7 Synchronisation

7.1 Problem. (Firing Squad Synchronisation Problem; FSSP) Gegeben ist eine Zustands(teil)menge $Q' = \{\#, g, s, f\}$. Gesucht ist ein eindimensionaler Zellularautomat C mit $H_1^{(1)}$-Nachbarschaft und Zustandsmenge $Q \supseteq Q'$, der die folgenden Eigenschaften hat:

1. $\{\#, s\}$ ist eine passive Zustandsmenge mit totem Ruhezustand #.
2. C überführt jede Konfiguration der Form $c_{gs^{n-1}} = \#gss\cdots s\#$ in die Konfiguration $c_{f^n} = \#fff\cdots f\#$ mit gleichem Träger, so dass
3. dabei in keiner der vorher während der Berechnungen auftretenden Konfigurationen der Zustand f vorkommt.

Man beachte, dass für *alle* Anfangskonfigurationen gleich welcher Länge n die *gleiche* Überführungsfunktion benutzt werden muss! Da die Zustandsmenge *endlich* sein muss, bedeutet das insbesondere, dass man im allgemeinen nicht in einem Zustand die Anzahl der zu synchronisierenden Zellen speichern kann. Die erste Forderung verbietet insbesondere, dass einfach alle Zellen im ersten Schritt in Zustand f übergehen.

Wir skizzieren zunächst einen „langsamen" Algorithmus von Balzer, um Prinzipien klar zu machen.

7.2 Algorithmus. (FSSP-1DIM-KLASSISCH-LANGSAM) Der größeren Übersichtlichkeit wegen sind im Gegensatz zu früheren Beispielen in Abbildung 1.1 nicht die Register in getrennten Kästchen dargestellt. Als General fungiert der Zustand (1> >.

Die Idee besteht darin, den zu synchronisierenden Bereich „in der Mitte zu teilen" und die beiden Hälften dann rekursiv nach dem gleichen Verfahren getrennt zu synchronisieren. Die Rekursion bricht ab, wenn die Abschnitte so klein geworden sind, dass jede Randzelle eines Abschnittes ihr Pendant in ihrer Nachbarschaft „sieht".

Da zu Beginn immer nur die g-Zelle „aktiv" ist, werden zur Bestimmung der Mitte von ihr zwei Signale mit den Geschwindigkeiten 1 und 1/3 nach rechts geschickt. Das schnellere wird am am weitesten rechts gelegenen s (das dabei explizit als rechter Rand markiert wird) „reflektiert" und läuft ebenso schnell zurück. Die beiden Signale treffen sich „in der Mitte" (die aus einer oder zwei Zellen bestehen kann; siehe Algorithmus 6.13).

Durch (und) sind die Grenzen von Synchronisationsabschnitten gekennzeichnet, die stets rekursiv nach dem (im Prinzip) gleichen Algorithmus weiterbearbeitet werden. Wie man sieht, kann eine Zelle unter Umständen sowohl nach links als auch nach rechts als Grenze wirken. Die Rekursion kann abgebrochen werden, sobald jede linke Grenze ihr Pendant (in der rechten Nachbarzelle) sieht und umgekehrt. Das geschieht stets überall gleichzeitig. Dann ist der Zeitpunkt erreicht, zu dem alle Zellen in den Zustand f übergehen.

Bezeichnet $t(n)$ den Zeitbedarf dieses Algorithmus für die Synchronisation eines Abschnittes mit n Zellen, so ist ungefähr $t(n) \approx \frac{3}{2}n + t(\frac{n}{2})$. Hieraus ergibt sich ein Zeitbedarf von $t(n) = 3n +$ Terme niedrigerer Ordnung.

Es zeigt sich, dass dieser Algorithmus noch nicht zeitoptimal ist:

(1> >	s	s	s	s	s	s	s	s	s
(2>	>	s	s	s	s	s	s	s	s
(3>		>	s	s	s	s	s	s	s
(	1>		>	s	s	s	s	s	s
(	2>			>	s	s	s	s	s
(	3>				>	s	s	s	s
(		1>				>	s	s	s
(		2>					>	s	s
(		3>						>	s
(			1>						<)
(			2>					<	)
(			3>				<		)
(				1>		<			)
(				2>	<				)
(				< <1)	(1> >				)
(			<	<2)	(2>	>			)
(		<		<3)	(3>		>		)
(	<		<1	)	(	1>		>	)
(>			<2	)	(	2>			<)
(	>		<3	)	(	3>		<	)
(		<<1)(1>>		)	(		<<1)(1>>		)
(	<	<2)(2>	>	)	(	<	<2)(2>	>	)
(>		<3)(3>		<)	(>		<3)(3>		<)
(	<<1)(1>>	)(	<<1)(1>>	)	(	<<1)(1>>	)(	<<1)(1>>	)
f	f	f	f	f	f	f	f	f	f

Abbildung 7.1: Skizze von Balzers „langsamem“ Algorithmus für das FSSP.

