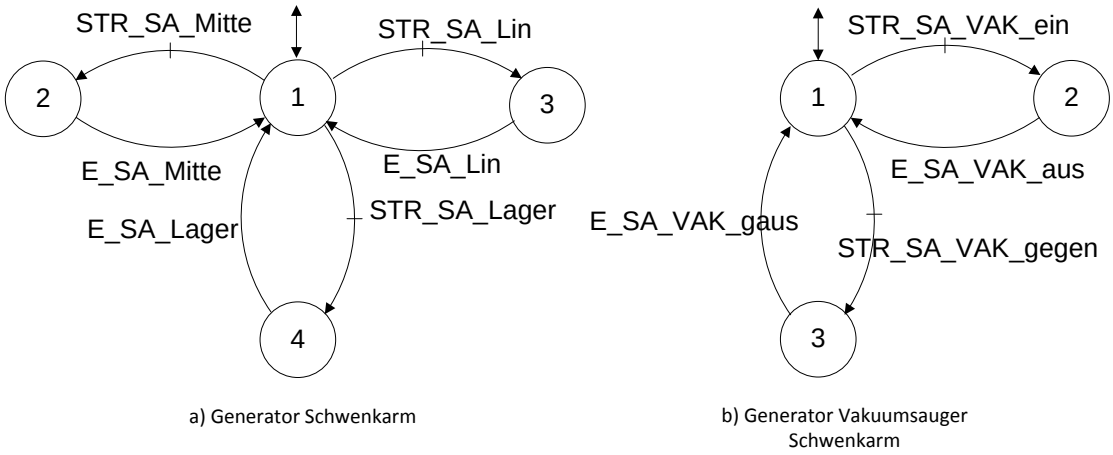


Abbildung 5.3.: Generatormodelle für den Schwenkarm

5.2.4. Transporteinrichtung

Das Verhalten der Transporteinrichtung wird durch die Linearachsenansteuerung bestimmt. Diese erlaubt, den Schlitten von einer beliebigen Position zur Entnahme- bzw. Beladungsstelle zu fahren. Zusätzlich lässt sich der Schlitten um eine definierbare Entfernung schrittweise vorfahren. Ein Positionswechsel ist erst möglich, wenn die vorherige Position angefahren wurde. Das Verhalten lässt sich in vier Zustände zusammenfassen:

1. Initialisierungszustand,
2. fahre zur Entnahmestelle,
3. fahre zur Beladungsstelle,
4. fahre eine definierte Länge vor.

Die Linearachsensteuerung erzeugt die Rückmeldung „E_LI_Position“, sobald die Position erreicht wurde. Abbildung 5.4 zeigt den Generator, der das Verhalten der Transporteinheit repräsentiert.

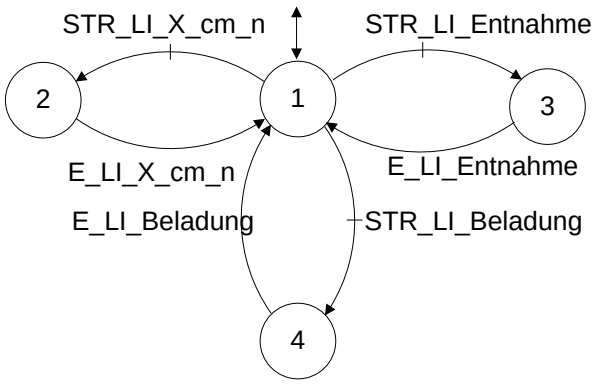


Abbildung 5.4.: Generatormodelle der Transporteinheit

5.2.5. Handling Portal mit Greifarm

Das Handling Portal kann in drei zueinander asynchrone Komponenten aufgeteilt werden. Zunächst die horizontale Achse. Sie kann in Richtung der Linearachse und in Richtung der Rutschen bewegt werden. In Richtung der Rutschen können die Positionsschalter „E_HU_Ablage_1“ und „E_HU_Ablage_2“ ausgelöst werden. Das Ereignis „E_HU_Entnahme“ wird beim Erreichen der Entnahme-Position an der Transporteinheit generiert. Die Richtungen werden durch zwei Ventile bestimmt, die durch die Ausgänge “A_HU_Ablage“ und „A_HU_Entnahme“ geöffnet werden. Es ist möglich aus jeder beliebigen Position den Steuerbefehl „STR_HU_POSITION“ zu generieren. Es ergeben sich für das Verhalten des Handling Portals vier Zustände:

1. Initialisierungszustand: keine Bewegung des Greifarms,
2. fahre zur Ablage 1 (Rutsche 1),
3. fahre zur Ablage 2 (Rutsche 2),
4. fahre zur Entnahme-Position an der Transporteinrichtung.

Der Greifarm kann unabhängig von der Achse geöffnet und geschlossen werden. Ebenso kann er in vertikaler Richtung bewegt werden. Dieses Verhalten lässt sich mit zwei Generatoren beschreiben. Das Öffnen bzw. Schließen des Greifers wird durch eine Schrittkette gesteuert. Zum Öffnen wird das Ereignis „STR_HU_Greifer_auf“ und zum Schließen „STR_HU_Greifer_zu“ generiert. Das Auf- und Abfahren des Greifers wird durch die steuerbaren Ereignisse „STR_HU_Greifer_oben“ bzw. „STR_HU_Greifer_unten“ repräsentiert. Im Initialisierungszustand ist der Greifer oben. Es wird das Ereignis „E_HU_Greifer_oben“ generiert. Für das Absenken des Greifers muss ein Ventil mit dem

Ausgang „A_HU_Greifer_unten“ geöffnet werden. Wurde die Position erreicht, wird das Ereignis „E_HU_Greifer_unten“ generiert.

5.2.6. Robotereinheit

Das Verhalten der Robotereinheit wird ausschließlich durch die im Steuerungsrechner implementierte Software definiert. Es können sechs Verfahrprogramme ausgewählt und die dazugehörigen Trajektorien abgefahren werden. Wurde die gewählte Trajektorie abgefahren, so wird die jeweilige Rückmeldung erzeugt. Das Verfahrprogramm muss den Auftrag abgeschlossen haben, bevor eine neue Trajektorie aufgegeben werden kann. Das Verhalten wird durch den in Abbildung 5.5 dargestellten Generator nachgebildet.

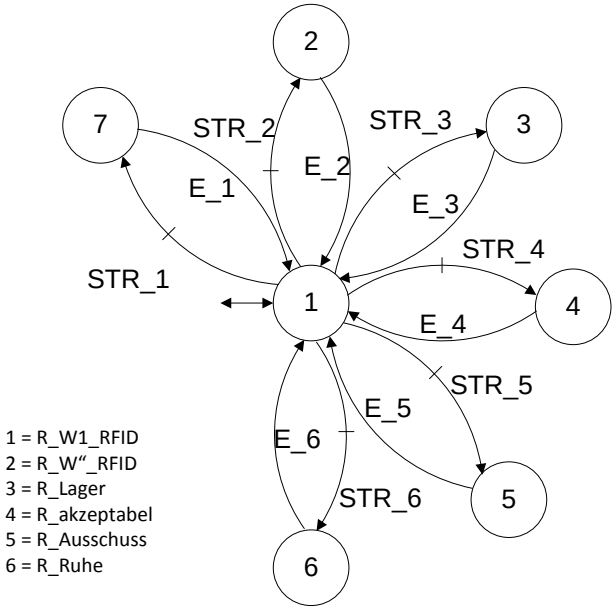


Abbildung 5.5.: Generatormodell für die Robotereinheit

5.3. Gesamtmodell der Strecke

Die Strecke besteht aus insgesamt 19 Komponenten. Sie werden im Folgenden durch G_i, $i = 1, \ldots 11$ in Module eingeteilt. Die Tabelle 5.3 listet die Namen der Komponenten auf. Der Schwenkarm besteht aus drei Komponenten, so dass das Modul durch eine Komposition zusammengeführt werden kann. Es sollen alle Komponenten unabhängig voneinander

Name	Beschreibung	**Name**	Beschreibung
G1	Auschieber Lager	G7_1	Handling Portal hor. Achse
G2_1	Schwenkarm	G7_2	Handling Portal Greifer
G2_2	Schwenkarm Sensor	G7_3	Handling Portal vert. Achse
G2_3	Schwenkarm Vakuum	G8	Farbsensor
G3	Transporteinrichtung	G9	Sensor Entnahme-Position
G4	On/Off Schalter	G10	Robotereinheit
G5	Sensor Lager	G11_1	Lichtschranke Rutsche 1
G6_1	RFID Werkstück akzeptabel	G11_2	Lichtschranke Rutsche 1
G6_2	RFID Werkstück Ausschuss	G11_3	Signal Rutschen leer
G6_3	RFID Werkstück reparabel		

Tabelle 5.3.: Streckenkomponenten

schalten können, so dass der SYPC Operator angewendet werden kann

$$G\_2 = SYPC(G2\_1, G2\_2, G2\_2). \tag{5.1}$$

Ebenso ergeben sich die Module für das Handling Portal, die RFID Rückmeldungen und der Lichtschranken der Rutschen mit

$$G\_7 = SYPC(G7\_1, G7\_2, G7\_2), \tag{5.2}$$

$$G\_6 = SYPC(G6\_1, G6\_2, G6\_2) \text{ und} \tag{5.3}$$

$$G\_11 = SYPC(G11\_1, G11\_2, G11\_2). \tag{5.4}$$

Die Komposition der Module zu dem Gesamtmodell der Strecke erfolgt ebenso durch den SYPC Operator

$$G\_Gesamt = SYPC(G\_1, G\_2, G\_3, G\_4, G\_5, G\_6, G\_7, G\_8, G\_9, G\_10, G\_11). \tag{5.5}$$

Die Komposition wurde mit der „Parallel-“ Operation von DESTool durchgeführt. Es werden nicht alle Generatoren in einem Schritt komponiert, dadurch werden Ressourcen gesparen. Ein Versuch die Komposition in einem Schritt durchzuführen führt zum Absturz von DESTool. Die Kompositionen werden in folgenden Schritten durchgeführt

$$G\_1 \| G\_2 \| G\_3 \| G\_4 = Parallel(G\_1, G\_2, G\_3, G\_4), \tag{5.6}$$

$$G\_1 \| G\_2 \| G\_3 \| G\_4 \| G\_5 = Parallel(G\_1 \| G\_2 \| G\_3 \| G\_4, G\_5), \tag{5.7}$$

$$G\_1 \| G\_2 \| G\_3 \| G\_4 \| G\_5 \| G\_6 = Parallel(G\_1 \| G\_2 \| G\_3 \| G\_4 \| G\_5, G\_6), \tag{5.8}$$

...

$$G\_Gesamt = Parallel(G\_1 \| G\_2 \| G\_3 \| G\_4 \| G\_5 \| G\_6 \| G\_7 \| G\_8 \| G\_9 \| G\_10, G\_11). \tag{5.9}$$

Durch die Komposition entsteht das Streckenmodell G_Gesamt mit $5 \cdot 10^5$ Zuständen und $16 \cdot 10^6$ Transitionen. Das Ereignisalphabet enthält 63 Ereignisse und ist in Tabelle 5.4 abgebildet.

Ereignis	c/uc	Ereignis	c/nc
STR_LA_AS_vor	c	A_HU_Ablage	uc
E_LA_AS_vor	uc	STR_HU_Greifer_zu	c
A_LA_AS_vor	uc	STR_HU_Greifer_auf	c
E_LA_AS_zu	uc	E_HU_Greifer_unten	uc
E_LS_Entnahme	uc	E_HU_Greifer_oben	uc
E_SA_Lin	uc	A_HU_Greifer_unten	uc
E_SA_Mitte	uc	STR_HU_Greifer_unten	c
STR_SA_Lager	c	STR_HU_Greifer_oben	c
STR_SA_Lin	c	E_HU_Greifer_rot	uc
STR_SA_Mitte	c	E_HU_Greifer_sw	uc
E_SA_Teil	uc	E_R_Lager	uc
E_SA_KTeil	uc	E_R_akzeptabel	uc
E_SA_Lager	uc	E_R_Ruhe	uc
STR_SA_VAK_ein	c	E_R_W1_RFID	uc
STR_SA_VAK_gegen	c	E_R_W2_RFID	uc
E_SA_VAK_aus	uc	E_R_Ausschuss	uc
E_SA_VAK_gaus	uc	STR_R_Lager	c
E_LI_Entnahme	uc	STR_R_akzeptabel	c
E_LI_Beladung	uc	STR_R_Ruhe	c
STR_LI_X_cm_n	c	STR_R_W1_RFID	c
STR_LI_Entnahme	c	STR_R_W2_RFID	c
STR_LI_Beladung	c	STR_R_Ausschuss	c
E_LI_X_cm_n	uc	E_LS_Rutsche_1	uc
E_LA_KTeil	uc	E_LS_Rutsche_2	uc
E_LA_Teil	uc	Rutschen_Leer	uc
E_HU_Ablage_2	uc	WS_akzeptabel	uc
E_HU_Entnahme	uc	WS_Ausschuss	uc
E_HU_Ablage_1	uc	WS_reparable	uc
STR_HU_Ablage_1	c	E_LS_NEntnahme	uc
STR_HU_Ablage_2	c	STR_ON	c
STR_HU_Entnahme	c	STR_OFF	c
A_HU_Entnahme	uc		

Tabelle 5.4.: Ereignisalphabet Σ_G der Strecke

6. Steuerungsentwurf

In diesem Kapitel werden die Spezifikationen und die daraus folgenden Supervisor entwickelt. Zunächst wird dafür die Steuerungsaufgabe erläutert, die durch die Steuerung erfüllt werden soll. Weiterhin erfolgt die Entwicklung der Spezifikationen für die einzelnen Komponenten der Anlage. Diese sind in modulare Steuerungen eingeteilt. Abschließend werden die Supervisor entwickelt, die die vorher entwickelten Spezifikationen erfüllen.

6.1. Steuerungsaufgabe

Es soll die Fertigungszelle zur Realisierung eines Stückgutprozesses automatisiert werden. Dazu kann der Stückgutprozess in drei Teilaufgaben aufgeteilt werden, die erfüllt werden sollen. Die Lagerung der Werkstücke ist die erste Teilaufgabe. Das Lager fasst bis zu zehn Werkstücke. Sie sollen durch den Ausschieber am Lager aus dem Lager geschoben und durch dem Schwenkarm entgegen genommen werden. Dabei ist es zu beachten, dass nicht ausgeschoben werden darf, wenn das Lager leer ist. Der Schwenkarm belädt den Transportschlitten der Linearachse, die zur zweiten Teilaufgabe (das Transportieren) gehört. Die Linearachse bietet Platz für fünf Werkstücke. Ist der Schlitten voll beladen, so transportiert er sie zum Greifportal. Es folgt die dritte Teilaufgabe, die Sortierung.

