

Converting between various sequence representations

Gilbert Ritschard, Alexis Gabadinho, Matthias Studer and Nicolas S. Müller

Please cite as: Ritschard, G., Gabadinho, A., Studer, M. & Müller, N.S. (2009), "Converting between various sequence representations", In Ras, Z. & Dardzinska, A. (eds) *Advances in Data Management. Series: Studies in Computational Intelligence*. Volume 223, pp. 155-175. Berlin: Springer. DOI 10.1007/978-3-642-02190-9_8

Abstract This chapter is concerned with the organization of categorical sequence data. We first build a typology of sequences distinguishing for example between chronological sequences and sequences without time content. This permits to identify the kind of information that the data organization should preserve. Focusing then mainly on chronological sequences, we discuss the advantages and limits of different ways of representing time stamped event and state sequence data and present solutions for automatically converting between various formats, e.g., between horizontal and vertical presentations but also from state sequences into event sequences and reciprocally. Special attention is also drawn to the handling of missing values in these conversion processes.

Key words: Sequence data organization, State sequence, Event sequence, Transition, Converting between sequence formats.

1 Introduction

Categorical sequence data appear in many different fields. We encounter for instance word or letter sequences in text mining, protein or DNA sequences in biology, functioning state sequences in device control, sequences of successively visited web pages in web log analysis and biographical data describing life trajectories in social sciences. There are also multiple ways of analysing such data: Markov chain models and their extensions for analysing transitions between states, data-mining-based methods for discovering regular patterns in sequences, aligning techniques for finding component similarities, edit based distances for measuring proximities between sequences, survival analysis for studying time-to-event distributions to men-

G. Ritschard, A. Gabadinho, M. Studer and N. S. Müller
Department of Econometrics and Laboratory of Demography, University of Geneva, Switzerland,
e-mail: gilbert.ritschard@unige.ch

tion just a few of them. Now, depending on how data were collected, longitudinal and more generally sequential data may be organized in many different manners. On the other hand, when it comes to analysis, each software and method requires data to be inputted in some specific form. For instance, the mining of frequent sequential pattern is usually intended for event sequences, Markov models and edit-distance-based methods work on state sequences, while discrete survival models need data in person-period form. The end-user, and especially the end-user who wants to combine different types of analyses thus faces the difficult and often discouraging task of transforming his data in the right form.

The aim of this chapter is to help the analyst in this data preparation task. We propose a systematic description of the different ways sequential data can be organized, which should help to identify the nature of the data at hand. We then discuss issues raised by the transformation of one type of organization into another one. We explain why some of these transformations can be done straightforwardly in an automatic way, while others may require the user to define some rules to ensure that the outcome best suits her/his needs. We describe the automatic and semi-automatic procedures for switching from one type of organization to the other.

Our primary interest is in sequential data describing life courses, that is in sequences with order determined by the time. Hence, we shall indeed also pay attention to the time content, that is to the different ways of accounting for time, essentially calendars for defining time stamps and clocks for measuring time spans and spell durations. We adopt however on this aspect a practical standpoint as opposed to the logical definition of time concepts that can be found for instance in Hobbs and Pan [6].

This chapter is, as far as we know, the first attempt to present a general systematic view of the different ways of organizing and reorganizing time sequenced data. Karweit and Kertzer [7] discussed some aspects of the organization and conceptualization of life course data, but they mainly focused on data storage and access issues and the characterization of the units of analysis in terms of case and time. Sequence data organization issues have indeed been considered in the literature, but most often for a specific task only as Zaki [11] who describes in details an efficient way of organizing data for mining frequent sequence patterns or Blossfeld et al. [2] who discuss data organization for event history analysis.

The remainder of the chapter is organized as follows. In section 2 we define the different kinds of discrete sequences. A comprehensive list of sequence formats is presented in Section 3 and Section 4 is devoted to the handling of missing values. Then, in Section 5 we discuss the conversion between formats proposing among others rough basic solutions for automatically converting between state and event sequence data. Section 6 shortly comments on the implementation of the proposed solutions in our TraMineR package for R and Section 7 presents a few concluding remarks.

2 Sequence concepts

We consider sequences of discrete or categorical data. Formally, we define thus a sequence of length k as an ordered list of k elements successively chosen from a finite set A . The set A is called the *alphabet*. A natural representation of a sequence x is by listing the successive elements that form the sequence, for example as a sorted list $x = (x_1, x_2, \dots, x_k)$, with $x_j \in A$. When there is no ambiguity, we can just concatenate the successive values into a string, $x = x_1x_2 \dots x_k$. A separator would indeed be necessary when the alphabet includes any non-single symbol, which happens if we use for instance M for *married* and MC for *married with a child*.

Now, the nature of the sequence, that is its information content, depends on

1. what the position j in the sequence refers to;
2. the nature of the elements that compose the alphabet.

Regarding the position of each element in a sequence, it is important to distinguish between sequences with a time dimension and those without any reference to time. An occupational trajectory, a buyer’s history, or a record of device control signals typically contain chronologically sorted elements and, hence, have a time dimension. On the other hand, the order in texts and DNA sequences does not refer to time. In the latter case, i indicates simply the rank position, while j may bear more information when time matters. For instance, when data are collected at periodic dates as with panel data, the positions correspond to pre-specified dates (or periods). In that case, the position j informs about the date and a difference between positions can be interpreted as a duration.

Concerning the second point, namely the nature of the elements in the alphabet, an important distinction to make is whether the elements are states or whether they represent events. A *state* lasts as long as nothing happens, i.e. during some interval time, while an event happens at a given time point and may cause a change of state. For instance, consider a device turned on at 9 for 3 hours, and turned off after that. We may either report the sequence of states at each hour, e.g., “off at 8, on at 9, on at 10, on at 11, off at 12”, or alternatively report the sequence of on-off events, that is “turns on at 9, turns off at 12”. This can indeed also be done for non-chronological sequences such as the sequence of nucleotides “AGGC”. Here we could say that we start with ‘A’, switch to ‘G’ at position 2 and then to ‘C’ at position 4.

Table 1 Types of sequences

Alphabet	Time Dimension	
	No	Yes
States or objects	sequence of labels	state sequence
Events or transitions	sequence of transitions between labels	event sequence

This preliminary discussion leads us to distinguish the four types of sequences depicted in Table 1. Notice that though an event can just be a transition between two successive states of a chronological sequence, a transition such as from ‘single’ to

‘married with a child’ for example, may be the result of more than one event, namely here ‘marriage’ and ‘childbirth’. We discuss this issue in more details in Section 5.