7.3 Satz. *Es gibt keinen Zellularautomaten, der das FSSP für alle n löst und für ein $n \geqslant 2$ höchstens $2n-3$ Schritte benötigt (Waksman 1966).*

Es gibt einen Zellularautomaten, der das FSSP für alle $n \geqslant 2$ in genau $2n-2$ Schritten löst (Goto 1962). Man kommt dabei sogar mit nur 7 Zuständen aus (Mazoyer 1987).

Es liegt hier also einer der seltenen Fälle vor, in denen man absolut zeitoptimale Algorithmen kennt, d. h. die bekannten oberen und unteren Schranken für die Laufzeit lückenlos aneinander stoßen.

7.4 Beweis (der unteren Schranke) Dass es tatsächlich zeitoptimale Lösungen für das FSSP gibt, werden wir gleich noch anhand der Algorithmen von Gerken und Mazoyer sehen. Wir beschränken uns daher hier auf den Nachweis der Behauptung, dass $2n-2$ untere Schranke in der behaupteten Form ist.

Der Beweis wird indirekt geführt. Angenommen, ein Zellularautomat würde das FSSP lösen und für ein k nur $2k-3$ Schritte benötigen. (Der Fall noch weniger Schritte kann analog behandelt werden.) Das Raum-Zeit-Diagramm für dieses k sieht dann so aus wie im linken Teil von Abbildung 1.2 dargestellt. Grau unterlegt ist der Bereich, der einen „Einfluss“ auf den Zustand von Zelle 1 zum Zeitpunkt $2k-3$ hat. Insbesondere ist die Zelle k zu allen dazu in Frage kommenden Zeitpunkten $0 \leqslant t \leqslant k-2$ noch im Zustand s.

Dies gilt natürlich auch für alle längeren Anfangskonfigurationen. Betrachten wir zum Beispiel den Fall gs^{2k-2}. Auch hier ist Zelle k zu Zeitpunkten t mit $0 \leqslant t \leqslant k-2$ noch im Zustand s. Also stimmen die gesamten in Abbildung 1.2 grau unterlegten Bereiche der beiden