Das Handling Portal soll die Werkstücke entnehmen, sobald der Schlitten mit den Werkstücken am Portal angekommen ist. Dabei erkennt der Farbsensor, ob es sich um ein rotes oder schwarzes Werkstück handelt. Je nach Farbe soll das Werkstück auf die Rutsche für schwarze oder die für rote Werkstücke abgelegt werden. Der Roboter entnimmt das Werkstück von der Rutsche, so bald es vorhanden ist. Danach führt er es zur RFID-Station, die das Werkstück ausliest.

Die Transponder werden mit einem Wert von 1 bis 3 beschrieben, bevor die Werkstücke dem Lager zugeführt werden. Durch das Auslesen des RFID Transponders wird entschieden, ob das Werkstück „akzeptabel“, „Ausschuss“, oder „reparabel“ ist. Ist der Wert auf dem RFID Transponder gleich 1, so ist das Werkstück in Ordnung und soll vom Roboter auf die Position „akzeptabel“ transportiert werden. Der Wert 2 bedeutet, dass das Werkstück „Ausschuss“ ist. In diesem Fall soll der Roboter das Werkstück auf die Position „Ausschuss“ transportieren.

Ist der RFID Transponder mit dem Wert 3 beschrieben, so ist das Werkstück „reparabel“ und soll dem Lager zurückgeführt werden.

Die Rutschen fassen jeweils sechs Werkstücke. Es ist unbedingt zu vermeiden, dass sie überladen werden. Ebenso ist zu beachten, dass der Transportschlitten mit genau fünf Werkstücken beladen wird, bevor er zum Portal fährt. Auch der Transportschlitten darf nicht überladen werden.

Die Steuerungsaufgabe soll erfolgreich durchgeführt werden ohne dass die Anlage unerwünschte Zustände erreicht. Auch das Blockieren des Prozesses ist unbedingt zu vermeiden.

6.2. Spezifikationen

In diesem Abschnitt werden die formalen Spezifikationen entwickelt. Sie legen fest, welche Strings im ungesteuerten Verhalten unerwünscht sind. Die Spezifikationen werden als Generatoren modelliert und durch den Namen K_x gekennzeichnet. Aus den formalen Spezifikationen werden die resultierenden Supervisor erstellt.

6.2.1. Spezifikation K_3 Lagereinheit

Der Ausschieber der Lagereinheit soll nur ein Werkstück aus dem Lager schieben, wenn das Lager nicht leer ist. Zum Ausschieben muss das Ereignis „A_LA_AS_vor“ erlaubt werden, bis der Ausschieber seine Endposition erreicht hat. Nachdem ein Werkstück ausgeschoben wurde, muss es vom Schwenkarm entnommen werden, bevor das nächste Werkstück ausgeschoben werden kann. Abbildung 6.1 stellt die Spezifikation als Generator dar.

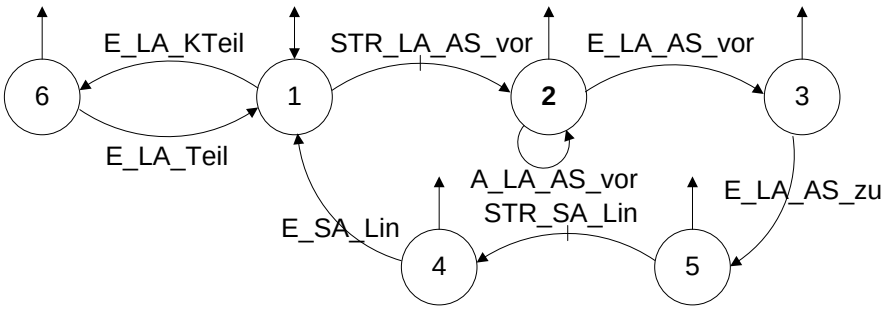


Abbildung 6.1.: Spezifikation $K3$ als Generator

6.2.2. Spezifikation K_4 und K_5 Schwenkarm

Die Spezifikation K_4 ist für die Bewegung des Schwenkarmes verantwortlich. Sie stellt sicher, dass ein Werkstück ausgeschoben wurde, bevor der Schwenkarm sich zum Lager bewegt. Der Schwenkarm darf auf keinen Fall am Lager sein während der Ausschieber ein Werkstück ausschiebt. Ansonsten würde das Werkstück zwischen dem Sauger des Schwenkarms und dem Ausschieber eingeklemmt werden. Sobald ein Werkstück am Ausgang des Lagers vorhanden ist, wird der Schwenkarm zum Lager bewegt. Hat er die Position erreicht, wird solange gewartet, bis das Werkstück angesaugt wurde. Das Werkstück wird von dem Schwenkarm auf den Transportschlitten abgesetzt. Wurde das Werkstück vom Schwenkarm gelöst, fährt der Schwenkarm die Warteposition an. Ist der Transportschlitten fertig beladen, darf der Schwenkarm keine Werkstücke entnehmen. Die Ansteuerung des Schwenkarms erfolgt über eine unterlagerte Schrittkette. Abbildung 6.2 zeigt die Spezifikation K_4 als Generator.

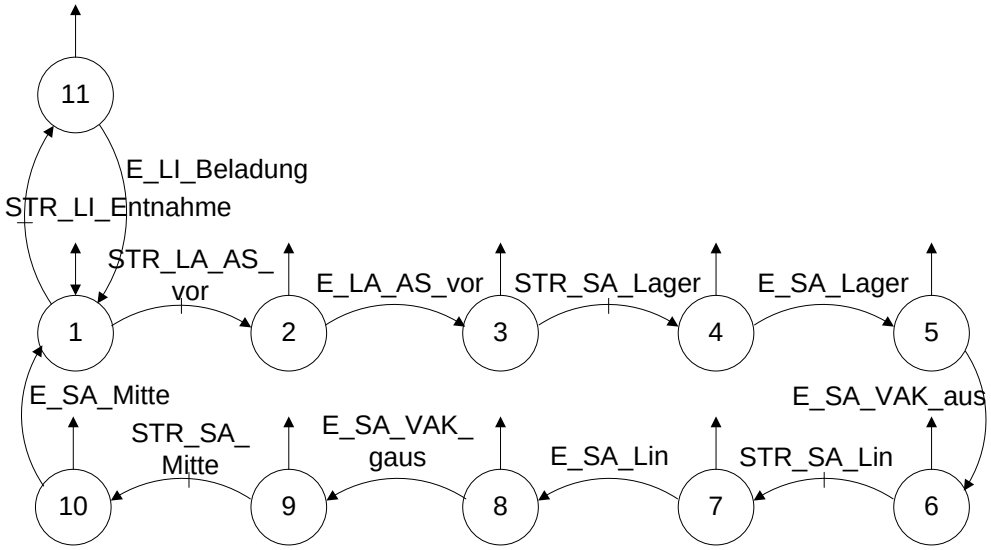


Abbildung 6.2.: Spezifikation $K4$ als Generator

Das Vakuum wird eingeschaltet, sobald der Schwenkarm am Lager angekommen ist. Wird nach dem Ausschalten des Vakuums kein Teil am Sauger detektiert, wird das Vakuum ein weiteres Mal eingeschaltet, bis ein fester Kontakt zwischen dem Sauger und dem Werkstück besteht. Erst dann darf der Schwenkarm zur Transporteinheit schwenken. Dort angekommen wird die Druckluft eingeschaltet, um das Werkstück wieder vom Sauger zu lösen. Abbildung 6.3 zeigt den Generator, der die Spezifikation K_5 repräsentiert.

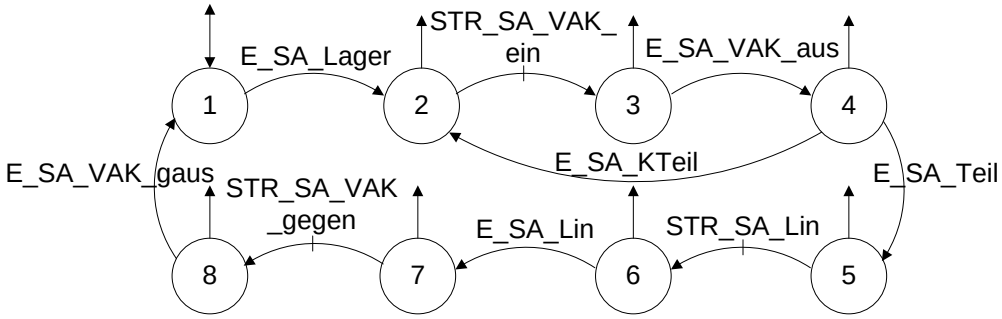


Abbildung 6.3.: Spezifikation $K5$ als Generator

6.2.3. Spezifikation K_7 Handling Portal

Die Spezifikation K_7 ist mit 19 Zuständen die größte Spezifikation der Steuerung. Sie ist verantwortlich für das Verhalten des Handling Portals. Sobald ein Werkstück an der Lichtschranke der Transporteinheit detektiert wurde, wird die Greifvorrichtung geöffnet und heruntergefahren, um das Werkstück zu greifen. Der Farbsensor benötigt eine kurze Zeit, um das Werkstück zu detektieren. Diese Zeit wird überbrückt, in dem die Greifvorrichtung hochgefahren wird. Erst wenn die Greifvorrichtung wieder oben ist, wird der Farbsensor abgefragt. Abbildung 6.4 zeigt diesen Ausschnitt der Spezifikation. Die Ereignisse der Schleifen an den Zuständen sind der Tabelle 6.1 zu entnehmen. In Abhängigkeit der Farbe wird die Rutsche für schwarze oder die Rutsche für rote Werkstücke angefahren. Dort wird das Werkstück abgesetzt und die Greifvorrichtung wieder in Richtung Linearachse gefahren. Sind weitere Werkstücke auf dem Transportschlitten vorhanden, so wird die Prozedur wiederholt.

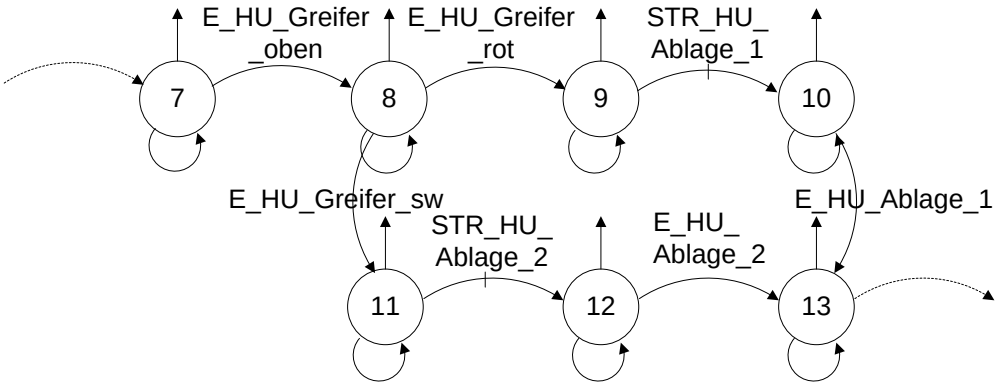


Abbildung 6.4.: Ausschnitt aus der Spezifikation $K7$

Zustand	Ereignisse	Zustand	Ereignisse
7	A_HU_Ablage A_HU_Entnahme	11	A_HU_Ablage A_HU_Entnahme
8	A_HU_Ablage A_HU_Entnahme	13	A_HU_Ablage A_HU_Entnahme
9	A_HU_Ablage A_HU_Entnahme	12	A_HU_Ablage
10	A_HU_Ablage		

Tabelle 6.1.: Ereignisse der Schleifen aus Abbildung 6.4

6.2.4. Spezifikation K_6 und K_8 Transporteinrichtung

Die Steuerungsaufgabe der Transporteinrichtung kann in zwei Spezifikationen aufgeteilt werden. Die Synchronisierung des Schwenkarms, der Greifvorrichtung des Handling Portals und der Transporteinrichtung beim Be- und Entladen wird von der Spezifikation K_6 übernommen. Sie stellt sicher, dass der Schwenkarm erst in die Warteposition fährt. Danach rückt der Transportschlitten weiter bzw. fährt zum Handling Portal. Dort angekommen stellt die Spezifikation sicher, dass das Handling Portal das Werkstück entnommen hat, bevor der Transportschlitten weiterrückt bzw. wieder zur Beladungsstelle fährt. Die Spezifikation wird durch den in Abbildung 6.5 dargestellten Generator repräsentiert.

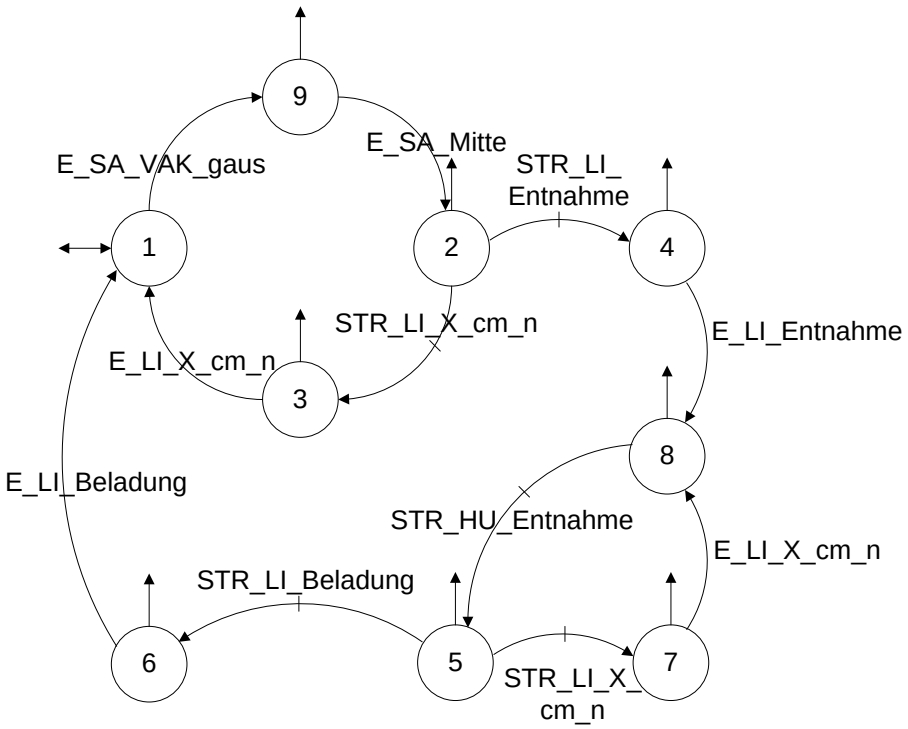


Abbildung 6.5.: Spezifikation $K6$ als Generator

Die zweite Spezifikation K_8 stellt sicher, dass der Transportschlitten mit fünf Werkstücken beladen wird. Sie sorgt dafür, dass der Transportschlitten zunächst vier Mal weiterrückt, um die Werkstücke auf dem Schlitten zu platzieren. Anschließend wird veranlasst, dass der Schlitten mit den Werkstücken zum Handling Portal gefahren wird. Dort angekommen wird der Schlitten wieder vier Mal vorgerückt, um die Werkstücke zu entnehmen. Ist der Vorgang abgeschlossen, so wird veranlasst, dass der Schlitten wieder zur Beladung gefahren wird. Die Steuerung der Linearachse wird von einer Schrittkette übernommen, die die jeweiligen steuerbaren Ereignisse zum Starten benötigt.