For sequences with a time dimension, it is important to preserve the time information in any attempt to change the sequence representation. Hence it is essential to know the kind of time information the sequence holds. There exist different concepts of time: instant time (‘I started a new job the 1st of December’), time interval (‘I had a job during the whole last year’), absolute time (birth date), relative time (age). For instance, assume we face a sequence of annual occupational statuses such as (full time, full time, unemployed, ...). What does ‘unemployed in 2008’ mean? Does it mean that the concerned person was continuously unemployed during the whole year (interval time), experienced unemployment in 2008 (interval time), or that he was unemployed at the time of observation say in December 2008 (instant time)? In the first case, the state change at the beginning of a sequence of unemployment states clearly corresponds to a ‘falling in unemployment’ event, while in the two latter situations, there could be alternating employed-unemployed sequences during the same spell. Time granularity is also an issue, Data collected with a year granularity are hardly comparable with monthly based sequences. Turning data into

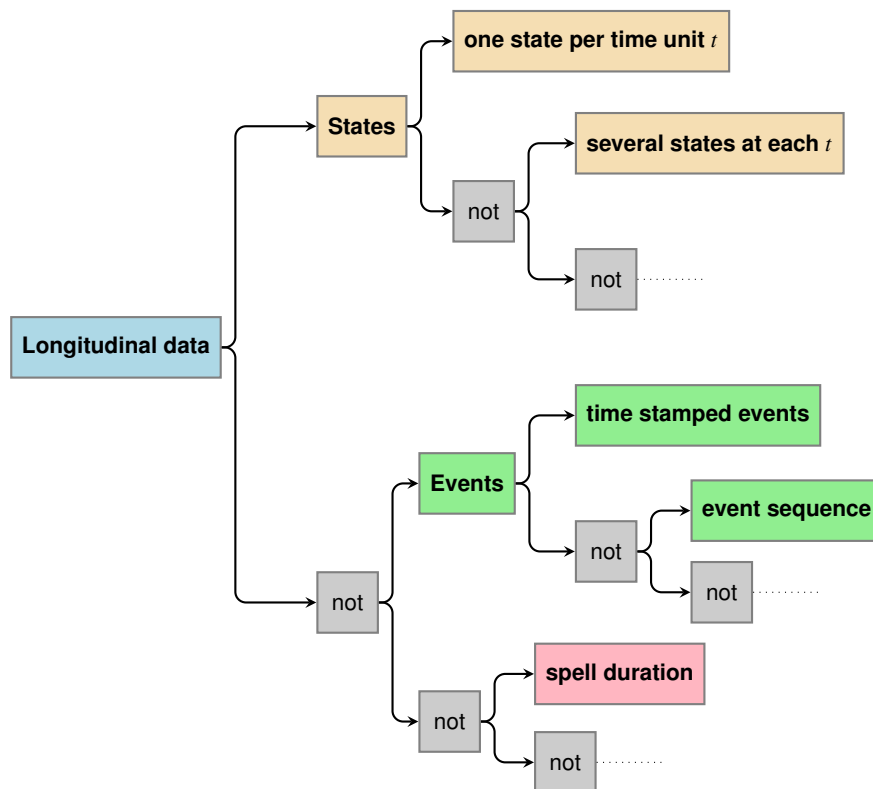


Fig. 1 Ontology of types of longitudinal data

finer granularity can only be done through rough approximations such as by assuming that the state reported for the year remains valid for all months, while the reverse raises time aggregation issues such as how can I transform monthly sequences into yearly sequences?

Another important point for characterizing a sequence is whether or not it can admit multiple elements at a same position. This is clearly a concern for event sequences, where multiple events (e.g., leaving home and getting married) may occur at a same time. For state sequences, we would get multiple states at a same position when we deal simultaneously with different dimensions such as the cohabitational and occupational statuses, i.e., when we have non exclusive states. The latter kind of sequence is sometimes referred to as multichannel sequence [5]. The Aristotelean tree of Figure 1 can be seen as a tentative ontology of longitudinal data types, that is types of sequential data with time content.

3 Basic Sequence Formats

This Section discusses different ways of representing sequence data. We explicit in each case the nature of the represented sequence elements as well as how time is accounted for when it matters.

3.1 Horizontal sequence organizations

We consider first representations in which sequences are organized in row (record) form, that is with the position in the sequence determined by a column index.

State sequence (STS)

We defined a sequence as an ordered list of k elements. A natural way of representing this list is as a (row) vector of k elements. For instance, by considering the states single, S , married, M , married with a child MC and divorced D , a sequence of length 10 would look as

$$(S, S, S, M, M, MC, MC, MC, MC, D) . \tag{1}$$

Using this representation, a set of n sequences would be stored in a $n \times k$ matrix in which each column j collects the states at position j for all cases.

We get a somewhat more compact form of exactly the same information by concatenating the elements into a single character string using for instance the ‘-’ separator

$$“S-S-S-M-M-MC-MC-MC-MC-D” .$$

This concatenated form should be more economical from the storage space standpoint since it requires a single character variable.

Time information is here accounted for by assigning absolute — date — or relative — age, process duration — times to each position. For our example, assuming that the sequence reports yearly states between ages 18 to 27 years, we assign age labels to the position indexes:

Age	18	19	20	21	22	23	24	25	26	27	
State	<i>S</i>	<i>S</i>	<i>S</i>	<i>M</i>	<i>M</i>	<i>MC</i>	<i>MC</i>	<i>MC</i>	<i>MC</i>	<i>D</i>	.

Notice that when we have a set of sequences, the time labeling of the position index needs just to be done once for the whole set of sequences, which is also economical in terms of required storage space.

Distinct-state, State-permanence and State-start sequence

In our previous example, we observe the same state in consecutive positions. We have for instance *S* in the three first positions. This suggests a simplified presentation in which we cite only one of several same consecutive states. This *distinct-state-sequence* (DSS) representation of our sequence is

$$(S, M, MC, D) .$$

It preserves the state sequencing, but clearly all time and more generally alignment information is lost.

We can, however, easily reintroduce it by assigning a time or duration stamp to each element of a DSS sequence. Aassve et al. [1], for instance, stamp each state with the number of times it is repeated and call the resulting format *State-permanence-sequence*. We denote it as SPS. For our example, the SPS form is:

$$((S,3) (M,2) (MC,4) (D,1)) .$$

We can indeed use any other notation for representing the (state, duration) couples, such as $(S/3, M/2, MC/4, D/1)$.

An alternative possibility, that we call *state-start-sequence* (SSS), consists in stamping states with start time instead of duration.

$$((S, 18) (M, 21) (MC, 23) (D, 27)) .$$

Notice that strictly speaking SPS and SSS formats do not reproduce exactly the content of the corresponding STS form. With SPS data, we would need also the start time of the sequence, while for data in the SSS form we should specify either the end time or the duration of the last state.