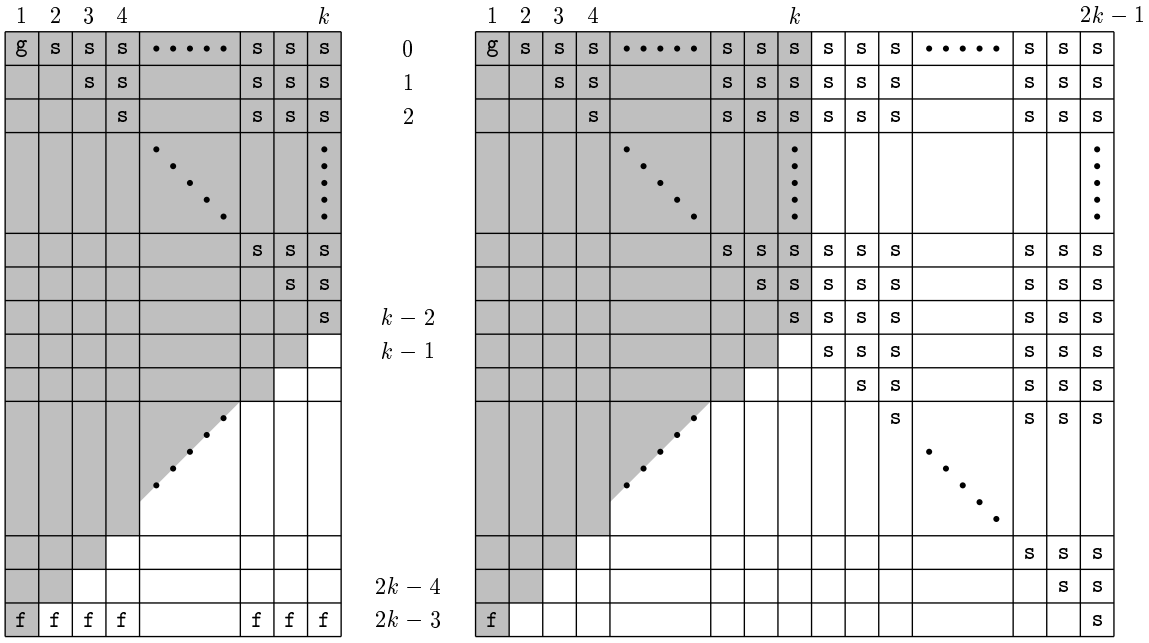

Abbildung 7.2: Zum Beweis 1.4. Links: Struktur des Raum-Zeit-Diagramms für die Synchronisation von k Zellen, falls sie in $2k-3$ Schritten durchgeführt werden könnte. Rechts: Struktur des Raum-Zeit-Diagramms für die Arbeit des gleichen Algorithmus für die Anfangskonfiguration mit $2k-1$ Zellen.

Raum-Zeit-Diagramme überein. Insbesondere ist also auch im rechten Diagramm Zelle 1 zum Zeitpunkt $2k-3$ im Zustand `f`. Andererseits muss aber wegen Bedingung 1 aus 1.1 gleichzeitig Zelle $2k-1$ auf jeden Fall noch im Zustand `s` sein. Damit ist aber Bedingung 3 aus 1.1 verletzt. Der Zellularautomat löst also gar nicht das FSSP im Widerspruch zur Annahme. ■

7.5 Algorithmus. (`FSSP-1DIM-GERKEN`) Wir skizzieren einen Algorithmus aus der Diplomarbeit von Gerken (1987); siehe Abbildung 1.3. Er benutzt Algorithmus 6.7 `ZICKZACK-SIGNAL`, um die linken zwei Drittel des Trägers in Absschnitte mit von links nach rechts exponentiell wachsender Größe aufzuteilen. Daran schließt sich unter Umständen eine „Lücke“ bis zur Zweidrittelgrenze an. Das rechte Drittel bildet den letzten Abschnitt. Alle Abschnitte werden rekursiv nach dem gleichen Algorithmus bearbeitet. Das Kriterium für den Rekursionsabbruch soll wie in Algorithmus 1.2 die Tatsache sein, dass sich jeder Abschnitt nur noch 2 Zellen erstreckt. Dann können alle Zellen in den Zustand `f` übergehen. Damit dies für alle Abschnitte gleichzeitig passiert, müssen in ihnen die Synchronisationen zu geeigneten Zeitpunkten gestartet bzw. zwischenzeitlich „eingefroren“ werden.

Wirklich problematisch ist dabei nur die Lücke. Ihre Synchronisation wird zu dem Zeitpunkt gestartet, zu dem die Zweidrittelgrenze markiert wird. Damit der Algorithmus in diesem Abschnitt nicht zu früh endet, wird er vorübergehend „eingefroren“ (und später wieder aufgetaut). Eine genauere Betrachtung von Abbildung 1.3 zeigt, dass die entsprechenden Signale leicht erzeugt werden können.

7.6 Algorithmus. (`FSSP-1DIM-MAZOYER`) Auch Mazoyer benutzt eine Divide-and-conquer-Strategie. Wie im Algorithmus von Gerken wird der Beginn des rechten Drittels markiert und es synchro-