6.2.5. Spezifikation K_{10} Robotereinheit

Der Roboter soll, je nach Werkstückfarbe und Qualität, sechs verschiedene Sequenzen abfahren. Zu jeder Sequenz gehört das Transportieren des Werkstücks von der Rutsche zur RFID Station. Dort wird entschieden, an welcher Stelle das Werkstück abgelegt werden soll. Der Roboter soll von hier, je nach Ereignis der RFID Station, die Position „akzeptabel“, „Ausschuss“ oder „reparabel“ anfahren. Abbildung 6.6 zeigt einen Teilausschnitt der Spezifikation, der ein Werkstück von der Rutsche 1 (rot) entnehmen lässt. Die Spezifikation für Rutsche 2

(schwarz) ist identisch. Es wird lediglich die Lichtschranke 2 (anstatt Lichtschranke 1) abgefragt und die Rutsche 2 angefahren. Nach der Detektion eines Werkstücks auf der Rutsche 1 wird das steuerbare Ereignis erlaubt, dass den Roboter zur Rutsche 1 fahren lässt und das Werkstück zur RFID Station transportiert. Die RFID Station entscheidet, welche Qualität das Werkstück besitzt. Je nach Qualität wird die entsprechende Position angefahren und anschließend die Ruheposition eingenommen. Je nach Priorität der Abfragen wird im Anschluss ein Werkstück von der Rutsche 2 oder wieder ein Werkstück der Rutsche 1 entnommen. Nach dem Entnehmen eines Werkstücks von Rutsche 1 wird erst die Lichtschranke der Rutsche 2 abgefragt, um beide Rutschen abwechselnd zu entladen. Sind beide Rutschen leer, wird der Anfangszustand eingenommen.

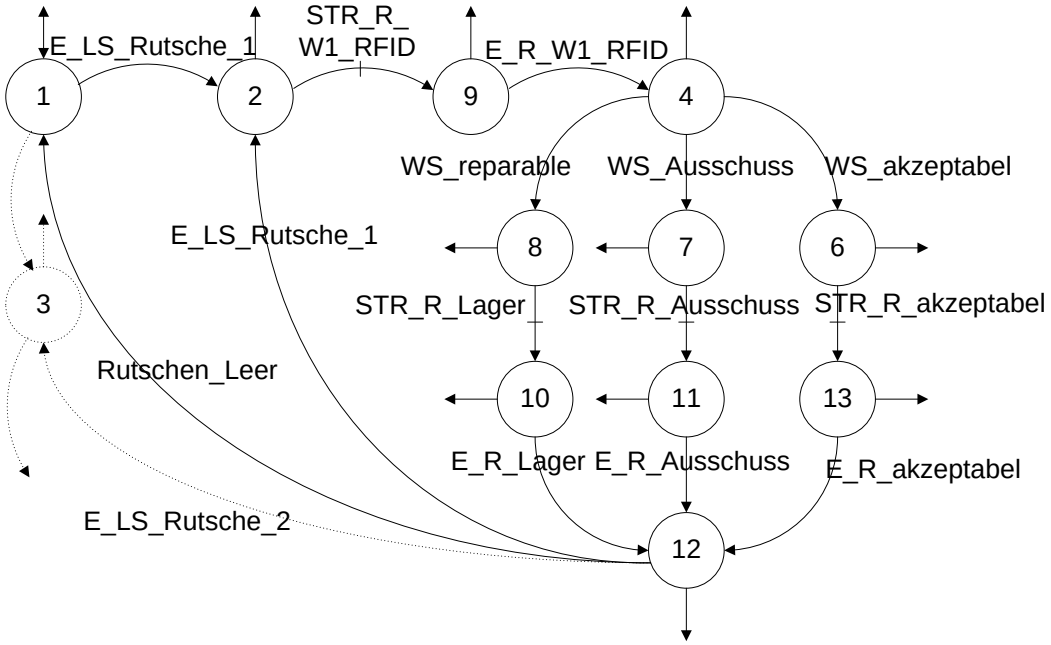


Abbildung 6.6.: Ausschnitt aus Spezifikation $K10$

6.3. Puffermanagement K_1, K_{11} und K_{12}

Die Puffer K_{11} und K_{12} der beiden Rutschen können als Instanz der in Abbildung 6.7 dargestellten generischen Spezifikation abgebildet werden. Die Puffer werden durch die Greifvorrichtung des Handling Portals gefüllt, so dass das Eingangsereignis „STR_HU_Ablage_i“ mit $i = 1, 2$ ist. Entleert wird der Puffer durch den Roboter, der mit dem Ereignis „STR_R_Wi_RFID“ mit $i = 1, 2$. Ist der Puffer voll, verbietet er die weitere Beladung durch

das Handling Portal, bis der Roboter ein Werkstück von der Rutsche entnommen hat. Jeder Puffer hat eine Kapazität von sechs Werkstücken, diese Anzahl darf nicht überschritten werden. Es wird davon ausgegangen, dass zu Beginn die Rutschen leer sind.

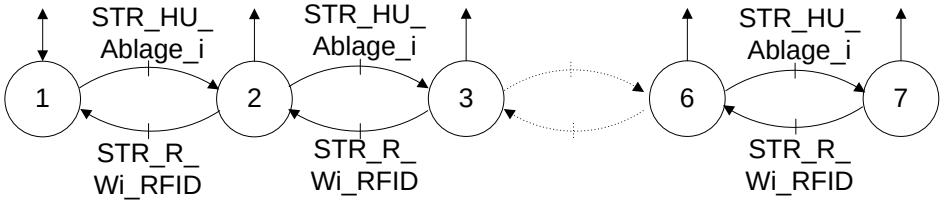


Abbildung 6.7.: Spezifikation K_{1i} als generisches Generatormodell, $i = 1, 2$

Die Warteposition vor dem Lager, in der das Werkstück liegt, nachdem es ausgeschoben wurde und auf den Schwenkarm wartet, kann ebenso als Puffer modelliert werden. Er besitzt die Kapazität 1 und das Eingangsereignis des Puffers ist „STR_LA_AS_vor“, das Ausgangsereignis des Puffers ist „STR_SA_VAK_ein“. Die Abbildung 6.8 zeigt die Pufferspezifikation K_1.

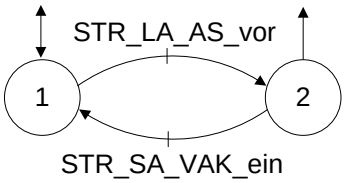


Abbildung 6.8.: Spezifikation K_1 als Generatormodell

6.4. DES Supervisor

Die zuvor entwickelten Spezifikationen müssen auf Steuerbarkeit bezüglich der Strecke überprüft werden. So wird geprüft, ob jeweils eine Steuerung existiert, die das gewünschte Verhalten nachbildet. Die Steuerbarkeit einer Spezifikation K bezüglich eines Generators G kann mit DESTool mit der Operation IsControllable(G,K) geprüft werden. Das Ergebnis der Prüfung ist eine boolesche Variable mit „true“ für steuerbar und „false“ für nicht steuerbar.

Die Spezifikationen berücksichtigen noch nicht, dass alle nicht steuerbaren Ereignisse erlaubt werden müssen. Sie werden dem Ereignisalphabet Σ durch die inverse Projektion hinzugefügt. Dies hat zur Folge, dass alle nicht steuerbaren Ereignisse, die nicht in Σ_{k_i} enthalten

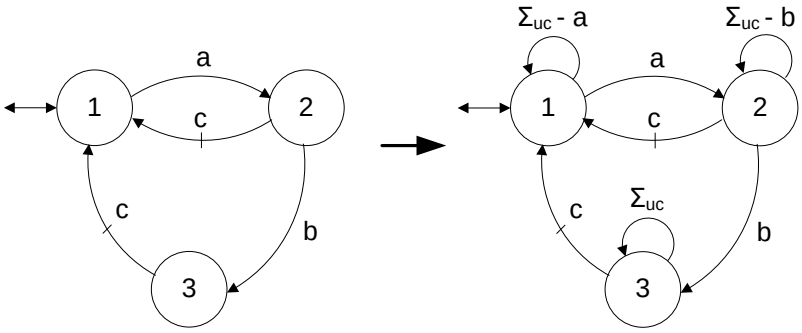


Abbildung 6.9.: Beispiel zum Hinzufügen von nicht steuerbaren Ereignissen

waren, als Selfloops an den Zuständen des Generators erscheinen. Abbildung 6.9 verdeutlicht das Vorgehen anhand eines konstruierten Beispiels. Die Operation SelfLoop(G, Σ_1) fügt jedem Zustand die Ereignisse aus Σ_1 als Selfloop zu. Die Untersuchung auf Steuerbarkeit der Spezifikationen aus Abschnitt 6.2 bezüglich der Strecke G_Gesamt ergaben folgende Ergebnisse:

IsControllable(K_1, G_Gesamt) ✓
IsControllable(K_2, G_Gesamt) ✓
IsControllable(K_3, G_Gesamt) ✓
IsControllable(K_4, G_Gesamt) ✓
IsControllable(K_5, G_Gesamt) ✓
IsControllable(K_6, G_Gesamt) ✓
IsControllable(K_7, G_Gesamt) ✓
IsControllable(K_8, G_Gesamt) ✓
IsControllable(K_9, G_Gesamt) ✓
IsControllable(K_{10}, G_Gesamt) ✓
IsControllable(K_{11}, G_Gesamt) ✓
IsControllable(K_{12}, G_Gesamt) ✓

Es muss demnach keine supremale steuerbare Teilsprache $K_i^{\uparrow}$, für $i = 1, \ldots 12$ ermittelt werden. Die Prüfung der Spezifikationen auf Präfix-Abgeschlossenheit erfolgt durch die DESTool-Operation IsPrefixClosed(G). Sie ergibt, dass alle Spezifikationen präfixgeschlossen sind. Die Forderung nach einer $L_m(G)$-Abgeschlossenheit kann demnach erfüllt werden. Die Spezifikationen können direkt als Supervisor eingesetzt werden.

Das gesteuerte Verhalten kann durch

$$G(S/G) = G\_Gesamt \times S \tag{6.1}$$

nachgebildet werden. Der Gesamtsupervisor S kann durch die Komposition der einzelnen Supervisor ermittelt werden. Dafür werden, ähnlich wie in Abbildung 6.9, durch die inverse Projektion alle steuerbaren Ereignisse, die nicht in Σ_{S_i} enthalten sind, hinzugefügt. S ergibt sich zu

$$S = S_1 \times S_2 \times S_3 \times S_4 \times S_5 \times S_6 \times S_7 \times S_8 \times S_9 \times S_{10} \times S_{11} \times S_{12}. \tag{6.2}$$

Die Prüfung auf Nichtblockieren kann in DESTool durch die Anweisung isNonblocking() erfolgen. Dies ist Aufgrund der Größe des Supervisors nicht möglich.

Im Shuffle-Fall („worst-case“) sind für den Supervisor 2×10^{10} Zustände zu erwarten. Diese Anzahl an Zuständen wäre sehr wahrscheinlich durch DESTool verwertbar. Neben der Anzahl an Zuständen, ist zusätzlich auch die Anzahl der Transitionen verantwortlich für eine hohe Modellkomplexität. DESTool erfordert eine Kapazität von etwa 100 Byte pro Transition. Bei einem 32 Bit Adressraum ergeben sich damit maximal etwa 40×10^6 Transitionen. Zum einen würde das bedeuten, dass im schlechtesten Fall nicht jeder Zustand eine Transition bekommen kann. Zum anderen können möglicherweise die inversen Projektionen der Ereignisse nicht berücksichtigt werden, sollte nicht der schlechteste Fall eintreten.

TCT bietet die Möglichkeit die Konfliktfreiheit der einzelnen Supervisor zu untersuchen [48]. Zwei Generatoren sind konfliktfrei, wenn sich jeder String des komponierten Systems zu einem String in $L_m(G)$ erweitern lässt [28]. So ist die Forderung nach Konfliktfreiheit identisch mit der Forderung nach Nichtblockieren.

Die Prüfung erfolgt in TCT mit dem Algorithmus NONCONFLICT(G1, G2). Die Generatoren $G1$ und $G2$ müssen beide über das gleiche Ereignisalphabet Σ verfügen. Sind die Ereignisalphabete unterschiedlich müssen sie gegebenenfalls mit der inversen Projektion erweitert werden. Bezogen auf die Supervisor und die Strecke müssen die Ereignisalphabete der Supervisor zusätzlich durch die fehlenden, nicht steuerbaren Ereignisse erweitert werden. Die Überprüfung auf Konfliktfreiheit der Supervisor wird paarweise durchgeführt. Die Ergebnisse sind in Tabelle 6.2 aufgelistet.

	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12
K1	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
K2		-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
K3			-	✓	✓	✓	✓	✓	✓	✓	✓	✓
K4				-	✓	✓	✓	✓	✓	✓	✓	✓
K5					-	✓	✓	✓	✓	✓	✓	✓
K6						-	✓	✓	✓	✓	✓	✓
K7							-	✓	✓	✓	✓	✓
K8								-	✓	✓	✓	✓
K9									-	✓	✓	✓
K10										-	✓	✓
K11											-	✓
K12												-

Tabelle 6.2.: Überprüfung der Konfliktfreiheit

7. Codegenerator DES2IEC

Die im Kapitel 6 erstellten Supervisor müssen auf einer SPS implementiert werden, um ihre Aufgabe erfüllen zu können. Das manuelle Programmieren der Hardware erzeugt Fehlerquellen, die dem Syntheseprozess entgegenwirken. Desweiteren benötigen reale Anlagen in der Regel größere Supervisor, die das manuelle Programmieren mühsam und zeitaufwendig machen. Im Folgenden wird ein Tool entwickelt, das aus den in DESTool erstellten Supervisor den AWL Code für eine S7 Steuerung erzeugt.

7.1. Anforderungen an den Codegenerator

Nachfolgend werden die Anforderungen an den Codegenerator aufgeführt und erläutert. Der Codegenerator soll das von DESTool V. 0.59 erzeugte Projekt einlesen und daraus die (zur Codeerzeugung) relevanten Daten extrahieren. Dazu zählen:

- Zustände (mit dazugehörigem Namen),
- Alphabet (Ereignisnamen),
- Transitionen,
- Initialisierungszustände.

Die Transitionen müssen entsprechend aufgearbeitet werden, so dass eine Adjazenzmatrix erstellt werden kann. Der Code soll so erzeugt werden, dass am Ende die Bedingung für einen modularen Supervisor erfüllt werden kann. Dafür müssen die Steuereingriffe des Supervisors mit den Steuereingriffen der anderen Supervisor konjungiert werden.

Die Zustände sollen mit einem symbolischen Namen benannt werden, um den erzeugten Code besser lesbarer zu machen. Die Form wird wie folgt definiert:

Supervisorname_Zustandsname.