Table 2 Sequence data representations

Code	Data type	(S)tates or (E)vents	Several rows for a same case	Usage examples
STS	State-sequence	S	No	Markov modeling, OM
SPS	State-permanence	S	No	Markov modeling, OM
SSS	State-start	S	No	Markov modeling, OM
SRS	Shifted-replicated-sequence	S	Yes	Mobility tree
DSS	Distinct-state-sequence	S	No	OM without time reference
SPELL	Spell	S	Yes	Survival analysis
PPER	Person-period	S	Yes	Discrete survival analysis
FCE	Fixed-column-event	E	No	Survival analysis
HTSE	Horizontal time-stamped-event	E	No	Event sequence mining
TSE	Vertical time-stamped-event	E	Yes	Event sequence mining

OM stands for optimal matching and other analyses based on dissimilarities between pairs of sequences.

Shifted-replicated-sequence

This data presentation is intended for analyses where the concern is the transition from the states observed at previous time points, $t - 1, t - 2, \dots$, to the one observed at time t . Consider for example the sequence A, A, C, D, D where the first element in the sequence corresponds to year 2000 and the last one to year 2004, that is

$$\begin{array}{cccccc} 2000 & 2001 & 2002 & 2003 & 2004 & \\ \hline A & A & C & D & D & . \end{array}$$

The *shifted-replicated-sequence* representation of this sequence is obtained by repeating each sequence $k - 2$ times, shifting it each time one step on the right and dropping at each i th step the i right most elements out:

$$\begin{array}{cccccc} t-4 & t-3 & t-2 & t-1 & t & \\ \hline A & A & C & D & D & \\ . & A & A & C & D & \\ . & . & A & A & C & \\ . & . & . & A & A & . \end{array}$$

Finally, we relabel the columns with the relative time labels $t - k + 1$ to t .

In this SRS form we collect for instance in the columns named ‘ $t - 1$ ’ and ‘ t ’ all consecutive subsequences of length two, and hence all observed transitions between two successive positions. The column $t - i$ gives the state found i positions before the state reported in the last column t . The column reference is no longer a given date or age, but is relative. This organization of the data is for example required for growing mobility trees [9].

Table 3 Sequence data representations: Examples (Code explanation in Table 2)