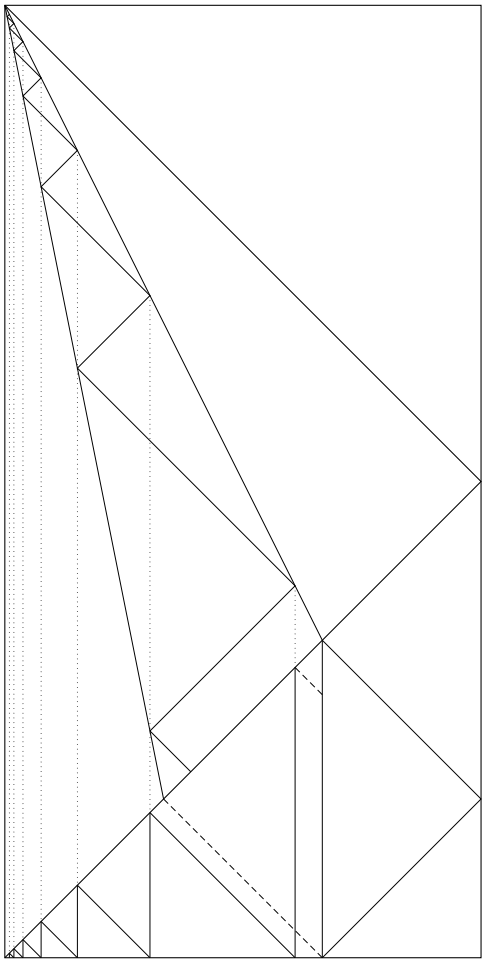

Abbildung 7.3: Skizze des FSSP-Algorithmus von Gerken. Die gestrichelt dargestellten Signale frieren den Ablauf des Synchronisationsalgorithmus in der Lücke links neben dem rechten Drittel ein bzw. tauen ihn wieder auf.

nisiert. Von den verbleibenden linken zwei Dritteln wird wieder der Beginn des rechten Drittels markiert und es synchronisiert. Von den verbleibenden linken vier Neunteln wird wieder der Beginn des rechten Drittels markiert und es synchronisiert. Und so weiter und so weiter. In Abbildung 1.4 ist die prinzipielle Vorgehensweise skizziert.

Es ist sehr erstaunlich, dass man mit nur 7 Zuständen (einschließlich `f`, `g`, `s` und `#` !) auskommt. Bezeichnet man die zusätzlichen Zustände mit `a`, `b` und `c`, so wählt Mazoyer die „Überführungstabellen“ für sie und `g` und `s` wie folgt:

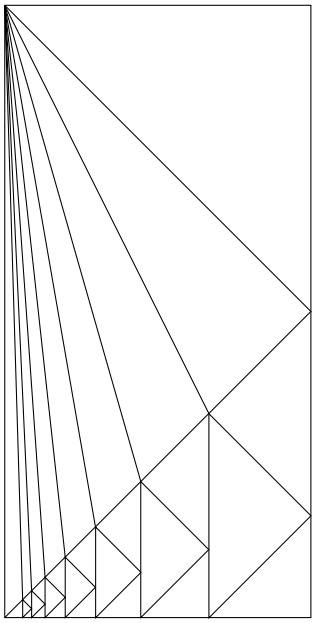

Abbildung 7.4: Skizze des FSSP-Algorithmus von Mazoyer.

s	#	s	a	b	c	g
#		s				
s	s	s		s	s	s
a	c	g	s	s	s	c
b	s	s	s	s	s	s
c	g	a	s	s	s	g
g	a	c	s	s	s	a

a	#	s	a	b	c	g
#			f		g	
s			a	s	g	
a	f	a	a	b	c	b
b	c	g		g	c	c
c		a	a			
g	c				c	c

b	#	s	a	b	c	g
#						
s			g	b	s	b
a		g	b	b	s	
b		g	a	b	c	b
c	s	s	a			s
g	g	c	c		b	g

c	#	s	a	b	c	g
#						
s		c	a	g	c	g
a	b	b		b		b
b	g	c			c	g
c		c	a	b	c	b
g	b	b		b		b

g	#	s	a	b	c	g
#		a		g	g	f
s			g	g	g	
a		b		g	g	
b	g	b		g	g	g
c	a	a		g	g	a
g	f	b		g	g	f

Für eine lokale Konfiguration l findet man $\delta(l)$, indem man in der Tabelle mit $l(0)$ in der oberen linken Ecke den Zustand in Zeile $l(-1)$ und Spalte $l(1)$ aufsucht. für # und f müssen für die korrekte Arbeitsweise des Algorithmus keine Funktionswerte spezifiziert werden. Ebenso sind in den Tabellen diejenigen Stellen frei gelassen, die lokalen Konfigurationen entsprechen, die nicht auftreten können und sowie die Zeilen und Spalten die Nachbarzuständen f entsprechen.

Sofern man sich nicht gleich eine Reihe diskreter Raum-Zeit-Diagramme als Beispiele ansieht, bleibt die Frage zu klären, wie man die unbeschränkte Anzahl von Signalen paarweise unterschiedlicher Geschwindigkeiten erzeugt. Da die Zustandsmenge endlich sein muss, kann man offensichtlich nicht für jedes die Modulozählerkonstruktion aus Algorithmus 6.5 benutzen.