Auch die Events sollen im Code einen symbolischen Namen erhalten. Das Setzen der Ausgänge soll nach dem Ansatz von Uzam [45] erfolgen.

7.2. Analyse des DESTool-Dateiformats

Es wird im Folgenden das Dateiformat des Projektfiles *.pro von DESTool Version 0.59[1] analysiert. Das Projektfile ist in einer Sprache erstellt, deren Syntax der der Auszeichnungssprache HTML ähnlich ist (Beispiel K_3):

```
<Generator>
K3
%
%  Statistics for K3
%
%  States:         6
%  Init/Marked:    1/0
%  Events:         8
%  Transitions:    8
%  StateSymbols:   0
%  Attrib. E/S/T: 2/0/0
%
<Alphabet>
STR_LA_AS_vor  +C+ E_LA_AS_zu ...
STR_SA_Lin    +C+  E_LA_KTeil ...
</Alphabet>
<States>
1 2 3 4 5 6
</States>
<TransRel>
1              STR_LA_AS_vor  2
1              E_LA_KTeil    6
2              E_LA_AS_vor   3
2              A_LA_AS_vor   2
3              E_LA_AS_zu    4
4              STR_SA_Lin    5
5              E_SA_Lin      1
6              E_LA_Teil     1
</TransRel>
<InitStates>
1
</InitStates>
<MarkedStates>
1 2 3 4 5 6
</MarkedStates>
</Generator>
```

[1] Dateiformate anderer Versionen von DESTool können von diesem hier analysierten Dateiformat abweichen.

Jede erstellte Variable (System, Alphabet, Boolean, etc.) erhält einen Eintrag, beginnend mit dem Namen und der Art der Variable. Ein Generator beginnt mit dem Schlüsselwort <Generator>. Darauf folgt der Name. In diesem Beispiel ist das K3. Nach der Statistik, die unter anderem die Anzahl von Zuständen und Ereignissen enthält, folgt mit dem Schlüsselwort <Alphabet> die Auflistung der Ereignisse. Diese endet mit </Alphabet>. Steuerbare Ereignisse werden mit einem $+C+$ markiert. Anschließend folgt die Liste der Zustände, die mit dem Schlüsselwort <States> beginnt und mit </States> endet. Mit dem Schlüsselwort <TransRel> beginnt die Auflistung der Transitionen. Die Transitionen werden in der Form

1 STR_LA_AS_vor 2

dargestellt. Dabei entspricht die 1 dem Anfangszustand. STR_LA_AS_vor stellt das Ereignis dar und die 2 entspricht dem Zielzustand. Der Initialisierungszustand wird in die Umgebung von <InitStates> und </InitStates> eingetragen.

7.3. Implementierung

Die Implementierung des Codegenerators erfolgt durch die Programmierumgebung Microsoft Visual C++ 6.0 in der Programmiersprache C. Die Aufgaben des Codegenerators teilen sich in fünf Unterfunktionen auf, die in folgender Reihenfolge aufgerufen werden.

int dataInput(void)
Die Unterfunktion dataInput() dient als Benutzerinterface und liest die anwenderspezifischen Daten, wie den Namen der DESTool Datei und die Größe der Zustands- und Ereignismenge ein. Zur Identifikation des gewünschten Generators im DESTool-Projekt muss der Name des Supervisors angegeben werden. Es wird ein sogenannter S7 Name definiert, um die Kommentare, die symbolischen Namen der Zustände und die Steuereingriffe der einzelnen Supervisor erzeugen zu können.

void alloc(void)
Die Unterfunktion alloc() übernimmt die dynamische Speicherallokierung des Programms. Alle verwendeten Arrays werden mit Hilfe der Speicherallokierung verwaltet, um für alle möglichen Supervisor ausreichend Arbeitsspeicher zur Verfügung zu stellen. Um einen Initialisierungszustand zu erreichen, beschreibt die Unterfunktion alle Arrays mit einem definierten Anfangswert.

void scanFile(void)
Diese Unterfunktion durchsucht das DESTool-Projekt nach dem gewünschten Generator und

extrahiert die relevanten Daten. Dafür wird zunächst das DESTool-Projekt geöffnet. Die Datei wird Zeile für Zeile eingelesen und ausgewertet. Durch das Erkennen des Supervisornamens wird der Speicherbereich der Informationen über den Generator gefunden. Im folgenden Funktionsablauf werden die Schlüsselwörter <Alphabet>, <States> und <TransRel> gesucht. Sie kennzeichnen jeweils die Umgebungen für die Ereignis-, Zustands- und Transitionsmengen. Die Mengen werden in die vorher allokierten Arrays sequentiell kopiert und zur weiteren Identifizierung mit Nummern versehen. Der Programmablaufplan in Abbildung 7.1 zeigt schematisch die Abarbeitung der Unterfunktion.

void createAdja(void)
Zur Erstellung der Adjazenzmatrix dient die Unterfunktion createAdja(). Dazu wird das in der Funktion scanFile() beschriebene zwei Dimensionale Array „transitionen“ verwendet. Es enthält i Einträge mit maximal 100 Zeichen. Dabei enthält jeder Eintrag die Komponenten Anfangszustand, Ereignis und Zielzustand. Jede Komponente wird mit einen (oder mehreren) Leerzeichen von den anderen getrennt. Jeder Eintrag i von „transitionen[i][x]“ wird nacheinander elementweise mit x durchsucht. Aus dem Aufbau der Transitionen lassen sich Situationen herleiten, wodurch auf die einzelnen Komponenten geschlossen werden kann. Es ergeben sich die folgenden sechs Fälle:

1. Der Zähler x befindet sich am Anfang des Eintrages (Fall 1):
   - das Element „transitionen[i][x]“ ist nicht leer und enthält kein Leerzeichen und
   - der Anfangszustand wurde noch nicht gefunden,

   $\Rightarrow$ dann ist der Inhalt von „transitionen[i][x]“ der Anfangszustand der Transition.
2. Der Zähler x befindet sich am ersten Leerzeichen zwischen Anfangszustand und Ereignis (Fall 2):
   - das Element „transitionen[i][x]“ enthält ein Leerzeichen und
   - die Anfangszustandssuche wurde noch nicht beendet,

   $\Rightarrow$ dann ist die Anfangszustandssuche abgeschlossen. Es wird registriert, dass ein Leerzeichen gefunden wurde.
3. Der Zähler x befindet sich am ersten Zeichen des Ereignisses (Fall 3):
   - das Element „transitionen[i][x]“ ist nicht leer und enthält kein Leerzeichen,
   - die Suche nach dem Anfangszustand wurde beendet und
   - die Suche nach dem Ereignis wurde noch nicht beendet,

   $\Rightarrow$ dann ist das Ereignis der Transition gleich dem Element aus „transitionen[i][x]“.

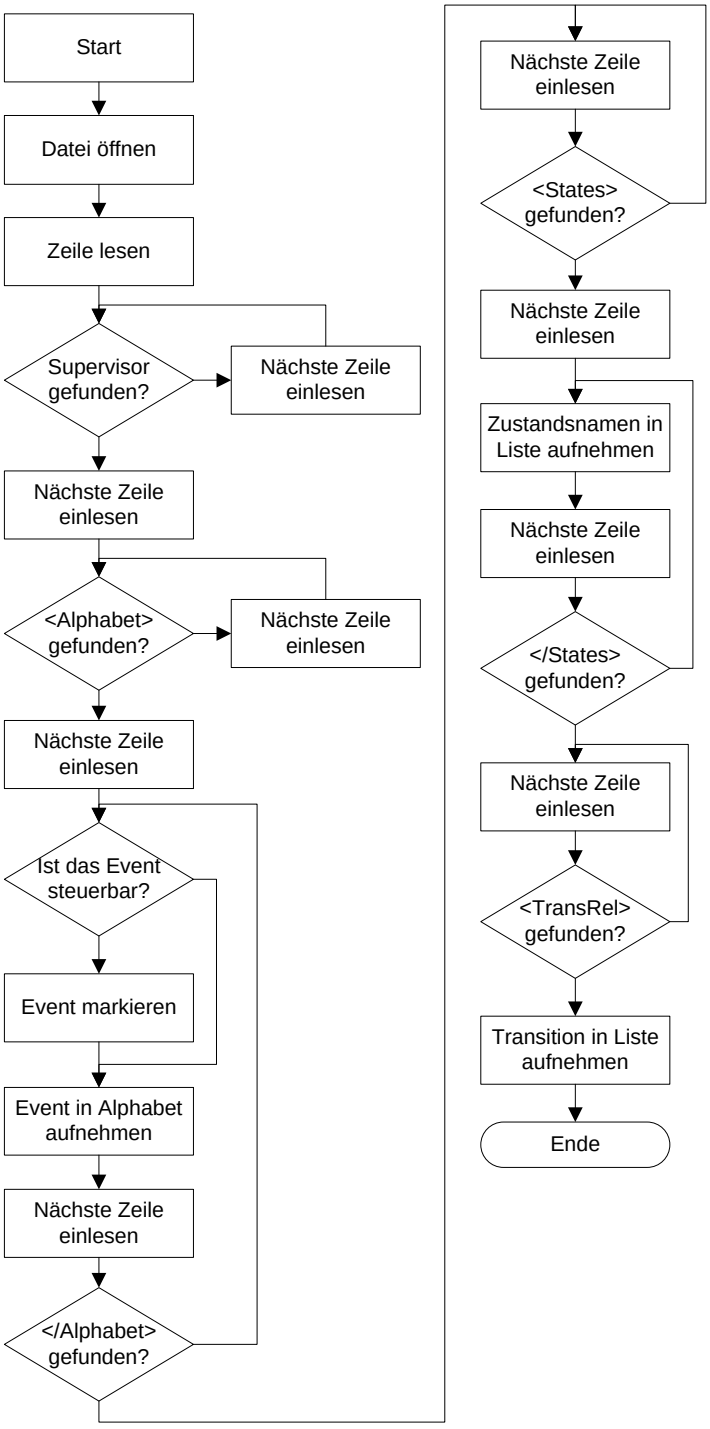


Abbildung 7.1.: Programmablaufplan Unterfunktion scanFile()

4. Der Zähler x befindet sich am ersten Leerzeichen zwischen Ereignis und Endzustand (Fall 4):
   - das Element „transitionen[i][x]“ enthält ein Leerzeichen,
   - die Anfangszustandssuche wurde beendet und
   - die Suche nach dem Ereignis wurde noch nicht beendet,

   $\Rightarrow$ dann ist die Ereignissuche beendet und das Leerzeichen wird registriert.
5. Der Zähler x befindet sich am ersten Zeichen des Zielzustandes (Fall 5):
   - das Element „transitionen[i][x]“ ist nicht leer und enthält kein Leerzeichen,
   - die Suche nach dem Ereignis wurde abgeschlossen und
   - der Zielzustand wurde noch nicht gefunden,

   $\Rightarrow$ dann ist das Element „transitionen[i][x]“ gleich dem Zielzustand.
6. Der Zähler x befindet sich hinter dem Zielzustand (Fall 6):
   - das Element „transitionen[i][x]“ enthält ein Leerzeichen (oder leer),
   - die Suche nach dem Ereignis wurde abgeschlossen und
   - die Suche nach dem Zielzustand wurde noch nicht abgeschlossen,

   $\Rightarrow$ dann ist die Suche nach dem Zielzustand abgeschlossen und es kann ein Eintrag in die Adjazenzmatrix erfolgen.

Durch das Abfragen dieser Fälle werden die einzelnen Komponenten der Transition getrennt. Dafür werden die Laufvariablen durch zwei geschachtelte For-Schleifen durchgezählt. Die Variable i repräsentiert dabei die Transitionsnummer und x die einzelnen Elemente der Transition. Die Abfrage vom Fall 1 erfolgt zum Beispiel durch den folgenden Codeausschnitt.

```
//Fall 1
if((transitionen[i][x]!='␣') && findFrom==0)
{
  fromState[y]=transitionen[i][x];
  space=0;
  y++;
}
```

Dadurch wird das erste Zeichen des Anfangszustandes der Transition extrahiert. Die Variable findFrom signalisiert, ob die Suche nach dem Anfangszustand schon beendet wurde. Sie dient als Verriegelungsvariable und hilft die einzelnen Fälle zu identifizieren. Wurde diese Variable gesetzt, erkennt die Software, dass das nächste Symbol, das kein Leerzeichen ist, das erste Zeichen des Ereignisnamens ist. So wird die Abfrage von Fall 3 durch die folgende Abfrage realisiert.

```
//Fall 3
if((transitionen[i][x]!='␣') && findFrom==1 && findTrans==0)
{
  trans[y]=transitionen[i][x];
  space=0;
  y++;
}
```

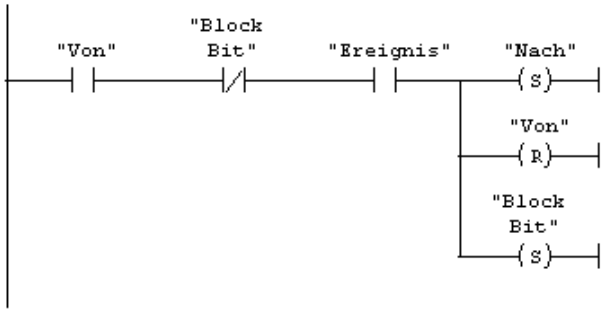
Wurden Anfangszustand, Ereignis und Zielzustand der Transition gefunden, so kann daraus die Adjazenzmatrix erstellt werden. Die Adjazenzmatrix wird durch das dreidimensionale Array „adjazenzMtx“ repräsentiert. Die dritte Dimension ist notwendig, um mehr als eine Transition vom Zustand x_1 zu x_2 oder mehrere Selfloops zu ermöglichen. Die Konvention von der Variable adjazenzMtx ist

adjazenzMtx [von Zustand] [nach Zustand] [Anzahl der Transitionen].

Der Wert im Feld des Arrays entspricht der Ereignisnummer. Initialisiert wurde das Array mit -1, so dass eine freie Zelle durch diesen Wert erkennbar ist. Eine Transition wird durch die Ereignisnummer im entsprechenden freien Element des Arrays mit folgendem Code abgespeichert.

```
if(fromNum>=0 && toNum>=0)
{
  for(ansTransState=0;ansTransState<maxStates;ansTransState++)
  {
    if(adjazenzMtx[fromNum][toNum][ansTransState]==-1)
    {
      adjazenzMtx[fromNum][toNum][ansTransState]=eventNum;
      anzEventAdja[fromNum][toNum]++;
      break;
    }
  }
}
```

void createCode(void)
Die Unterfunktion createCode() bildet den Kern des Codegenerators. Sie erzeugt den STEP7




```
U    "Von"
UN   "Block Bit"
U    "Ereignis"
S    "Nach"
R    "Von"
S    "Block Bit"
```

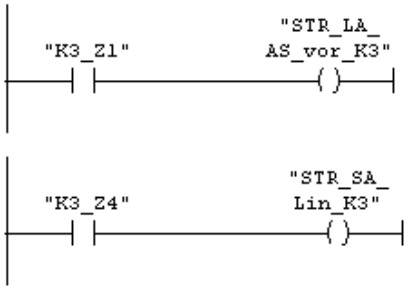
a) Kontaktplan (KOP) b) Anweisungsliste (AWL)

Abbildung 7.2.: Schema der Codegenerierung

AWL Code, der direkt in den STEP7 AWL Editor eingefügt werden kann. Zunächst müssen die Aktivitäten nach [45] identifiziert werden. Dies muss manuell durch den Benutzer erfolgen. Hierzu werden alle nicht steuerbaren Ereignisse aufgelistet. Der Benutzer muss durch Tasteneingabe die Ereignisse auswählen, die einen Ausgang repräsentieren. Durch die Eingabe von -1 wird die Auswahl beendet. Wurde eine Aktivität *i* gewählt, so wird dieses Ereignis in dem Array activity[i] mit einer 1 markiert.