Code	Example																																																																																																			
STS	<table border="0"> <tr> <td><i>Id</i></td> <td>18</td> <td>19</td> <td>20</td> <td>21</td> <td>22</td> <td>23</td> <td>24</td> <td>25</td> <td>26</td> <td>27</td> </tr> <tr> <td>101</td> <td>S</td> <td>S</td> <td>S</td> <td>M</td> <td>M</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>D</td> </tr> <tr> <td>102</td> <td>S</td> <td>S</td> <td>S</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> </tr> </table>	<i>Id</i>	18	19	20	21	22	23	24	25	26	27	101	S	S	S	M	M	MC	MC	MC	MC	D	102	S	S	S	MC	MC	MC	MC	MC	MC	MC																																																																		
<i>Id</i>	18	19	20	21	22	23	24	25	26	27																																																																																										
101	S	S	S	M	M	MC	MC	MC	MC	D																																																																																										
102	S	S	S	MC	MC	MC	MC	MC	MC	MC																																																																																										
SPS	<table border="0"> <tr> <td><i>Id</i></td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>101</td> <td>(S,3)</td> <td>(M,2)</td> <td>(MC,4)</td> <td>(D,1)</td> </tr> <tr> <td>102</td> <td>(S,3)</td> <td>(MC,7)</td> <td></td> <td></td> </tr> </table>	<i>Id</i>	1	2	3	4	101	(S,3)	(M,2)	(MC,4)	(D,1)	102	(S,3)	(MC,7)																																																																																						
<i>Id</i>	1	2	3	4																																																																																																
101	(S,3)	(M,2)	(MC,4)	(D,1)																																																																																																
102	(S,3)	(MC,7)																																																																																																		
SSS	<table border="0"> <tr> <td><i>Id</i></td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>101</td> <td>(S,18)</td> <td>(M,21)</td> <td>(MC,23)</td> <td>(D,27)</td> </tr> <tr> <td>102</td> <td>(S,18)</td> <td>(MC,21)</td> <td></td> <td></td> </tr> </table>	<i>Id</i>	1	2	3	4	101	(S,18)	(M,21)	(MC,23)	(D,27)	102	(S,18)	(MC,21)																																																																																						
<i>Id</i>	1	2	3	4																																																																																																
101	(S,18)	(M,21)	(MC,23)	(D,27)																																																																																																
102	(S,18)	(MC,21)																																																																																																		
SRS	<table border="0"> <tr> <td><i>Id</i></td> <td><i>t-9</i></td> <td><i>t-8</i></td> <td><i>t-7</i></td> <td><i>t-6</i></td> <td><i>t-5</i></td> <td><i>t-4</i></td> <td><i>t-3</i></td> <td><i>t-2</i></td> <td><i>t-1</i></td> <td><i>t</i></td> </tr> <tr> <td>101</td> <td>S</td> <td>S</td> <td>S</td> <td>M</td> <td>M</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>D</td> </tr> <tr> <td>101</td> <td>.</td> <td>S</td> <td>S</td> <td>S</td> <td>M</td> <td>M</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> </tr> <tr> <td>101</td> <td>.</td> <td>.</td> <td>S</td> <td>S</td> <td>S</td> <td>M</td> <td>M</td> <td>MC</td> <td>MC</td> <td>MC</td> </tr> <tr> <td>⋮</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>101</td> <td>.</td> <td>.</td> <td>.</td> <td>.</td> <td>.</td> <td>.</td> <td>.</td> <td>.</td> <td>S</td> <td>S</td> </tr> <tr> <td>102</td> <td>S</td> <td>S</td> <td>S</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> </tr> <tr> <td>102</td> <td>.</td> <td>S</td> <td>S</td> <td>S</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> <td>MC</td> </tr> <tr> <td>⋮</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	<i>Id</i>	<i>t-9</i>	<i>t-8</i>	<i>t-7</i>	<i>t-6</i>	<i>t-5</i>	<i>t-4</i>	<i>t-3</i>	<i>t-2</i>	<i>t-1</i>	<i>t</i>	101	S	S	S	M	M	MC	MC	MC	MC	D	101	.	S	S	S	M	M	MC	MC	MC	MC	101	.	.	S	S	S	M	M	MC	MC	MC	⋮											101	S	S	102	S	S	S	MC	MC	MC	MC	MC	MC	MC	102	.	S	S	S	MC	MC	MC	MC	MC	MC	⋮										
<i>Id</i>	<i>t-9</i>	<i>t-8</i>	<i>t-7</i>	<i>t-6</i>	<i>t-5</i>	<i>t-4</i>	<i>t-3</i>	<i>t-2</i>	<i>t-1</i>	<i>t</i>																																																																																										
101	S	S	S	M	M	MC	MC	MC	MC	D																																																																																										
101	.	S	S	S	M	M	MC	MC	MC	MC																																																																																										
101	.	.	S	S	S	M	M	MC	MC	MC																																																																																										
⋮																																																																																																				
101	S	S																																																																																										
102	S	S	S	MC	MC	MC	MC	MC	MC	MC																																																																																										
102	.	S	S	S	MC	MC	MC	MC	MC	MC																																																																																										
⋮																																																																																																				
DSS	<table border="0"> <tr> <td><i>Id</i></td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>101</td> <td>S</td> <td>M</td> <td>MC</td> <td>D</td> </tr> <tr> <td>102</td> <td>S</td> <td>MC</td> <td></td> <td></td> </tr> </table>	<i>Id</i>	1	2	3	4	101	S	M	MC	D	102	S	MC																																																																																						
<i>Id</i>	1	2	3	4																																																																																																
101	S	M	MC	D																																																																																																
102	S	MC																																																																																																		
SPELL	<table border="0"> <tr> <td><i>Id</i></td> <td><i>Index</i></td> <td><i>From</i></td> <td><i>To</i></td> <td><i>State</i></td> </tr> <tr> <td>101</td> <td>1</td> <td>18</td> <td>20</td> <td>Single (S)</td> </tr> <tr> <td>101</td> <td>2</td> <td>21</td> <td>22</td> <td>Married (M)</td> </tr> <tr> <td>101</td> <td>3</td> <td>23</td> <td>26</td> <td>Married w Children (MC)</td> </tr> <tr> <td>101</td> <td>4</td> <td>27</td> <td>27</td> <td>Divorced (D)</td> </tr> <tr> <td>102</td> <td>1</td> <td>18</td> <td>20</td> <td>Single (S)</td> </tr> <tr> <td>102</td> <td>2</td> <td>21</td> <td>27</td> <td>Married w Children (MC)</td> </tr> </table>	<i>Id</i>	<i>Index</i>	<i>From</i>	<i>To</i>	<i>State</i>	101	1	18	20	Single (S)	101	2	21	22	Married (M)	101	3	23	26	Married w Children (MC)	101	4	27	27	Divorced (D)	102	1	18	20	Single (S)	102	2	21	27	Married w Children (MC)																																																																
<i>Id</i>	<i>Index</i>	<i>From</i>	<i>To</i>	<i>State</i>																																																																																																
101	1	18	20	Single (S)																																																																																																
101	2	21	22	Married (M)																																																																																																
101	3	23	26	Married w Children (MC)																																																																																																
101	4	27	27	Divorced (D)																																																																																																
102	1	18	20	Single (S)																																																																																																
102	2	21	27	Married w Children (MC)																																																																																																
PPER	<table border="0"> <tr> <td><i>Id</i></td> <td><i>Index</i></td> <td><i>Age</i></td> <td><i>State</i></td> </tr> <tr> <td>101</td> <td>1</td> <td>18</td> <td>Single (S)</td> </tr> <tr> <td>101</td> <td>2</td> <td>19</td> <td>Single (S)</td> </tr> <tr> <td>101</td> <td>3</td> <td>20</td> <td>Single (S)</td> </tr> <tr> <td>101</td> <td>4</td> <td>21</td> <td>Married (M)</td> </tr> <tr> <td>⋮</td> <td>⋮</td> <td>⋮</td> <td></td> </tr> <tr> <td>101</td> <td>10</td> <td>27</td> <td>Divorced (D)</td> </tr> <tr> <td>102</td> <td>1</td> <td>18</td> <td>Single (S)</td> </tr> <tr> <td>⋮</td> <td>⋮</td> <td>⋮</td> <td></td> </tr> </table>	<i>Id</i>	<i>Index</i>	<i>Age</i>	<i>State</i>	101	1	18	Single (S)	101	2	19	Single (S)	101	3	20	Single (S)	101	4	21	Married (M)	⋮	⋮	⋮		101	10	27	Divorced (D)	102	1	18	Single (S)	⋮	⋮	⋮																																																																
<i>Id</i>	<i>Index</i>	<i>Age</i>	<i>State</i>																																																																																																	
101	1	18	Single (S)																																																																																																	
101	2	19	Single (S)																																																																																																	
101	3	20	Single (S)																																																																																																	
101	4	21	Married (M)																																																																																																	
⋮	⋮	⋮																																																																																																		
101	10	27	Divorced (D)																																																																																																	
102	1	18	Single (S)																																																																																																	
⋮	⋮	⋮																																																																																																		
FCE	<table border="0"> <tr> <td><i>Id</i></td> <td><i>#marr.</i></td> <td><i>1st marr.</i></td> <td><i>2nd marr.</i></td> <td><i>...</i></td> <td><i>#child.</i></td> <td><i>1st child</i></td> <td><i>2nd child</i></td> <td><i>...</i></td> </tr> <tr> <td>101</td> <td>1</td> <td>21</td> <td>.</td> <td>.</td> <td>2</td> <td>23</td> <td>26</td> <td>.</td> </tr> <tr> <td>102</td> <td>1</td> <td>21</td> <td>.</td> <td>.</td> <td>1</td> <td>21</td> <td>.</td> <td>.</td> </tr> </table>	<i>Id</i>	<i>#marr.</i>	<i>1st marr.</i>	<i>2nd marr.</i>	<i>...</i>	<i>#child.</i>	<i>1st child</i>	<i>2nd child</i>	<i>...</i>	101	1	21	.	.	2	23	26	.	102	1	21	.	.	1	21	.	.																																																																								
<i>Id</i>	<i>#marr.</i>	<i>1st marr.</i>	<i>2nd marr.</i>	<i>...</i>	<i>#child.</i>	<i>1st child</i>	<i>2nd child</i>	<i>...</i>																																																																																												
101	1	21	.	.	2	23	26	.																																																																																												
102	1	21	.	.	1	21	.	.																																																																																												
HTSE	<table border="0"> <tr> <td><i>Id</i></td> <td>1</td> <td>2</td> <td>3</td> <td>...</td> </tr> <tr> <td>101</td> <td>(marriage, 21)</td> <td>(childbirth, 23)</td> <td>(childbirth, 26)</td> <td>(divorce, 27)</td> </tr> <tr> <td>102</td> <td>(marriage, 21)</td> <td>(childbirth, 21)</td> <td></td> <td></td> </tr> </table>	<i>Id</i>	1	2	3	...	101	(marriage, 21)	(childbirth, 23)	(childbirth, 26)	(divorce, 27)	102	(marriage, 21)	(childbirth, 21)																																																																																						
<i>Id</i>	1	2	3	...																																																																																																
101	(marriage, 21)	(childbirth, 23)	(childbirth, 26)	(divorce, 27)																																																																																																
102	(marriage, 21)	(childbirth, 21)																																																																																																		
TSE	<table border="0"> <tr> <td><i>Id</i></td> <td><i>Time</i></td> <td><i>Event</i></td> </tr> <tr> <td>101</td> <td>21</td> <td>Marriage</td> </tr> <tr> <td>101</td> <td>23</td> <td>Childbirth</td> </tr> <tr> <td>101</td> <td>26</td> <td>Childbirth</td> </tr> <tr> <td>101</td> <td>27</td> <td>Divorce</td> </tr> <tr> <td>102</td> <td>21</td> <td>Marriage</td> </tr> <tr> <td>102</td> <td>21</td> <td>Childbirth</td> </tr> </table>	<i>Id</i>	<i>Time</i>	<i>Event</i>	101	21	Marriage	101	23	Childbirth	101	26	Childbirth	101	27	Divorce	102	21	Marriage	102	21	Childbirth																																																																														
<i>Id</i>	<i>Time</i>	<i>Event</i>																																																																																																		
101	21	Marriage																																																																																																		
101	23	Childbirth																																																																																																		
101	26	Childbirth																																																																																																		
101	27	Divorce																																																																																																		
102	21	Marriage																																																																																																		
102	21	Childbirth																																																																																																		