Wir erläutern das Prinzip für den etwas anschaulicheren Fall, dass man Signale S_0, S_1, usw.

mit den Geschwindigkeiten 1/1, 1/3, 1/7, 1/15, usw. konstruieren möchte, die sich etwa alle nach rechts bewegen sollen: S_0 wird auf die bekannte Art und Weise realisiert. Außerdem sendet es in jedem Schritt ein Signal < mit Geschwindigkeit 1 nach links. Da sich S_0 gleichzeitig weiterbewegt, laufen die <-Signale „mit einer Zelle Zwischenraum" nach links.

Jedes langsamere Signal S_i, $i \geqslant 1$, ist durch eine Marke realisiert, die sich *nicht* „von selbst" bewegt. Wann immer ein S_i von einem <-Signal „getroffen" wird, bewegt es sich um eine Zelle nach rechts. Außerdem löscht es jedes zweite <-Signal. Die anderen lässt es passieren. Dadurch wird S_{i+1} nur von halb so vielen <-Signalen getroffen wie S_i und bewegt sich daher langsamer. Eine neue Marke wird immer dann am linken Rand erzeugt, wenn ein <-Signal dort ankommt.

In Abbildung 1.5 sind die ersten Schritte des Algorithmus dargestellt. Marken sind durch `0` und `1` dargestellt. Dabei lassen `1`-Marken ein <-Signal passieren, `0`-Marken löschen es.

7.7 Problem. (1-dimensionales FSSP mit dem General an beliebiger Stelle) Die Aufgabenstellung ist analog zu der für das „klassische" FSSP 1.1, nur dass der Zellularautomat diesmal jede Konfiguration der Form `#s`$\cdots$`sgs`$\cdots$`s#` mit einer (aber auch nur einer) `g`-Zelle an *beliebiger* Stelle in die Konfiguration `#fff`$\cdots$`f#` mit gleichem Träger überführen soll.

Natürlich muss wieder die *gleiche* Überführungsfunktion für *alle* Anfangskonfigurationen das Gewünschte leisten.

7.8 Satz. *Das FSSP mit dem General an beliebiger Stelle ist in* $2n - 2 - k$ *Schritten lösbar, wobei* n *die Größe des Trägers und* k *die Länge des kürzeren Abschnittes neben dem General (ohne diesen) ist. Diese Zeit ist für* $n \geqslant 2$ *optimal.*

7.9 Man beachte, dass die Zeit $2(n-k)-2$, also die minimal nötige Zeit für die Synchronisation nur des größeren Abschnittes *nicht* ausreicht (wenn die beiden Abschnitte unterschiedlich groß sind). Den Beweis kann man analog zu Beweis 1.4 führen.

7.10 Algorithmus. (`FSSP-1DIM-GENERAL-BELIEBIG`) Wir skizzieren den Ablauf: Der am Anfang in Zelle $\mathbf{x}_g$ vorhandene General sendet mit Geschwindigkeit 1 Signale an beide Ränder, die dort FSSP-Algorithmen starten, deren Wirkungsbereich sich bis zum Treffpunkt $\mathbf{x}_t$ der reflektierten Signale erstreckt. Im schmaleren Bereich muss der Ablauf ähnlich wie in Algorithmus 1.5 zwischendurch eingefroren werden. Dies geschieht ausgehend von $\mathbf{x}_t$ durch ein Signal mit Geschwindigkeit 1. Aufgetaut wird der Algorithmus durch eine Signal, das von der Mitte $\mathbf{x}_m$ des Trägers gestartet wird. Einfrier- und Auftausignal sind in Abbildung 1.6 gestrichelt dargestellt.

Zelle $\mathbf{x}_m$ kann ebenfalls als Treffpunkt zweier Signale bestimmt werden, da sie auch in der Mitte zwischen $\mathbf{x}_g$ und $\mathbf{x}_t$ liegt. Sie sind in Abbildung 1.6 gepunktet dargestellt und haben die Geschwindigkeiten 1/3 und 1.

7.11 Problem. (FSSP für Rechtecke) Im Zweidimensionalen betrachtet man im einfachen Fall als Anfangskonfigurationen solche mit rechteckigem Träger, dessen Zellen alle im Zustand `s` sind mit Ausnahme der linken oberen Ecke, die im Zustand `g` ist. Alle Zellen sollen gleichzeitig erstmals in den Zustand `f` übergehen.