Die Codegenerierung wird in vier Abschnitte eingeteilt. Zuerst werden die Abfragen der nicht steuerbaren Ereignisse nach [14] erstellt. Durch eine dreifach geschachtelte For-Schleife wird die Adjazenzmatrix nach nicht steuerbaren Ereignissen durchsucht. Selfloops und Ereignisse, die Aktivitäten darstellen, werden dabei ignoriert. Im zweiten Abschnitt werden die Abfragen der steuerbaren Ereignisse erstellt. Diese Abfragen werden ebenfalls -wie die nicht steuerbaren Ereignisse- erstellt. Es wird lediglich die Adjazenzmatrix nach steuerbaren Ereignissen durchsucht. Wurde in der Adjazenzmatrix ein entsprechendes Element gefunden, so werden durch die Zustandsnummern und der Transitionsnummer die symbolischen Namen ermittelt und der zur Transition passende Code in die Datei Step7AWLCode.txt im Programmverzeichnis geschrieben. Der Code besteht immer aus dem in Abbildung 7.2 dargestellten Schema. Das Beschreiben der Datei erfolgt durch den folgenden Codeausschnitt.

```
fprintf(pout,"//Z%s\n",fromState);
fprintf(pout,"U␣%s_Z%s\n",s7Name,fromState);
fprintf(pout,"UN␣block_%s\n",s7Name);//block_s7Name
fprintf(pout,"U␣");
for(i=0;i<LTRANSNAME;i++)
  {
  if(alphabet[adjazenzMtx[von][nach][anzElement]][i]!=0 &&
  alphabet[adjazenzMtx[von][nach][anzElement]][i]!=-52)
    fprintf(pout,"%c",alphabet[adjazenzMtx[von][nach][anzElement]][i]);
  }
```



a) Kontaktplan (KOP)

```
U     "K3_Z1"
=     "STR_LA_AS_vor_K3"
U     "K3_Z4"
=     "STR_SA_Lin_K3"
```

a) Anweisungsliste (AWL)

Abbildung 7.3.: Setzen der steuerbaren Ereignisse

```
fprintf(pout,"\n");
fprintf(pout,"S␣%s_Z%s\n",s7Name,toState);
fprintf(pout,"R␣%s_Z%s\n",s7Name,fromState);
fprintf(pout,"S␣block_%s\n",s7Name);
```

Durch die For-Schleife werden die einzelnen Buchstaben des Ereignisses aus dem Array „alphabet“ gelesen und in die Datei geschrieben.

Im dritten Abschnitt werden die steuerbaren Ereignisse gesetzt, die durch die SPS forciert werden. Im modularen Ansatz werden die Steuereingriffe jedes einzelnen Supervisors durch eine Konjunktion verknüpft. Nicht steuerbare Ereignisse müssen vom Supervisor immer erlaubt werden. Die Steuereingriffe verschiedener modularer Supervisor können sich somit nur in der Menge der nicht steuerbaren Ereignisse unterscheiden. Es ist somit dringend notwendig, alle steuerbaren Ereignisse der Supervisor aus der Menge der Steuereingriffe mit einer Konjunktion zu verknüpfen. Es ergibt sich dadurch eine Notwendigkeit für temporäre steuerbare Ereignisse, die vom Supervisor gesetzt werden und den steuerbaren Teil des Steuereingriffs repräsentieren. Sie können nicht alleine über das Setzten der eigentlichen Ereignisse entscheiden. Die Notation der Ereignisse ist wie folgt:

Ereignisname_S7Name.

Die Variable S7Name wurde vom Benutzer vergeben und in der Funktion dataInput() eingelesen. Die Konjunktion der temporären steuerbaren Ereignisse entscheidet über das Setzen der eigentlichen Ereignisse (siehe Abbildung 7.4). In diesem Beispiel kann das Ereignis a nur gesetzt werden, wenn in jedem Steuereingriff $S_x(s)$ für $x = 1 \ldots 3$ von jedem Supervisor das Ereignis a_Sx erlaubt wird.

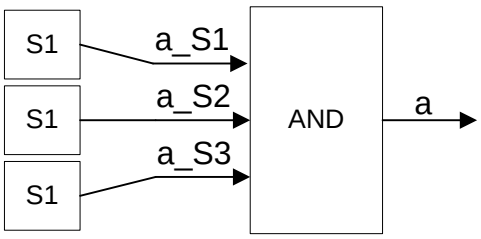


Abbildung 7.4.: Konjunktion von temporären Ereignissen

Ein steuerbares Ereignis wird gesetzt, wenn der Zustand aktiv ist, von dem die Kante mit dem Ereignis abgeht. In der Spezifikation K3 (siehe Abschnitt 6.2.1) muss demnach in Zustand 1 das Ereignis „STR_LA_AS_vor_K3“ und im Zustand 4 „STR_SA_Lin3“ gesetzt werden. Es ergibt sich die in Abbildung 7.3 dargestellte Anweisung.

Werden Ereignisse in mehr als einem Zustand gesetzt, so müssen die Merker der Zustände durch eine Disjunktion verknüpft werden, bevor das temporäre Ereignis gesetzt wird. Um dies im Codegenerator umzusetzen, muss bekannt sein, ob und wie oft dieser Fall im Supervisor vorkommt. Dafür wird gezählt, wie oft ein Ereignis im Supervisor erzeugt wird. Durch eine dreifache For-Schleife wird die Adjazenzmatrix durchsucht und das Vorkommen der steuerbaren Ereignisse im Array „cTable[Ereignisnummer]“ gezählt.

```
for(von=0; von<maxStates;von++)
{
  for(nach=0; nach<maxStates;nach++)
  {
    for(anzElement=0;anzElement<anzEventAdja[von][nach];anzElement++)
    {
      if(adjazenzMtx[von][nach][anzElement]!=-1)
      {
        if(ctrblEvent[adjazenzMtx[von][nach][anzElement]]!=0)
          cTable[adjazenzMtx[von][nach][anzElement]]++;
      }
    }
  }
}
```

Zum Erzeugen der Codezeilen, die das Setzen der steuerbaren Ereignisse bewirken, wird die Adjazenzmatrix nach steuerbaren Ereignissen durchsucht. Da ein Supervisor Ereignisse erlauben kann, ohne dass ein Zustandswechsel stattfinden muss, dürfen die Selfloops hier *nicht* ignoriert werden. Wurde ein zutreffendes Ereignis gefunden, so wird zunächst geprüft, ob dieses Ereignis nur einmal im Supervisor vorkommt. Ist dies der Fall, so wird das Ereignis mit Hilfe einer UND-Verknüpfung in die Text Datei geschrieben.

```
if(cTable[adjazenzMtx[von][nach][anzElement]]<=1)
{
...
  fprintf(pout,"U␣%s_Z%s", s7Name, fromState);
  fprintf(pout,"\n");
  fprintf(pout,"=");
  for(i=0;i<LTRANSNAME;i++)
  {
    if(alphabet[adjazenzMtx[von][nach][anzElement]][i]!=0)
      fprintf(pout,"%c",alphabet[adjazenzMtx[von][nach][anzElement]][i]);
  }
  fprintf(pout,"_%s",s7Name);
  fprintf(pout,"\n");
  fprintf(pout,"\n");
}
```

Wird das Ereignis mehr als einmal verwendet, so müssen die Merker der Zustände durch eine Disjunktion verknüpft werden. Dabei wird das erste Ereignis mit einer Konjunktion versehen und die darauf folgenden Zustände mit einer Disjunktion. Dafür zählt der Zähler (cCounter), wie oft die Disjunktion geschrieben wurde. Ist der aktuelle Zustand der letzte Zustand, dessen Merker durch eine Disjunktion verknüpft wird, wird das Ereignis in der nächsten Zeile mit einem „Gleich“ in die Textdatei geschrieben.

```
//Letzer Zustand
if(cCounter[z]==cTable[adjazenzMtx[von][nach][anzElement]])
{
  fprintf(pout,"O␣%s_Z%s", s7Name, fromState);
  fprintf(pout,"\n");
  fprintf(pout,"=");
  for(i=0;i<LTRANSNAME;i++)
  {
  if(alphabet[adjazenzMtx[von][nach][anzElement]][i]!=0
  && alphabet[adjazenzMtx[von][nach][anzElement]][i]!=-52)
    fprintf(pout,"%c",alphabet[adjazenzMtx[von][nach][anzElement]][i]);
  }
  fprintf(pout,"_%s",s7Name);
  fprintf(pout,"\n");
  fprintf(pout,"\n");
}
//Ereignis kommt mehr als einmal vor, der erste
//Zustand wurde schon geschrieben und
//es ist noch nicht der letzte Zustand
if(cCounter[z]!=1 && cCounter[z]!=
cTable[adjazenzMtx[von][nach][anzElement]])
```

```
{
  fprintf(pout,"O %s_Z%s", s7Name, fromState);
  fprintf(pout,"\n");
  cCounter[z]++;
}
//Erster Zustand
if(cCounter[z]==1 && cCounter[z]!=
cTable[adjazenzMtx[von][nach][anzElement]])
{
  fprintf(pout,"U %s_Z%s", s7Name, fromState);
  fprintf(pout,"\n");
  cCounter[z]++;
}
```

Der letzte Abschnitt beinhaltet das Setzten der Ausgänge nach [45]. Auch hier ist es möglich, in mehreren Zuständen die gleiche Aktivität zu setzen. Dafür wird die Anzahl der Zustände ermittelt, die das Ereignis y als Aktivität in Form einer Schleife enthalten. Diese Menge wird in dem Array „actCounter[y]“ gespeichert. Das Array „ucTable“ enthält die Plätze der nicht steuerbaren Ereignisse in der Ereignisliste.

```
for(y=0;y<ucEvent;y++)
{
  if(activity[y]!=-1)
  {
    for(von=0; von<maxStates;von++)
    {
      for(nach=0; nach<maxStates;nach++)
      {
        for(anzElement=0;anzElement<anzEventAdja[von][nach];anzElement++)
        {
          if(adjazenzMtx[von][nach][anzElement]==ucTable[y])
            actCounter[y]++;
        }
      }
    }
  }
}
```

Der Code zum Setzen der Ausgänge wird nach dem gleichen Verfahren erzeugt, wie der zum Setzen der steuerbaren Ereignisse. Die Adjazenzmatrix wird für jede Aktivität durchsucht. Sobald die Ereignisnummer y der Aktivität in der Adjazenzmatrix gefunden wurde, werden die passenden Zustands- und Ereignisnamen herausgefunden. Wurde der erste Aktivitätszustand gefunden, wird der Name des Zustandes mit einer Konjunktion in die Step7AWLCode.txt Datei geschrieben.

```
...
else
{
  ...
  fclose(statesFile);
  fprintf(pout,"U␣%s_Z%s",s7Name, fromState);
  fprintf(pout,"\n");
  count++;
}
```

Beim Durchlaufen der Anweisung wird ein Zähler (count) hochgezählt, um die Anzahl der Zustände zu registrieren. Ist der Zähler gleich dem Wert in actCounter[y], wird das Ereignis der Aktivität mit einem Gleich hinter die Konjunktion gesetzt.

```
if(actCounter[y]==(count))
{
  fprintf(pout,"=");
  for(i=0;i<LTRANSNAME;i++)
  {
    if(alphabet[adjazenzMtx[von][nach][anzElement]][i]!=0)
      fprintf(pout,"%c",
      alphabet[adjazenzMtx[von][nach][anzElement]][i]);
  }
  fprintf(pout,"\n");
}
```

Beinhaltet mehr als ein Zustand die Aktion, werden alle folgenden Zustände mit einer Disjunktion verknüpft. Auch hier wird der Zähler (count) hochgezählt, um beim letzten Zustand das Ereignis zu setzten.

```
if(actCounter[y]>=1 && count>0)
{
  ...
  fprintf(pout,"O␣%s_Z%s",s7Name, fromState);
  fprintf(pout,"\n");
}
count++;
```

Die Codegenerierung ist an dieser Stelle abgeschlossen. Der Inhalt der Datei Step7AWLCode.txt im Programmverzeichnis kann direkt in den STEP7 AWL Editor eingefügt und auf die SPS übertragen werden. Es ist ein manuell erstellter Baustein zur Konjunktion der temporären steuerbaren Ereignisse und dem endgültigen Setzen der steuerbaren Ereignisse nötig.

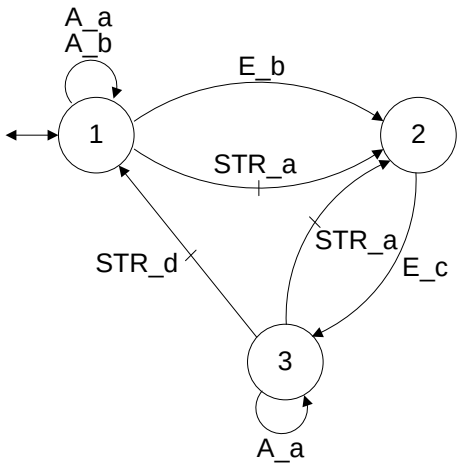


Abbildung 7.5.: Beispielgenerator K_B1 zum Testen von DES2IEC

7.4. Ergebnis

Das Ziel war es, ein Codegenerator zu entwickeln, der aus einem in DESTool modellierten Generator einen Quellcode für eine S7-300 Steuerung erzeugt. Das Verhalten des Generators soll dabei exakt nachgebildet werden. Genannt wird der Codegenerator DES2IEC. Dabei stehen DES für Discrete Event Systems, die 2 steht als Synonym für nach (englisch to, gesprochen wie two). IEC steht für die International Electrotechnical Commission (als Synonym für die IEC 61131-1 [11]) und somit für die Normierung der Programmiersprachen für Speicherprogrammierbare Steuerungen.