3.2 Vertical sequence organizations

We now consider representations in which the elements of the sequence are given in successive rows. Such data representation is for instance especially useful for discrete time survival analysis [10].

Person-period data

The *Person-period* (PPER) data form is obtained by defining a separate record for each period lived by each individual. The person-period representation of a single sequence can be seen as the transpose of its STS form. In PPER form however, the set of sequences is arranged in a single column by laying the sequences on top of each other. One advantage of this representation is that it allows to handle time varying covariates in a straightforward manner by simply completing the data with columns giving the values of the covariates for each considered time (row). A second advantage is that, unlike the STS form, it does not require all sequences to have the same start and end times. Periods not observed for a given case can be simply omitted. The price to pay for these advantages, especially the last one, is that in the PPER format the concerned period must be explicitly specified for each record. Here is the PPER format of our earlier example (1):

<i>Id</i>	<i>Index Age</i>		<i>State</i>
101	1	18	Single (S)
101	2	19	Single (S)
101	3	20	Single (S)
101	4	21	Married (M)
101	5	22	Married (M)
101	6	23	Married w Child (MC)
101	7	24	Married w Child (MC)
101	8	25	Married w Child (MC)
101	9	26	Married w Child (MC)
101	10	27	Divorced (D)

Spell data

The *Spell* data (SPELL) organization is a compacted person-format form that uses a single record for representing successive periods with unchanged state. For a given sequence, it can be seen as the transpose of either the SPS (state-permanence) or SSS (state-start) format. As with the PPER format, however, in Spell form the sequences are stacked and not laid one beneath each other. Each record should indeed specify either the start and end times of the spell, or equivalently its start time and duration. Notice that if we can assume, what is not too restrictive, that each spell ends when the next one starts, then it would be sufficient to give the start time only (or the duration only) of each spell. The SPELL format of example (1) looks as follows

<i>Id</i>	<i>Index From To</i>			<i>State</i>
101	1	18	20	Single (S)
101	2	21	22	Married (M)
101	3	23	26	Married w Children (MC)
101	4	27	27	Divorced (D)

3.3 Event sequences

The format discussed so far are primarily intended for state sequences. States are supposed to last and are naturally associated with interval time. Here we consider events, that is phenomenons that occur at given time-points and do not last. For instance, starting a new job, getting married and switching a device off are events. They may result in a lasting new state, but events do not persist themselves. Hence, it is in time reference that the representation of a sequence of events will differ from state sequences.

As long as we are only interested in the sequencing of the events, we may rely on STS like or DSS representations. State-permanence has indeed no sense for non lasting events. Likewise, spell representation are not suited for event data. The most common way of representing event sequences is as *time-stamped-event* either horizontally or vertically.

Horizontal time stamped events

There are two possibilities for presenting *time-stamped-event* data horizontally. The first is similar to the STS form, with the dates at which events occur as column headings. This may be justified when events occur at the same regular dates for each case. Most often, however, a sequence of time stamped events is represented by a sequence of (event, time stamp) couples. We call this format *horizontal-time-stamped-event* (HTSE). For instance, if we consider the events defined by the state transitions of our state sequence example plus a second childbirth at 26, we would write down the data as

$$((\text{starts as single}, 18) (\text{marriage}, 21) (\text{childbirth}, 23) (\text{childbirth}, 26) (\text{divorce}, 27)) . \quad (2)$$

Notice that it is most often necessary to specify the state at the start of the observed period if we want to retrieve the whole state information from the sole knowledge of the events.

When events are repeatable, such as “starting a new job”, “childbirth” or “turning a device on”, it may be useful to know the rank of the event. In such cases, a some-time more convenient representation consists in grouping the events and reporting the number of events of a certain type, let us say the number of childbirths, and then list the successive time stamps of the events in fixed columns. We call this form *fixed-column-event* (FCE). For instance, the childbirth information of our previous

example would look as follows in FCE format

<i>id</i>	<i>Number of childbirths</i>	<i>Age at 1st childbirth</i>	<i>Age at 2nd childbirth</i>	<i>Age at 3rd childbirth</i>	<i>...</i>
101	2	23	26	NA	...

Biographical data bases are often presented this way with for instance dates of changes in living arrangement, marital status, number of children, education and professional careers. It is convenient for survival methods such as for instance the estimation of a Kaplan-Meier curve, since it permits to easily compute the required duration from a start event until the event of interest. The disadvantage is that it may result in a very scarce table with plenty of empty entries.

Vertical time stamped events

As for state sequences, event sequences can indeed also be organized vertically by reporting each (event, time stamp) couple in a new line. We designate this *vertical-time-stamped-event* simply as TSE. Here is how our example looks out in this TSE format

<i>id</i>	<i>index</i>	<i>time</i>	<i>event</i>
101	1	18	Start as single
101	2	21	Marriage
101	3	23	Childbirth
101	4	26	Childbirth
101	5	27	Divorce

In this format, two simultaneous events, that is events with same time stamp such as (Marriage, 25) and (Childbirth, 25) would be represented by two lines. A variant sometime considered consists in giving the time stamp with the list of events in a same single line

<i>id</i>	<i>index</i>	<i>time</i>	<i>event</i>
102	1 – 2	25	Marriage, Childbirth

4 Missing values

Before discussing how we can convert between formats, a few words are worth about how to record missing elements in sequences. Depending on where they appear in sequence, we distinguish the following types of missing values:

- *left-missing-values*, that is missing values appearing before the first valid entry in the sequence;
- *inner-missing-values* or *gaps*, that is missing values appearing somewhere between the first and last valid entries;
- *right-missing values*, that is missing values appearing after the last valid entry.