7.12 Algorithmus. (`FSSP-2DIM-RECHTECK`) Das Rechteck bestehe aus m Zeilen und n Spalten. Der aus dem General, der von ihm ausgehenden Zeile `0` und der von ihm ausgehenden Spalte `0` bestehende „Haken" M_0 kann mit einer einfachen Variante von Algorithmus `FSSP-1DIM-BELIEBIG` in $2(m+n-1)-2-(m-1) = m+n+\max\{m,n\}-3$ Schritten synchronisiert werden. Analog benötigt der sich innen anschließende Haken M_1 $m-1+n-1+\max\{m-1,n-1\}-3 = m+n+\max\{m,n\}-3-3$ Schritte, d.h. drei weniger. Seine Synchronisation wird also zum gleichen Zeitpunkt fertig, wenn sie drei Schritte später gestartet wird.

>											
	< >										
1		< >									
1	< 0		< >								
1	0	<		< >							
1	1		<		< >						
1	1	< 0		<		< >					
1	< 0	0	<		<		< >				
1	0	1		<		<		< >			
1	0	1	< 0		<		<		< >		
1	0	<	0	<		<		<		< >	
1	1		1		<		<		<		< >
1	1		1	< 0		<		<		<	
1	1		<	0	<		<		<		<
1	1	< 0		1		<		<		<	
1	< 0	0		1	< 0		<		<		<
1	0	0		<	0	<		<		<	
1	0	0	<		1		<		<		<
1	0	1			1	< 0		<		<	
1	0	1			<	0	<		<		<
1	0	1		<		1		<		<	
1	0	1	< 0			1	< 0		<		<
1	0	<	0			<	0	<		<	
1	1		0		<		1		<		<
1	1		0	<			1	< 0		<	
1	1		1				<	0	<		<
1	1		1			<		1		<	
1	1		1		<			1	< 0		<
1	1		1	< 0				<	0	<	
1	1		<	0			<		1		<
1	1	< 0		0		<			1	< 0	
1	< 0	0		0	<				<	0	<
1	0	0		1				<		1	
1	0	0		1			<			1	< 0
1	0	0		1		<				<	0
1	0	0		1	< 0				<		1

Abbildung 7.5: Sukzessive Erzeugung der Signale mit Geschwindigkeiten $1/(2^i - 1)$ für $i = 1, 2, \ldots$.

Allgemein startet man die Synchronisation aller Zellen der Menge $M_i = \{(i, j) \mid j \geqslant i\} \cup \{(j, i) \mid j \geqslant i\}$ (mit $i \geqslant 0$) nach $3i$ Schritten. Dazu lässt man vom ursprünglichen General ein Signal mit Geschwindigkeit $\frac{1}{3}$ entlang der Diagonale laufen und nach und nach die Synchronisationen der (immer kleiner werdenden) Mengen M_i von Zelle (i, i) starten, so dass alle gleichzeitig fertig sind. In Abbildung 1.7 sind die Zellen, die als Generäle für die Mengen M_i fungieren grau dargestellt.

7.13 Wer wissen will, wie man d-dimensionale Quader mit dem General an beliebiger Stelle zeitoptimal synchronisiert, lese die Arbeit von Szwerinski (1982).

Oder man versuche es selbst. Die optimale Zeit ist im Zweidimensionalen $m + n + \max\{m, n\} -$

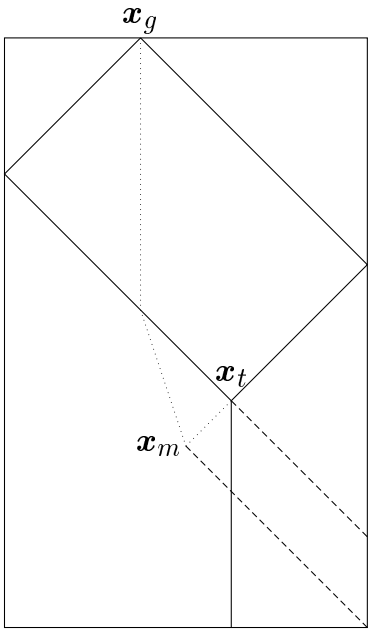

Abbildung 7.6: Zur Synchronisation eines Trägers mit dem General an beliebiger Stelle.