DES2IEC liest die DESTool Projektdatei ein und extrahiert die zur Codegenerierung relevanten Informationen, aus denen eine Adjazenzmatrix erzeugt wird. Mit Hilfe der Adjazenzmatrix erfolgt die Codeerzeugung. Die Funktion von DES2IEC wird durch einen Beispielgenerator getestet. Der in Abbildung 7.5 dargestellte Generator enthält alle Sonderfälle, die bei der Codeerzeugung zu beachten sind:

1. Ausgänge werden durch Aktivitäten ersetzt und als Selfloops an die entsprechenden Zuständen gesetzt.
2. Es werden in Zustand 2 zwei Ausgänge gleichzeitig gesetzt.
3. Vom Zustand 1 gehen zwei Kanten ab.
4. Das steuerbare Ereignis STR_a wird zwei Mal im Generator verwendet.
5. Vom Zustand 3 gehen zwei steuerbare Ereignisse ab.

Die Zustandsmenge des Generators ergibt sich zu

$$X = \{1, 2, 3\}\,. \tag{7.1}$$

Das Ereignisalphabet wird beschrieben durch

$$\Sigma = \{A\_a, A\_b, E\_b, E\_c, STR\_a, STR\_d\}\text{ , mit} \tag{7.2}$$

$$\Sigma_c = \{STR\_a, STR\_d\}\text{ und} \tag{7.3}$$

$$\Sigma_{uc} = \Sigma - \Sigma_c. \tag{7.4}$$

Die transponierte Adjazenzmatrix ergibt sich zu

$$A^T = \begin{pmatrix} A\_a, A\_b & E\_b, STR\_a & 0 \\ 0 & 0 & E\_c \\ STR\_d & STR\_a & A\_a \end{pmatrix}. \tag{7.5}$$

DES2IEC hat nach der Ereignislistensuche die Ausgabe 7.6 erzeugt. Zu sehen sind die gefundenen Ereignisse, die mit dem Ereignisalphabet aus (7.2) übereinstimmen.

```
0: E_b - not controllable
1: STR_a - controllable
2: E_c - not controllable
3: STR_d - controllable
4: A_a - not controllable
5: A_b - not controllable
```

Abbildung 7.6.: Screenshot von der Ereignisausgabe aus DES2IEC

Die Ausgabe der transponierten Adjazenzmatrix ist in Abbildung 7.7 dargestellt. Durch die in Abbildung 7.6 abgebildeten Ereignisnummern können die Ereignisse aus der Adjazenzmatrix den jeweiligen Namen zugeordnet werden. Die transponierte Adjazenzmatrix stimmt mit der in Gleichung (7.5) überein.

```
Adjazenzmatrix:
   nach
von
    4;5;    0;1;      -
      -       -       2
      3       1       4
```

Abbildung 7.7.: Screenshot von der Adjazenzmatrixausgabe aus DES2IEC

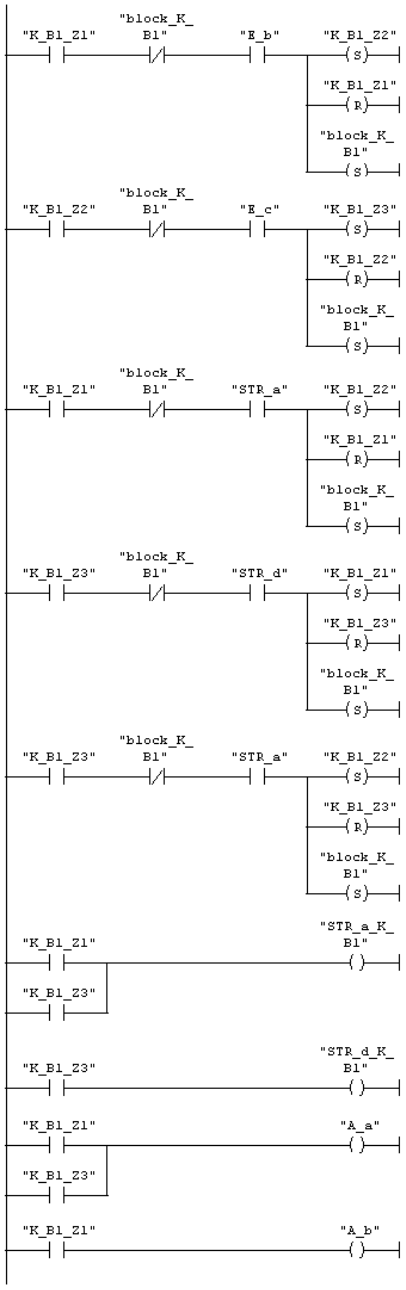
Die Abbildung 7.8 zeigt den von DE2IEC generierten Quellcode für die SPS. Er ist in 7.8a als Anweisungsliste (AWL) dargestellt. In dieser Form wird er in das Textfile Step7AWLCode.txt

geschrieben. Abbildung 7.8b zeigt den Code in der Sprache Kontaktplan (KOP) übersetzt. Die Zustände und Ereignisse sind symbolisch benannt und benötigen einen Eintrag in die Symboltabelle des STEP7 Editors.

Der Quellcode bildet alle fünf Transitionen ab, die zu einem Zustandswechsel führen. Begonnen wird mit den nicht steuerbaren Ereignissen, es folgen die steuerbaren Ereignisse und anschließend werden die temporären steuerbaren Ereignisse gesetzt. Diese werden nach der vorher festgelegten Notation benannt. Das Ereignis „STR_a_K_B1“ wird im Zustand 1 und 3 gesetzt. Die Aktivitäten werden am Ende des Codes gesetzt. Auch dabei wird beachtet, dass die Aktion „A_a“ in Zustand 1 und 2 verwendet wird. Der Quellcode bildet den in Abbildung 7.5 dargestellten Generator exakt nach, so dass das Ziel erreicht wurde.

```
// Code generated by DES2IEC 2.01
//Z1
      U     "K_B1_Z1"
      UN    "block_K_B1"
      U     "E_b"
      S     "K_B1_Z2"
      R     "K_B1_Z1"
      S     "block_K_B1"

//Z2
      U     "K_B1_Z2"
      UN    "block_K_B1"
      U     "E_c"
      S     "K_B1_Z3"
      R     "K_B1_Z2"
      S     "block_K_B1"

//Z1
      U     "K_B1_Z1"
      UN    "block_K_B1"
      U     "STR_a"
      S     "K_B1_Z2"
      R     "K_B1_Z1"
      S     "block_K_B1"

//Z3
      U     "K_B1_Z3"
      UN    "block_K_B1"
      U     "STR_d"
      S     "K_B1_Z1"
      R     "K_B1_Z3"
      S     "block_K_B1"

//Z3
      U     "K_B1_Z3"
      UN    "block_K_B1"
      U     "STR_a"
      S     "K_B1_Z2"
      R     "K_B1_Z3"
      S     "block_K_B1"

//Steuerbare Ereignisse setzen
      U     "K_B1_Z1"
      O     "K_B1_Z3"
      =     "STR_a_K_B1"

      U     "K_B1_Z3"
      =     "STR_d_K_B1"


//Ausgänge Setzen
      U     "K_B1_Z1"
      O     "K_B1_Z3"
      =     "A_a"
      U     "K_B1_Z1"
      =     "A_b"
```

a) AWL Code



a) KOP Code

Abbildung 7.8.: Generierter Code zum Beispiel K_B1

8. Realisierung und Inbetriebnahme

Das folgende Kapitel teilt sich in zwei Teile auf. Im ersten Abschnitt wird die Implementierung der Steuerung vorgestellt. Dabei wird neben der Implementierung der Supervisor auch auf die Implementierung der Schrittketten und der Robotersteuerung eingegangen. Im zweiten Abschnitt werden die Betriebsergebnisse der Steuerung vorgestellt.

8.1. Implementierung der Steuerung

Schrittketten
Zur Modellierung der Strecke wurden untergelagerte Schrittketten vorausgesetzt. Sie fungieren als Schnittstellen zwischen Supervisor und der realen Hardware und realisieren zeitbehaftete Prozesse. Zusätzlich werden sie für deutlich aufwendige Befehlsabläufe verwendet, die z.B. bei der Transporteinheit essenziell sind. Abbildung 8.1 zeigt die implementierte Steuerungsstruktur.

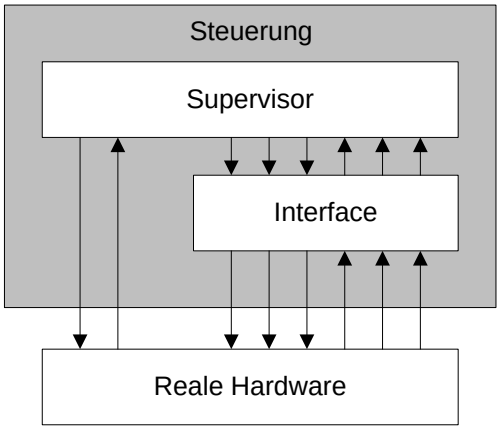


Abbildung 8.1.: Steuerungsstruktur

Der Supervisor kann direkt auf die Ausgänge der SPS und somit auf die reale Hardware zugreifen. Dies wird für den Ausschieber und das Handling Portal genutzt. Bei bestimmten Prozessen wird für den Zugriff auf die Hardware ein Interface benutzt. Dies betrifft z.B. die

Ansteuerung des Schwenkarms und des Saugers. Der Roboter und die Transporteinheit werden ebenfalls über ein Interface angesteuert. Die Rückmeldungen der Hardware werden entweder direkt an den Supervisor geleitet oder über das Interface ausgelesen. So werden die Lichtschranken direkt aus der Hardware im Supervisor verwendet. Die Rückmeldungen vom Roboter werden durch das Interface ausgewertet und erst dann an den Supervisor weitergereicht.

Abbildung 8.2 zeigt die Schrittkette zur Steuerung des Vakuums am Schwenkarmsauger. Durch das steuerbare Ereignis „STR_SA_VAK_ein“ wird die Schrittkette aus dem Supervisor heraus aktiviert. Der Ausgang zum Öffnen des Ventils wird gesetzt, so dass ein Werkstück angesaugt werden kann. Dieser Schritt bleibt $1s$ aktiv. Anschließend schaltet die Transition T2 und das Ventil wird wieder geschlossen. Es wird die Rückmeldung „E_SA_VAL_aus“ erzeugt, so dass der Generator vom Zustand 2 wieder in Zustand 1 wechseln kann. Durch die Abfrage der Rückmeldung vor dem Sprung zu Schritt 1 wird sichergestellt, dass das Ereignis genau einen Zyklus aktiv ist, bevor sie in Schritt 1 wieder zurückgesetzt wird.

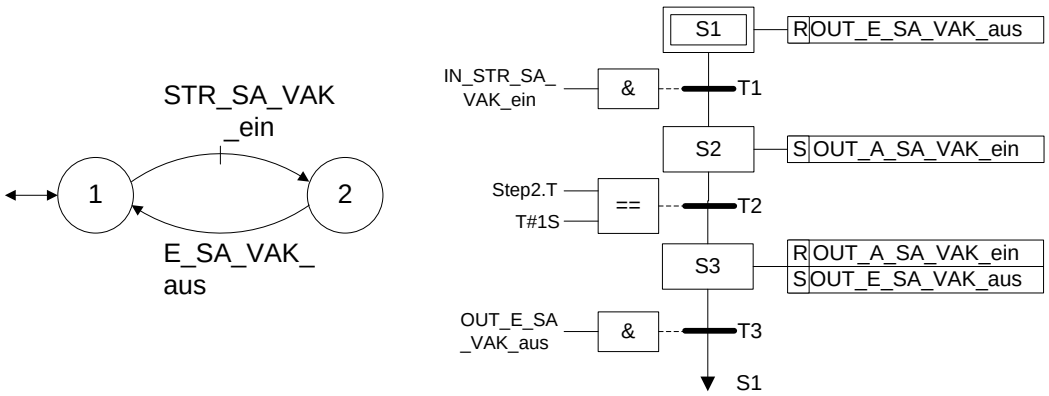


Abbildung 8.2.: Schrittkette der Vakuumsteuerung

Die Schrittketten der anderen Komponenten sind ähnlich aufgebaut. Sie werden durch ein steuerbares Ereignis gestartet und abgearbeitet. Am Ende wird eine Rückmeldung für die Supervisor erzeugt, die aus der Schrittkette selber oder durch einen Sensor von der Strecke generiert wird.

Robotersteuerung

Der Roboter wird über den Steuerungsrechner gesteuert. Die Steuersoftware ist in Visual Basic 6.0 implementiert und kommuniziert über die RS-232 Schnittstelle mit der Drive Unit D/U-M1. Die Form „com_schnittstelle.FRM“ von Crescent Software ermöglicht das Senden von Befehlen aus dem Programm an die Drive Unit. Sie bietet eine Vielzahl an Befehlssätze zum Steuern des Roboters (siehe [13]). Für die Steuerung in dieser Anwendung reichen drei Befehle:

1. **MP** Move Position (Absolute Position anfahren):
   Bewegt das Ende der Hand zu einer Position, deren Koordinaten (Position und Winkel) durch das Eingabeformat

   MP [*x-Achsen Koordinate*],[*y-Achsen Koordinate*],
   z-Achsen Koordinate,[*Neigungswinkel*],[*Rollwinkel*]

   definiert werden.

2. **GC**: Grip Close (Hand schließen):
   Dieser Befehl schließt die motorgesteuerte Hand.

3. **GO** Grip Open (Hand öffnen):
   Dieser Befehl öffnet die motorgesteuerte Hand.

Die Startposition des Roboters zwischen den beiden Rutschen wird durch die Befehlsfolge

```
pos$ = "-170.2,+268.5,+68.5,-89.6,-33.3"
RobotString$ = "MP " + pos$ + vbCr
MSComm1.Output = RobotString$
```

angefahren. Die Methode MSComm1.Output sendet den String über die RS-232 Schnittstelle an die Drive Unit. In identischer Weise wird der Befehl zum Öffnen bzw. Schließen des Greifers gesendet:

```
RobotString$ = "GC" + vbCr
MSComm1.Output = RobotString$.
```

Durch das Senden des Lesebefehls **WH** (Where, Momentane Position lesen) wird die aktuelle Position des Roboterarms gelesen. Durch die Befehlsfolge

```
Do
  Call wait_time
  MSComm1.Output = "WH" + vbCr
  Call wait_time
  actual_position = MSComm1.Input
Loop While (StrComp(set_position, actual_position) = 1)
```

wird abgewartet, bis die gewünschte Position erreicht wurde. Durch das Wiederholen dieser Befehlsfolgen können die in 3.2.4 genannten Trajektorien definiert und abgefahren werden. Die Kommunikation zwischen dem Steuerungsrechner und der SPS wird durch das AG-Link.BAS Modul von DELTALOGIC ermöglicht. Es stellt den Zugriff auf zwei Merkerbytes der SPS sicher. Dabei wird ein Merkerbyte zum Lesen und eins zum Schreiben der Software freigegeben. Zum Lesen der Befehle aus der SPS wird das Merkerbyte 100 (MB100) verwendet. Es wird wie folgt belegt:

- M100.0 STR_R_Nest:
  Führt die Nestfahrt aus.
- M100.1 STR_R_W1_RFID:
  Greift ein Werkstück von der Rutsche 1 (rot) und transportiert es zur RFID Station.
- M100.2 STR_R_W2_RFID:
  Greift ein Werkstück von der Rutsche 2 (schwarz) und transportiert es zur RFID Station.
- M100.3 STR_R_Lager:
  Transportiert das Werkstück zum Lager.
- M100.4 STR_R_akzeptabel:
  Transportiert das Werkstück zur Position „akzeptabel“.
- M100.5 STR_R_Ausschuss:
  Transportiert das Werkstück zur Position „Ausschuss“.
- M100.7 STR_R_Ruhe:
  Fährt die Ruheposition an.