The distinction is important for time referenced sequences, where each type may result from different reasons and may require different handling. For instance, *left-missing-values* may occur with sequences that do not start at the same time, *right-missing-values* when sequences are of different lengths, and gaps when we missed the observation at some date. In the first case we may want to align the beginning of the sequences, preferring for instance an age or process time to a calendar time. On the other hand, right-missing-values corresponding to truncation could be simply ignored, while for gaps the treatment may depend on whether or not it is important to preserve the time alignment across sequences. These are indeed just examples, the specific treatment of each type of missing values will indeed depend of whether the analysis method we envisage to use supports missing values, and if yes which kind it supports. For instance, a missing element will have less dramatic consequences for running methods for event sequences than methods for state sequences.

Beside radical list wise deletion solutions, basic handling of missing values include:

- deleting them from the sequence, which implies shifting one position to the left all elements appearing on the right side of the missing value;
- maintaining missing values at their place, and using special treatments for them during the analysis stage;
- completing the alphabet with a “missing” term, i.e., treat missing values as if they were normal elements of the sequence;
- missing data imputation using for instance techniques for microarrays [3] or for repeated measures [8].

It is behind the scope of this chapter to discuss the handling of missing values for specific types of analysis. Nevertheless, it is important to have the distinctions made above into the mind when we want convert data into a new format.

5 Transforming between Formats

When converting from one format into the other we usually want to preserve the information. This should not be a problem when turning a state sequence format into another state sequence format, as well as when converting between event sequence formats. The transformation of states into events and vice-versa is more complicated and requires additional information from the user. This section addresses some of the issues raised by format conversions.

5.1 Converting between state sequence formats

Conversion between STS (state-sequence), SPS (state-permanence), SSS (state-start) and SRS (shifted-replicated) horizontal formats of state sequences is straight-

Table 4 Feasible automatic and semi-automatic conversions

From/To	STS	SPS	SRS	PPER	SPELL	DSS	TSE	HTSE	FCE
STS	.	A	A	A	A	A	A/U	A/U	A/U
SPS	A	.	A	A	A	A	A/U	A/U	A/U
SRS	A	A	.	A	A	A	A/U	A/U	A/U
PPER	A	A	A	.	A	A	A/U	A/U	A/U
SPELL	A	A	A	A	.	A	A/U	A/U	A/U
DSS	N	N	N	N	N	.	N	N	N
TSE	A/U	A/U	A/U	A/U	A/U	?	.	A	A
HTSE	A/U	A/U	A/U	A/U	A/U	?	A	.	A
FCE	A/U	A/U	A/U	A/U	A/U	?	A	A	.

A: automatic, U: needs user intervention, N: not possible
 A/U: automatic under some conditions, otherwise needs user intervention

forward as shown from the examples in Table 3. Such conversions can easily been automatized with a few lines of code and functions doing the job are for instance proposed in our TraMineR package [4]. However, to get exactly the same information, that is to make the transformation reversible, we should store separately the start time with both SRS and duration stamped SPS formats, and either the duration or end time of the last spell with the SSS format. For instance, from STS to SRS we just repeatedly replicate and shift each sequence. The relabeling of the columns with the appropriate lag length needs no further information. For the converse, that is from SRS to STS, we just have to retain the left most aligned sequence for each case. The relabeling of the columns with time stamps requires however here the externally stored start time of each sequence.

Retained missing values are dealt with as other state values and need no special attention. Conversion can be done as well when there are dropped out missing values. In case of dropped out gaps, however, the lags used in SRS and the duration stamps in SPS will lose their meaning. To be able to restore true time or duration stamps requires then to hold somewhere the original positions of these dropped out missing values. Information about left and right missing values is less important, except for the true observation start time when the duration since this start time matters.

Transforming from an horizontal to a vertical format is almost as straightforward. PPER and SPELL are the vertical equivalents of respectively the STS and SPS forms. Be aware, however, that left and right missing values are typically ignored in vertical formats, i.e. they are dropped out. Hence, when we want explicitly account for the existence of such left and right missing values, the information should be held separately.

The distinct-states format DSS is indeed the SPS form without the time stamp information. Hence it can automatically be obtained from any of STS, SPS, SRS, PPER or SPELL format. The transformation is clearly not reversible however since DSS holds no time stamped information.

5.2 *Converting between event sequence formats*

Conversion between event sequence formats, that is between FCE, HTSE and TSE can also be automatized. The FCE (fixed-column-event) form requires either to determine in advance the maximal number of each kind of events known by each subject (number of marriages, number of childbirths, etc.), or to be able to insert columns when necessary. Conversion to HTSE and TSE has no such requirement and is therefore easier to implement.

In any of the event sequence formats, it may be useful, especially for a later transformation into a state sequence format, to consider a “start of observation” event together with the original state of the subject at this start time.

5.3 *Conversion from state to event sequences*

Transforming state data into event data as well as the converse, that is event data into state data is less straightforward. It is easy to derive a sequence of transitions between states from a state sequence and reciprocally states from the transitions between them. Transitions, however, are not equivalent to events. Let us first clarify the distinction between them.

We define a *transition* as the change between two consecutive states in the sequence. This definition holds whether or not the sequence includes time information. An *event*, on the other hand, is something that happens at a given time point and hence makes sense for chronological sequences only. Though a transition in a chronological sequence can be considered as an event, the two concepts are not equivalent. The event ‘Marriage’ for instance characterizes the transition $S \rightarrow M$, but participates also to the characterization of the transition $S \rightarrow MC$, that is ‘from ‘single’ to ‘married with a child’’. Assuming we have yearly data, the latter transition results when both events ‘Marriage’ and ‘Childbirth’ happen the same year, and hence requires that the event ‘Marriage’ occurs. This example illustrates also that a transition may reflect the conjunction of several events. Another example is the transition $S \rightarrow D$ (single to divorced) which makes only sense when divorce follows a marriage within the same year.

Converting state sequences into event sequences requires to specify the mapping between transitions and events, that is to specify the events that must necessarily happen to generate each given transition. This can be formalized by a *transition-definition matrix* where we give the ‘from’ states as row labels, the ‘to’ states as column labels, and list the joint events that characterize each transition in the corresponding cell.