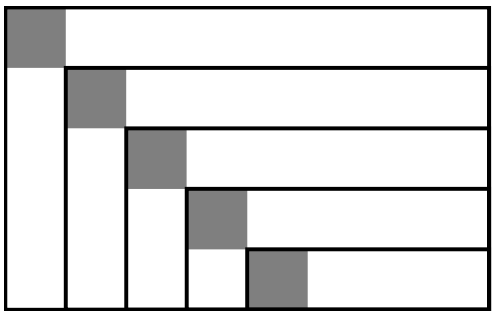

Abbildung 7.7: Synchronisation von Rechtecken. Die grauen Zellen fungieren nach und nach als Generäle.

$3-(i-1)-(k-1)$, wobei m und n die Längen des Rechteckes sind und i und k die jeweils kürzeren Abstände des Generals von den Rändern.

Eine zusätzliche Verkomplizierung der Aufgabe besteht darin, dass die Träger nicht mehr regelmäßige Quader sondern beliebige zusammenhängende Gebiete sein dürfen.

Eine weitere Verallgemeinerung besteht darin, dass der Algorithmus auch dann noch funktionieren muss, wenn am Anfang unter Umständen *mehrere* g-Zellen vorhanden sind. Im Eindimensionalen ist dieses Problem noch relativ einfach lösbar, sogar zeitoptimal. Im Zweidimensionalen erlebt man Überraschungen. Dann wird es zum Beispiel anscheinend wichtig, welche Nachbarschaft man verwendet. Für Moore-Nachbarschaft (mit Radius 1) kennt man einen zeitoptimalen Algorithmus; für von Neumann-Nachbarschaft ist noch nicht einmal klar, ob im Allgemeinen ein zeitoptimaler Algorithmus existiert. Diese und andere Ergebnisse findet man in der Diplomarbeit von Schmid (2003), die den aktuellen Stand der Technik repräsentiert.

Noch schwieriger wird es, wenn im Laufe eines bereits gestarteten Synchronisationsvorganges zusätzlich irgendwelche vorher noch in Ruhe befindlichen Zellen „von außen“ zu g-Zellen gemacht werden können. Eine mögliche Vorgehensweise besteht dann darin, festzustellen, welche g-Zellen als erste vorhanden waren; siehe zum Beispiel Vollmar (1977).

Die Aufgabenstellung, überhaupt nur die Zelle zu markieren, die als erste im Zustand g war, wird als *Early Bird Problem* bezeichnet; siehe Rosenstiehl, Fiksel und Holliger (1972) und Vollmar (1977).

Zusammenfassung

- Beim eindimensionalen FSSP ist für jeden der Fälle

noitemisep ein General am linken Ende oder an beliebiger Stelle

noitemisep mehrere Generäle an beliebigen Stellen

zeitoptimale Lösungen bekannt.

- Im zweidimensionalen Fall ist das nur für die Fälle eines Generals bekannt.

Literatur

Gerken, H.-D. (1987). „Über Synchronisationsprobleme bei Zellularautomaten“. Diploma thesis. Technische Universität Braunschweig (siehe S. 3).

Goto, E. (1962). *A minimum time solution of the Firing Squad Problem*. Course notes, Harvard University (siehe S. 2).

Mazoyer, Jacques (1987). „A six-state minimal time solution to the firing squad synchronization problem“. In: *Theoretical Computer Science* 50, S. 183–238 (siehe S. 2).

Rosenstiehl, P., J. R. Fiksel und A. Holliger (1972). „Intelligent Graphs: Networks of Finite Automata Capable of Solving Graph Problems“. In: *Graph Theory and Computing*. Hrsg. von R. C. Read. New York: Academic Press, S. 219–265 (siehe S. 9).

Schmid, Hubert (2003). „Synchronisationsprobleme für zelluläre Automaten mit mehreren Generälen“. Diploma thesis. Fakultät für Informatik, Universität Karlsruhe (siehe S. 8).

Szwerinski, Helge (1982). „Time Optimal Solution of the Firing Squad Synchronization Problem for n-dimensional Rectangles with the General at an Arbitrary Position“. In: *Theoretical Computer Science* 19, S. 305–320 (siehe S. 7).

Vollmar, Roland (1977). „On two modified problems of synchronization in cellular automata“. In: *Acta Cybernetica* 3, S. 293–300 (siehe S. 9).

Waksman, A. (1966). „An Optimum Solution to the Firing Squad Synchronization Problem“. In: *Information and Control* 9, S. 66–78 (siehe S. 2).