Durch das Merkerbyte MB101 meldet die Steuersoftware die Abarbeitung des Auftrags mit den jeweiligen Bits zurück.

- M101.0 E_R_Nest
- M101.1 E_R_W1_RFID
- M101.2 E_R_W2_RFID
- M101.3 E_R_Lager
- M101.4 E_R_akzeptabel
- M101.5 E_R_Ausschuss
- M101.7 E_R_Ruhe

Abbildung 8.3 zeigt schematisch die Implementierung der Steuerungssoftware.

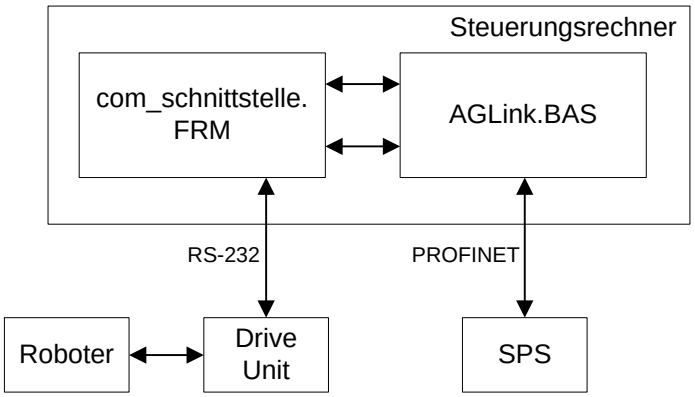


Abbildung 8.3.: Implemetierung der Steuerungssoftware für den Roboter

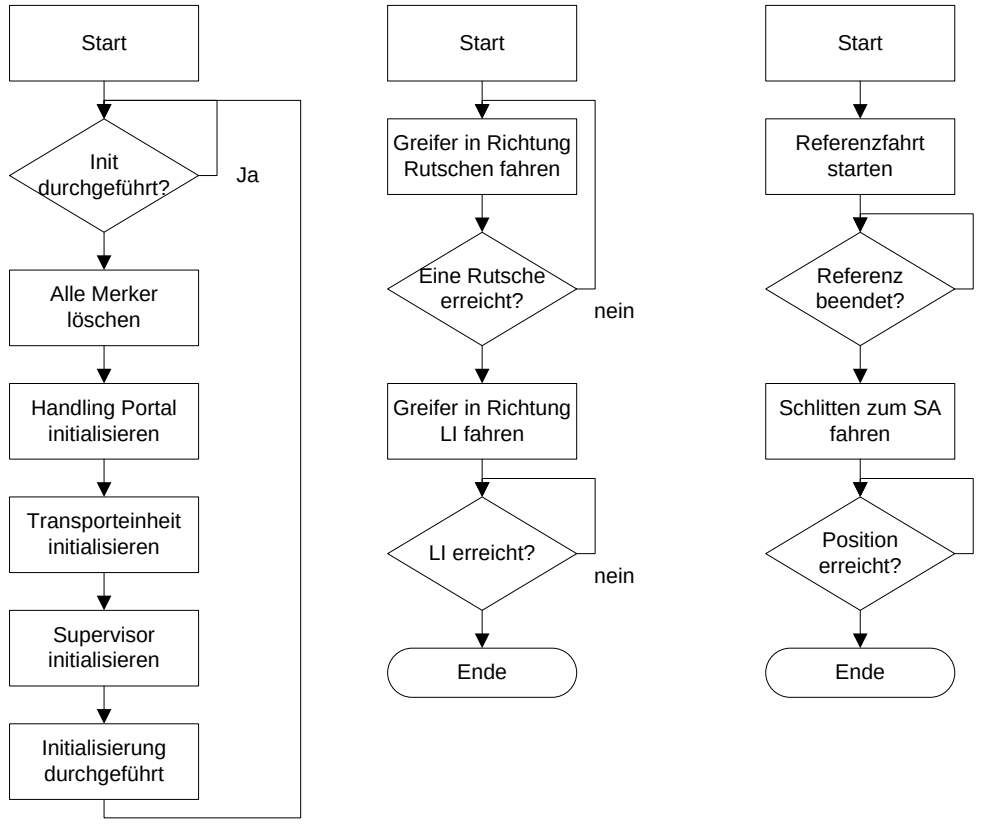
Initialisierung
Die Initialisierung findet direkt nach dem Einschalten der Anlage statt. Durch die Initialisierung wird die Anlage in einen definierten Zustand gefahren, an dem die Supervisor ansetzen. Sie ist ebenfalls in Form einer Schrittkette implementiert und enthält die in Abbildung 8.4a dargestellten vier Phasen.

In der ersten Phase werden alle Merker der Supervisor gelöscht. Dadurch wird sichergestellt, dass sich alle Supervisor nach der Initialisierungsphase in ihren Initialisierungs-Zuständen befinden. Zusätzlich werden die Schrittketten durch ihre Rücksetzfunktion „INIT_SQ“ zurückgesetzt.

Die zweite Phase initialisiert das Handling Portal (siehe Abbildung 8.4b). Befindet sich der Greifer nicht zufällig auf einem der Positionsschalter, so ist keine Aussage über seine aktuelle Position zu machen. Der Greifer wird in diesem Fall in Richtung der Rutschen gefahren, bis einer der Endschalter ausgelöst wird. Daraufhin wird der Greifer zur Position über der Transporteinheit gefahren. Ist diese Position erreicht, so ist die Initialisierung des Handling Portals abgeschlossen.

Die dritte Phase bildet die Initialisierung der Transporteinheit (siehe Abbildung 8.4c). Der Transportschlitten wird an den Endschalter der Linearachse gefahren, um eine definierte Position des Schrittmotors zu erlangen. Diese Funktion ist als Referenzpunktfahrt (Modus 1, siehe [39]) in der FM-353 bereits implementiert und wird durch eine weitere Schrittkette parametriert und gestartet. Nach Abschluss der Referenzfahrt wird der Transportschlitten zur Beladung auf die Position des Schwenkarms gefahren.

Die vierte und letzte Phase setzt die Supervisor auf ihre Anfangszustände. Dafür werden die entsprechenden Merker, die die Zustände repräsentieren, auf „true“ gesetzt. Die Initialisierung ist mit dem Setzten des Merkers „INIT_DONE“ abgeschlossen.



a) Allgemeiner Ablauf der Initialisierung

b) Ablauf Initialisierung des Handling Portals

c) Ablauf Initialisierung der Transporteinheit

Abbildung 8.4.: Programmablaufpläne der Initialisierung

Supervisor Implementierung
Die in Kapitel 6 modellierten Supervisor werden mit Hilfe von DES2IEC in die S7 Sprache AWL übersetzt. Die Programmcodes der Supervisor S_1 bis S_{12} werden in die Funktionen FC301 bis FC312 kopiert. In der Funktion FC400 erfolgt die Konjunktion der Steuereingriffe. Dafür werden die jeweiligen temporären steuerbaren Ereignisse aus den einzelnen Supervisor mit der „UND-Funktion“ verknüpft. Abbildung 8.5 zeigt die Verknüpfung der Bausteine und den Code aus dem Baustein FC400 für das steuerbare Ereignis „STR_SA_Lin“ in AWL, das in den Spezifikationen K_3, K_4, K_5 und K_9 verwendet wird. Das Ereignis darf nur gesetzt werden, wenn es von allen vier Spezifikationen erlaubt wird. Andere Spezifikationen erlauben das Ereignis zu jedem Zeitpunkt. Sie können demzufolge diese Ereignisse nicht verbieten und werden in der Konjunktion nicht berücksichtigt. Erst ein wahres Ergebnis der Konjunktion veranlasst das Setzen des Ereignisses.

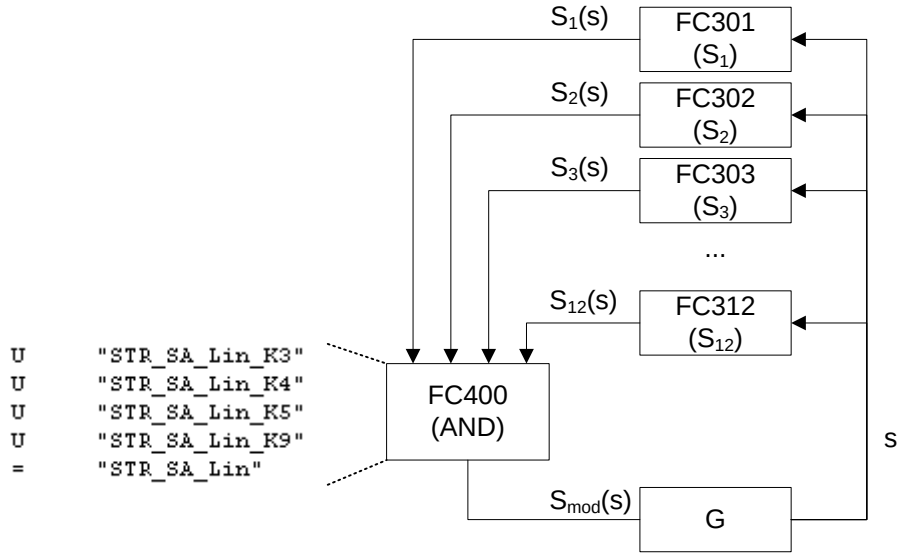


Abbildung 8.5.: Beispiel der Konjunktion

8.2. Betriebsergebnisse

Nach der Implementierung der Steuerung wurde die Anlage einem umfangreichen Praxistest unterzogen. Zur besseren Rekonstruktion des Verhaltens der Anlage wurde eine WinCC PC-Visualisierung implementiert, die es ermöglicht, dass Schalten der Supervisor zu beobachten. Die Abbildung 8.6 zeigt einen Screenshot der WinCC Oberfläche. Auf der abgebildeten Seite sind die Spezifikationen K_1 bis K_6 dargestellt. Die aktiven Zustände sind grün markiert. Der Screenshot zeigt einen Ausschnitt vom Beladungsprozess des Transportschlittens.

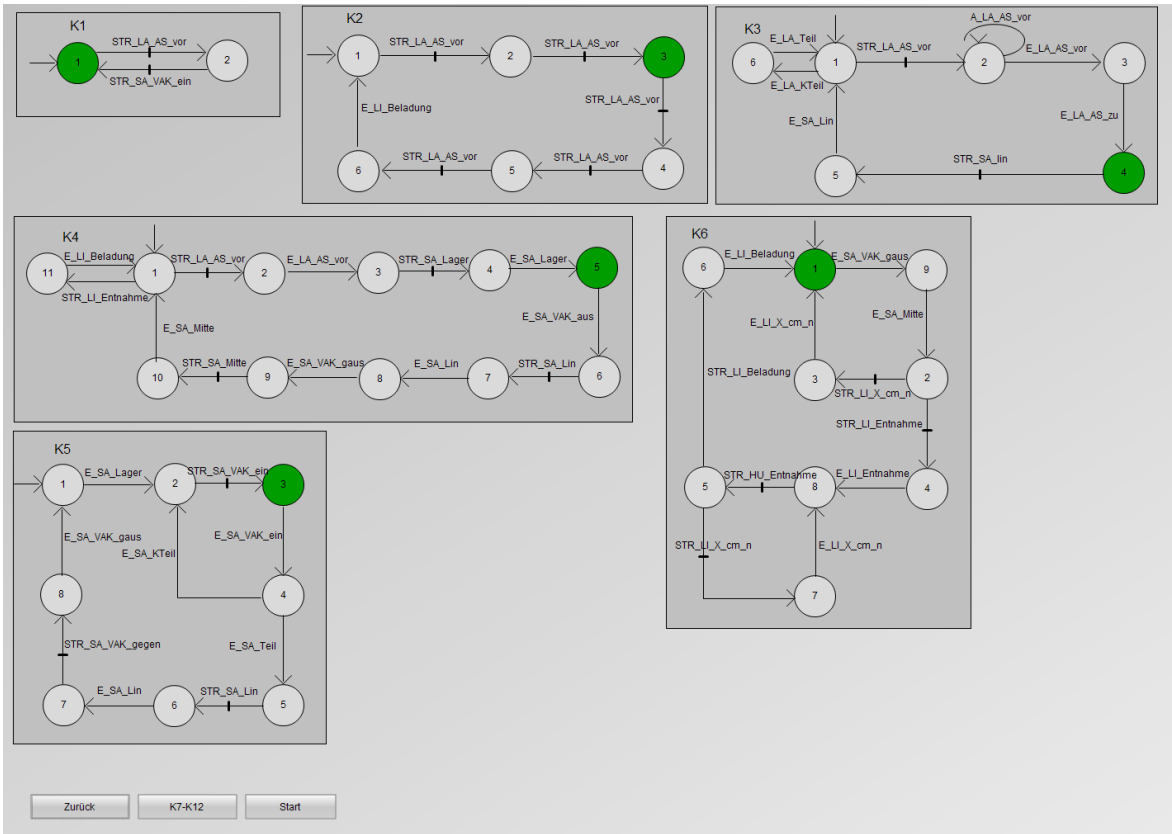


Abbildung 8.6.: Screenshot der WinCC Oberfläche

Die Ergebnisse der Tests ergaben, dass die erstellten Streckenmodelle und Spezifikationen für die gegebene Problemstellung und Steuerungsaufgabe korrekt sind. Es wurden Situationen getestet, die zu falschen Steuereingriffen geführt hätten, wenn die Supervisor nicht die unerwünschten Ereignisse blockiert hätten.

Eine Situation war das Beladen des Schlittens, dessen Kapazität auf 5 beschränkt und das Lager mit 10 Werkstücken beladen ist. Der Supervisor verhindert das weitere Beladen, wenn genau 5 Werkstücke auf dem Schlitten liegen.

Ein weiteres Beispiel beinhaltet das Füllen der Rutschen. Sie werden bis zur maximalen Kapazität beladen. Das Handling Portal wird durch die Pufferspezifikation gehindert, weitere Werkstücke auf die Rutsche zu legen, sobald die Rutschen voll sind. Erst wenn der Roboter ein Werkstück von der entsprechenden Rutsche entnommen hat, kann das Handling Portal ein weiteres Werkstück auf die Rutsche legen. Der Produktionsfluss konnte erfolgreich beendet werden.

9. Zusammenfassung und Ausblick

Zusammenfassung
Gegenstand der hier vorliegenden Arbeit war die Anwendung der Supervisory Control Theory auf ein reales System. Es sollte gezeigt werden, dass die SCT sich auf reale komplexe Systeme anwenden lässt. Dafür sollten geeignete Tools und Verfahren ausgewählt werden, die die Modellierung und Steuerungssynthese ermöglichen und praktikabel machen.