Table 5 shows how this matrix could look out for the transitions between the states considered in sequence example (1). We would expect also the conversion process to account for the state in which the sequence starts. This requires to assign one of the events ‘starts as S ’, ‘starts as M ’, ‘starts as MC ’ or ‘starts as D ’ to the beginning of the sequence. For convenience, we could just insert these start events

Table 5 Example of a transition-definition matrix for state sequence (1)

From\To	<i>S</i>	<i>M</i>	<i>MC</i>	<i>D</i>
<i>S</i>	<i>starts as S</i>	Marriage	Marriage, Childbirth	Marriage, Divorce
<i>M</i>	<i>impossible</i>	<i>starts as M</i>	Childbirth	Divorce
<i>MC</i>	<i>impossible</i>	Child leaving	<i>starts as MC</i>	Divorce, Child leaving
<i>D</i>	<i>impossible</i>	Marriage	Marriage, Childbirth	<i>starts as D</i>

on the otherwise unused diagonal of the matrix. Using this transition-definition matrix we can then automatically convert state sequences by replacing all encountered transitions by the associated events and stamping them with the time at which the transition occurs. A convention must however be adopted for the time stamp depending on whether we assume the state reported for time unit t , say year t , is the state at the beginning of this observed time unit or at the end of it. In the first case, we would stamp events with the time of the ‘from’ state of the transition and otherwise with that of the ‘to’ state. Adopting this latter convention for converting our example sequence (1) we get the following event time stamped sequence

$$((\text{starts as } S, 18) (\text{marriage}, 21) (\text{childbirth}, 23) (\text{divorce}, 27)) \quad (3)$$

Notice that this sequence differs from that of example (2), which mentions an additional childbirth that cannot be accounted for with the four sole states considered. A state ‘MC2’, married with two children, would be necessary for that. This illustrates the tight relationships that should exist between states and events when we want to get state and event representations holding exactly the same information.

The designing of the transition-definition matrix belongs to the user, which prevents the conversion process to be fully automatized. As shown by the above small example, it may also be an awkward task. It may therefore be useful to be able to automatically generate some rough transition-definition matrix. Even when not completely relevant, such a matrix could serve as a start point for the design process. It could also be used as is for applying quickly tentatively methods of event sequence analyses on data presented in state sequence formats. We propose hereafter two such rough automatic methods.

A first method that we call ‘Transition’, consists in considering each observed transition as a simple event. The transition-definition matrix generated from our example sequence is given in Table 6.

The second method named ‘End-Begin’ assigns two events to each transition, namely the end of the ‘from’ state and the beginning of the ‘to’ state. We use the prefixes ‘end_’ and ‘bgn_’ to denote these events in Table 7 that reports the matrix obtained this way from our example.

The diagonal terms of the matrices should indeed not be interpreted as the other entries. They do not stand for the transition of the corresponding state to itself, but indicate the event that initiates sequences starting in the corresponding column state. Remember also that the automatically generated transition-definition matrices

Table 6 Transition-definition matrix for state sequence (1) generated by the ‘Transition’ method.

From\To	S	M	MC	D
S	starts as S	$S \rightarrow M$	$S \rightarrow MC$	$S \rightarrow D$
M	$M \rightarrow S$	starts as M	$M \rightarrow MC$	$M \rightarrow D$
MC	$MC \rightarrow S$	$MC \rightarrow M$	starts as MC	$MC \rightarrow D$
D	$D \rightarrow S$	$D \rightarrow M$	$D \rightarrow MC$	starts as D

Table 7 Transition-definition matrix for state sequence (1) generated by the ‘End-Begin’ method.

From\To	S	M	MC	D
S	bgn_ S	end_ S , bgn_ M	end_ S , bgn_ MC	end_ S , bgn_ D
M	end_ M , bgn_ S	bgn_ M	end_ M , bgn_ MC	end_ M , bgn_ D
MC	end_ MC , bgn_ S	end_ MC , bgn_ M	bgn_ MC	end_ MC , bgn_ D
D	end_ D , bgn_ S	end_ D , bgn_ M	end_ D , bgn_ MC	bgn_ D

are just rough solutions that will most often require adjustments to suite the user’s research objectives.

5.4 Conversion from event to state sequences

The reverse conversion from event to state sequences needs again a definition of transitions from the events. However, we face now a different problem, the goal being to find the a priori unknown states resulting from the successive known events. To do that, we assume that a state transition can only occur when an event happens and that the state at each time position t depends uniquely of the events that occurred before t , including indeed the sequence initiating event. Under these hypotheses the successive states can be determined recursively from the event sequence. The first state is defined by the initiating event, which means that we have to know the starting state of the sequence. This state is then replicated for each time until the time at which the next event happens. At this point we switch to the new state caused by the event. We then repeat the process until we get the state generated by the last event. When necessary, we can repeat the last state until a fixed sequence end time.

The only difficulty in implementing the process is the determination of the state in which we fall after each event. Note that, as in a Markov chain, the ancestor state at t summarizes all the information we need from the sequence of previous events. The new state can be determined from the joint knowledge of the event and this previous state. Thus, what we need is a *state-definition* matrix giving the resulting new state for each (previous state, event) couple. Table 8 shows one possible matrix for the events considered in sequences (2) and (3). To keep the example small, the design of this matrix assumes that we are only interested in the four states S , M , MC and D , i.e., we are not interested to distinguish among single (S) or divorced (D) people those who live with children from those who live without children, that is the ‘child’ distinction is supposed relevant only for married people.

Table 8 Example of a state-definition matrix for event sequences (2) and (3)

From\Event	Marriage	Childbirth	Divorce
<i>S</i>	<i>M</i>	<i>S</i>	<i>impossible</i>
<i>M</i>	<i>impossible</i>	<i>MC</i>	<i>D</i>
<i>D</i>	<i>M</i>	<i>D</i>	<i>impossible</i>
<i>MC</i>	<i>impossible</i>	<i>MC</i>	<i>D</i>

Once we have defined the state-definition matrix, we can proceed with converting the event sequence into a state sequence. At each event we switch to the state found at the intersection of the row corresponding to the ancestor state and the column associated with the event.

As for the state to event conversion, we may imagine some automatic process for generating a basic state-definition matrix. One possibility consists in defining the state by the set of experienced events without accounting for their order, i.e., by associating a state to each possible combination of events. For c different events we would thus generate 2^c possible states, including a *none* state that remains valid as long as no event is experienced. Table 9 shows the state-definition matrix automatically generated from the three events considered in sequences (2) and (3). The matrix contains a row for each of the 2^3 combinations of the three states. In each row, we read the state in which we fall when the corresponding column event happens.

Table 9 State-definition matrix for generated from the events in sequence (2) and (3)

From\Event	Marriage	Childbirth	Divorce
<i>none</i>	{marr}	{child}	{div}
{marr}	{marr}	{marr,child}	{marr,div}
{child}	{marr,child}	{child}	{child,div}
{div}	{marr,div}	{child,div}	{div}
{marr,child}	{marr,child}	{marr,child}	{marr,child,div}
{marr,div}	{marr,div}	{marr,div,child}	{marr,div}
{child,div}	{marr,child,div}	{child,div}	{child,div}
{marr,child,div}	{marr,child,div}	{marr,child,div}	{marr,child,div}

This automatically generated state-definition matrix is just a rough basic solution. It may be worth to make some adjustments before using it for making the conversion. First, it can happen that the automatic process generates some theoretically unattainable states. In our example, for instance, it makes no sense to consider states where we have divorced without getting married, which suggests to exclude the states {div} and {child,div}. Maintaining them would nevertheless have no consequences, since we should never fall in such unattainable states. Secondly, the number of automatically generated states, which raises exponentially with the number of events, may become too large for an efficient state sequence analysis. For example, with $c = 5$ events we get 32 states, and $c = 10$ leads to 1024 states. The user may then want to reduce the number of states by selecting only the more relevant of

them. A possible empirical solution — or at least an empirical guide line — could be here to consider only states that exceed some threshold frequency for the whole sequence data set. This would indeed also exclude unfeasible states.