Die Modellierung und Steuerungssynthese wurde mit dem Entwicklungstool DESTool durchgeführt, dass durch seine umfangreiche Funktionsbibliothek, aber auch durch seine intuitive Bedienung überzeugen konnte.

Die Strecke wurde komponentenweise moduliert und durch Kompositionsoperationen zusammengeführt. Es konnten generische Modelle genutzt werden, so dass der Modulierungsaufwand reduziert werden konnte.

Der Steuerungsentwurf wurde mit dem modularen Ansatz realisiert. So konnten strukturiert Spezifikationen entwickelt werden, die zu einzelnen Supervisor geführt haben. Jeder Supervisor hat eine Teilaufgabe in der gesamten Steuerung übernommen. Die Spezifikationen waren allesamt steuerbar, so dass keine supremale steuerbare Teilsprache berechnet werden musste. Durch die Präfix-Geschlossenheit konnten die Spezifikationen direkt als Supervisor implementiert werden. Durch die Modularisierung der Strecke und der Spezifikationen konnte die Komplexität der implementierten Supervisor reduziert werden.

Die Komplexität des komponierten Supervisors konnte durch DESTool nicht bearbeitet werden, so dass es nicht möglich war, das Nichtblockieren der Anlage mit DESTool zu berechnen. Mit Hilfe von TCT konnte die Konfliktfreiheit der Supervisor ermittelt werden. Dies entspricht dem Nichtblockieren der Anlage, so dass die Steuerung dennoch verifiziert wurde.

Die Implementierung der Steuerungen erfolgte auf einer SIMATIC S7-300 Steuerung. Der erforderliche Code wurde durch einen in dieser Arbeit entwickelten Codegenerator erstellt. Dieser generiert aus den in DESTool modellierten Generatoren AWL Code nach der IEC 61131-3. Zur Generierung des Codes wurde das Verfahren nach Balemi et al. [5] ausgewählt.

Durch die Automatisierung der Codegenerierung können Änderungen an den Supervisor schnell durchgeführt werden, ohne dass dabei durch Unachtsamkeit Fehler entstehen. Die

Änderungen der Supervisor werden in DESTool durchgeführt. Nachdem die Untersuchungen des Supervisors durchgeführt wurden, kann der Code generiert und in die SPS kopiert werden, ohne dass weitere Fehlerquellen entstehen.

Die in Abschnitt 6.1 formulierte Steuerungsaufgabe wird ausnahmslos erfüllt, ohne zu blockieren. Um dies zu prüfen wurde die Steuerung umfangreichen Tests unterzogen. Es wurden Situationen untersucht, die ohne die korrekt funktionierenden Supervisor, die Anlage in unerwünschte Zustände gefahren hätte.

Ausblick
Die Fertigungszelle bietet Ansätze zur Optimierung der Steuerung hinsichtlich der Parallelität der vorhandenen Prozesse. Besonders hier besteht die Notwendigkeit von stabilen, fehlerfreien Entwicklungstools. DESTool bietet eine gute Grundlage zur Modellierung und Synthese, ist aber spürbar in der Entwicklungsphase. Zur Anwendung der SCT in der Industrie ist ein großer Bedarf an verlässlicher Software nötig.

Der in dieser Arbeit entwickelte Codegenerator bietet einen zusätzlichen Ansatz für weiterführende Arbeiten. Eine grafische Bedieneroberfläche erleichtert die Eingabe und Auswahl von Parametern und kann eine Fehlbedienung hinsichtlich inkorrekter Eingaben vermindern.

Eine Steuerungssynthese mit dem lokal-modularen Ansatz kann Inhalt einer weiterführenden Arbeit bieten. Dieser Ansatz würde ermöglichen, dass die gesamte Verifikation der Steuerung, mit Hilfe von DESTool automatisiert erfolgen könnte.

Fazit
Die SCT konnte erfolgreich auf die Fertigungszelle angewandt werden. Die Steuerung konnte nach dem modularen Ansatz entworfen und implementiert werden. Die Verifikation der Anlage konnte zum Teil nicht durch DESTool erfolgen, so dass das Prüfen auf Nichtblockieren der Steuerung durch TCT ermittelt wurde. Ein Entwurf mit dem lokal-modularen Ansatz hätte die algorithmische Komplexität gesenkt, aber gleichzeitig zu einem erhöhten Entwicklungs- und Implementierungsaufwand geführt.

Das in dieser Arbeit entstandene DES wurde von dem DESTool Entwickler zum Anlass genommen, die Speicherverwaltung von DESTool zu überarbeiten. So dass in Zukunft die Berechnung größerer Systeme auch mit diesem Tool durchzuführen ist.

Literaturverzeichnis

[1] AKESON, K. ; FABIAN, M. ; FLORDAHL, H.: Supremica in a Nutshell Draft. (2007)

[2] AKESON, K. ; FABIAN, M. ; FLORDAHL, H. ; VAHIDI, A.: Supremica - A Tool for Verification and Synthesis of Discrete Event Supervisors. (2004)

[3] AKESSON, K. ; M. FABIAN, A. V.: Supremica- A Tool for Verification and Synthesis of Discrete Event Supervisors. In: *Proceedings of the 11th Mediterranean Conference on Control and Automation* (2003)

[4] ANTASAKLIS, Panos J. ; MOODY, John O.: *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998. – ISBN 0-7923-8199-8

[5] BALEMI, S. ; HOFFMANN, G. J. ; GYUGYI, P. ; WONG-TOI, H. ; FRANKLIN, G. F.: Supervisory Control of a Rapid Thermal Multiprocessor. In: *IEEE Transactions on Automatic Control* 38 (1993), Juli, Nr. 7

[6] BRANDIN, B. A.: Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell. In: *IEEE Transactions on Robotics and Automation* 12 (1996), Februar, Nr. 1, S. 1–14

[7] BRANDT, R.D. ; GARG, V. ; KUMAR, R. ; LIN, F. ; MARCUS, S.I. ; WONHAM, W.M.: Formulas for calculating supremal controllable and normal sublanguages. In: *Systems Control Letters* 15 (1990), S. 111–117

[8] BUDHA, M. ; THAPA, D. ; PARK, S.C. ; WANG, G.N.: Generation of PLC Ladder Diagram Using Modular Structure. In: *International Conference on Intelligent Agents* (2008)

[9] CASSANDRAS, Christos G. ; LAFORTUNE, Stéphan: *Introduction to Discrete Event Systems, Second Edition*. Springer Verlag, 2010. – ISBN 978-1-4419-4419-0

[10] CIESLAK, R. ; DESCLAUX, C. ; FAWAZ, A.S. ; VARAIYA, P.: Supervisory control of discrete-event processes with partial observations. In: *IEEE Transactions on automatic control* 33 (1988), August, Nr. 3, S. 249–260

[11] COMMISSION, International E.: *IEC 61131-3: Programmable controllers - Part 3: Programming languages*. 2009

[12] DIERKS, Henning: PLC-automata: A new class of implementable real-time automata. In: *Theoretical Computer Science* 253 (2001), S. 61–93

[13] ELECTRIC, Mitsubishi: *Industrie-Roboter Anwender Handbuch RV-M1*

[14] FABIAN, M. ; HELLGREN, A.: PLC-based Implementation of Supervisory Control for Discrete Event Systems. In: *Proceedings of the 37th IEEE Conference on Decision and Control* (1998)

[15] GMBH, Bibliographisches I.: *Duden Online*. Januar 2012. – URL `http://www.duden.de/rechtschreibung/System#Bedeutung4`

[16] GOUYON, D. ; PETIN, J.F. ; GOUIN, A.: A pragmatic approach for modular control synthesis and implementation. In: *International Journal of Production Research* (2006)

[17] HASDEMIR, I.T ; KURTULAN, S. ; GÖREN, L.: Modular implementation of discrete event systems as sequential function charts applied to an assembly cell. In: *Discrete event systems 2004 (WODES'04)* (2003)

[18] HELLGREN, A. ; FABIAN, M. ; LENNARTSON, B.: Modular implementation of discrete event systems as sequential function charts applied to an assembly cell. In: *Proceedings of the 2001 IEEE International Conference on Control Applications* (2001)

[19] HELLGREN, A. ; LENNARTSON, B. ; FABIAN, M.: Modelling and PLC-based implementation of modular supervisory control. In: *Sixth International Workshop on Discrete Event Systems* (2002), Januar

[20] KIENCKE, Uwe: *Ereignisdiskrete Systeme Modellierung und Steuerung verteilter Systeme*. Oldenbourg Wissenschaftsverlag, 2006. – ISBN 978-3486580112

[21] LEDUC, Ryan J.: *PLC implementation of a DES Supervisor for a manufacturing testbed: an implementation perspective*, Dep. of Computer and Electrical Engineering, University of Toronto, Masterthesis, 1996

[22] LIN, F. ; WONHAM, W.M.: Decentralized supervisory control of discrete-event systems. In: *Information Sciences* 44 (1988), Nr. 3

[23] LIN, F. ; WONHAM, W.M.: Decentralized control and coordination of discrete-event systems with partial observation. In: *IEEE Transactions on Automatic Control* 35 (1990), Nr. 12, S. 1330–1337

[24] LIPSCHUTZ, S.: *Finite Mathematik*. McGraw-Hill Book Company GmbH, 1980. – ISBN 0-07-092025-8

[25] LIU, Jing ; DARABI, Houshand: Ladder logic implementation of Ramadge-Wonham supervisory controller. In: *Sixth International Workshop on Discrete Event Systems* (2002)

[26] LUNZE, Jan: *Ereignisdiskrete Systeme*. Oldenbourg Verlag München Wien, 2006. – ISBN 978-3-486-58071-6

[27] MALIK, Petra: *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems*, Fachbereich Informatik der Universität Kaiserslautern, Dissertation, 2003

[28] MALIK, R.: On the Set of Certain Conflicts of a Given Language. In: *Proc. of 7th International Workshop on Discrete Event Systems (WODES 2004)* (2004)

[29] MOOR, Thomas: *DESTool*. Februar 2012. – URL `http://www.rt.eei.uni-erlangen.de/FGdes/destool`

[30] MOOR, Thomas: *LibFAUDES*. März 2012. – URL `http://www.rt.eei.uni-erlangen.de/FGdes/faudes/`

[31] NORMUNG, Deutsches I. für: *DIN 19226-4 Leittechnik; Regelungs- und Steuerungstechnik; Begriffe für Regelungs- und Steuerungssysteme.* 1994

[32] PU, Ken Q.: *Modeling and control of Discrete-Event Systems with hierarchical abstraction*, Dep. of Computer and Electrical Engineering, University of Toronto, Masterthesis, 2000

[33] QUEIROZ, M. H. de ; CURY, J.R.E.: Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell. In: *Sixth International Workshop on Discrete Event Systems, 2002* (2002)

[34] QUEIROZ, M.H. de ; CURY, J.E.R: Modular Control of Composed Systems. In: *Proceedings of the American Control Conference* (2000)

[35] QUEIROZ, M.H. de ; CURY, J.E.R: Modular Supervisory Control of Large Scale Discrete Event Systems. In: *Discrete Event-Systems: Analysis and Control*. R. Boel (Hrsg.), 2000, S. 103–110

[36] RAMADGE, P.J.: *Supervision of discrete event processes*, Dep. of Electrical and Computer Engineering, University of Toronto, Dissertation, 1983

[37] RAMADGE, P.J. ; WONHAM, W.M.: The control of discrete event systems. In: *Proceedings of the IEEE* 77 (1989), Nr. 1, S. 81–98

[38] RAUSCH, Mathias P.: *Modulare Modellbildung, Synthese und Codegenerierung ereignisdiskreter Steuerungssysteme*. VDI Verlag, 1997. – ISBN 3-18-361308-5

[39] SIEMENS: *Positionierbaugruppe FM 353 für Schrittantrieb*. – URL `http://cache.automation.siemens.com/dnl/Tg/TgzNjIwMwAA_1111008_HB/Fm353v2_d.pdf`

[40] SIEMENS: *RFID SYSTEME - SIMATIC RF300 - Systemhandbuch*

[41] SIEMENS: *SIMATIC - Kontaktplan (KOP) für S7-300/400 - Referenzhandbuch*

[42] SIEMENS: *SIMATIC - S7-GRAPH für S7-300/400 - Ablaufsteuerungen programmieren*

[43] SIEMENS: *SIMATIC Controller Software - Werkzeuge zum Projektieren und Programmieren von SIMATIC Controllern*. – URL `http://www.automation.siemens.com/salesmaterial-as/brochure/de/brochure_simatic-industrial-software_de.pdf`

[44] SIEMENS: *SIMATIC Sensors - RFID Systeme FB45*. – URL `http://cache.automation.siemens.com/dnl/DY/DYxMTg3NwAA_21738808_HB/FB45_FHB_200910_de.pdf`

[45] UZAM, M. ; GELEN, G. ; DALCI, R.: A new approach for the ladder logic implementation of Ramadge-Wonham supervisors. In: *XXII International Symposium on Information, Communication and Automation Technologies* (2009)

[46] VIEIRA, A. D. ; CURY, J. E. R. ; QUEIROZ, M. H.: A Model for PLC Implementation of Supervisory Control of Discrete Event Systems. In: *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)* (2006), Mai

[47] WENCK, Florian: *Modellbildung, Analyse und Steuerungsentwurf für gekoppelte ereignisdiskrete Systeme*. Shaker, 2006. – ISBN 978-3-8322-5573-2

[48] WONHAM, W.M.: Supervisory Control of Discrete Event Systems: An introduction. (1999)

[49] WONHAM, W.M.: *Supervisory Control of Discrete- Event Systems*. Dep. of Electrical and Computer Engineering, University of Toronto, Kanada, 2004

[50] WONHAM, W.M. ; RAMADGE, P.J.: Modular Supervisory Control of Discrete-Event Systems. In: *Mathematics of Control, Signals, and Systems* (1988), Nr. 1, S. 13–30

A. Verzeichnis über den Anhang auf der DVD

Der Anhang dieser Arbeit befindet sich auf der beiliegenden DVD. Sie kann bei Herrn Prof. Wenck eingesehen werden.

A.1. DES2IEC

Der Ordner „DES2IEC“ enthält zum einen die ausführbare Datei „DES2IEC.exe“, zum anderen den Ordner „DES2IEC_Workspace“ der den Microsoft Visual C++ 6.0 Workspace und alle dazugehörigen Quellcodes beinhaltet.

A.2. DESTool

Der Ordner „DESTool“ enthält das DESTool Projekt (Version 0.66) mit allen Streckenkomponenten und Spezifikationen.

A.3. Steuerung

Der Ordner „Steuerung“ enthält das archivierte Siemens STEP7 und das Visual Basic 6.0 Projekt zur Steuerung der Anlage und des Roboters.

A.4. WinCC Oberfläche

Der Ordner „WinCC Oberfläche“ enthält die Visualisierung der einzelnen Supervisor.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 29. März 2012	
Ort, Datum	Unterschrift