An important limitation of the just described method is that a new state can only be obtained by augmenting the set of events that defines the ancestor state. This precludes any return to a previously visited state. We can overcome this limitation by combining the process with an event dropping out mechanism. We define such a mechanism by means of a binary $c \times c$ *event-drop-out-matrix* in which a 1 is set in cell (i, j) to indicate that event i should be dropped out when event j happens. For our example, we could define the matrices shown in Table 10. The left side matrix states that a divorce cancels a previous marriage, but also that any previous divorce will be ignored after a new marriage. In the right side matrix, we state in addition that we should forget about any preceding childbirth when a divorce happens.

Table 10 Two possible event-drop-out-matrices

Element to drop out	Occurring event			Element to drop out	Occurring event		
	Marriage	Childbirth	Divorce		Marriage	Childbirth	Divorce
marr	0	0	1	marr	0	0	1
child	0	0	0	child	0	0	1
div	1	0	0	div	1	0	0

Using the right side matrix for the drop out mechanism in the automatic design of the state-definition-matrix, we get Table 11. This matrix differs from the one we defined by hand in Table 8 in that it defines specific states for people that have a child while they are not married, namely states $\{\text{child}\}$ and $\{\text{child,div}\}$.

Table 11 State-definition matrix generated with a drop-out mechanism

From\Event	Marriage	Childbirth	Divorce
<i>none</i>	{marr}	{child}	{div}
{marr}	{marr}	{marr,child}	{div}
{child}	{marr,child}	{child}	{div}
{div}	{marr}	{child,div}	{div}
{marr,child}	{marr,child}	{marr,child}	{div}
{child,div}	{marr,child}	{child,div}	{div}

Similarly to the event-drop-out-mechanism, we can implement a ‘cancel event effect’ mechanism that would prevent any transition after events occurring in specified states. This requires to specify a binary $(c + 1) \times c$ *cancel-event-matrix* in which a 1 in cell (i, j) indicates that event j should be ignored when it occurs while we are in any state containing event i . Considering again our example, we would define the matrix as shown in Table 12 for ignoring childbirths experienced by unmarried people. Notice that we have here the *none* row for accounting for the state that prevails while no event occurs.

Table 12 Cancel-event-matrix for preventing transitions after childbirths for non married people

Element of state definition	Canceled event		
	Marriage	Childbirth	Divorce
<i>none</i>	0	1	0
marr	0	0	0
child	0	0	0
div	0	1	0

Generating the state-definition matrix with both the drop-out and cancel-event-effect mechanisms we get Table 13. If we relabel $S = none$, $M = marr$, $C = child$ and $D = div$, this matrix appears to be equivalent to our first state-definition matrix of Table 8 except for the cells labeled as *impossible*. The latter have, however, no importance since they correspond to states that will never be reached.

Table 13 State-definition matrix generated with drop-out and cancel-event mechanisms

From\Event	Marriage	Childbirth	Divorce
<i>none</i>	{marr}	<i>none</i>	{div}
{marr}	{marr}	{marr,child}	{div}
{div}	{marr}	{div}	{div}
{marr,child}	{marr,child}	{marr,child}	{div}

The last solutions proposed are not wholly automatic since they require the user to specify the event-drop-out and cancel-event matrices. In our experiences, the specification of these matrices was however much simpler than the complete design of the state-definition matrix and proved thus to be a valuable help in the conversion process.

6 Implementation in the R environment

Most of the sequence formats discussed in this chapter are already supported by our TraMineR package for rendering, mining and analysing sequence data in R [4]. The package offers functions that do the automatic conversion between either state sequence formats or between event sequence formats, as well as the conversion between state and event sequences from a user provided definition matrix. The building of rough basic transition-definition and state-definition matrices were also implemented in the latest — currently in testing stage — release of the package.

TraMineR uses state-sequence objects and event-sequence objects. The former store the data internally in STS form and the latter in TSE form. To avoid the multiplication of the conversion procedures, the conversion between any two state sequence formats, say SPS to SSS, is done by converting first to the internal STS and then to the destination format. Likewise, the conversion between formats of event se-

quences is done by passing through the TSE form. This remains indeed transparent for the user. The transformation between state and event sequences is implemented for a conversion between the default STS and TSE forms. As with other functions in TraMineR, a different input format can however be specified, in which case an automatic conversion into the default format is applied before the state-event transformation. Similarly, an option allows to further transform the output in any supported output format.

7 Conclusion

The aim of this chapter was to respond to an obvious lack in the literature of a general reference for all questions regarding the preparation of categorical sequence data. The comprehensive overview of the different ways of organizing discrete sequence data and of the possibilities to pass from one presentation to the other one makes this chapter unique. The overview was built on our experiences in the analysis of life trajectory data. The chapter presents thus also original data transformation solutions such as those adopted for converting state data into event sequences. The material assembled here should undoubtedly help others in preparing data for sequence analysis. At least it corresponds to what we would have liked to find when we started to work with sequence data.

The data organization strongly depends indeed on the nature of the sequence and it is therefore important to identify the kind of sequence data at hand. We have seen that an important distinction that should be done is between chronological sequences and sequences without time content. Behind positions and sequence lengths, the former hold indeed time information that we should care to preserve when manipulating and converting sequences. Then, for time stamped sequences, a second important distinction is between state and event sequence data. The conversion from one of these types into the other one may be awkward and the solutions proposed here constitute perhaps the most original part of the chapter. Now, though the overview looks complete we can imagine some further developments, for instance regarding the automatic detection of the data organization or in the designing of additional solutions for automatizing the conversion between states and events.

Acknowledgement

This work was realized within a research project entitled “Mining event histories: Towards new insights on personal Swiss life courses”. The authors are grateful to the Swiss National Foundation for scientific research who supported this project under grant SNF-100012-113998.

References

1. Aassve, A., Billari, F., Piccarreta, R. Strings of adulthood: A sequence analysis of young British women's work-family trajectories. *European Journal of Population*, 23(3):369–388, 2007. doi: 10.1007/s10680-007-9134-6.
2. Blossfeld, H.P., Golsch, K., Rohwer, G. *Event History Analysis with Stata*. Lawrence Erlbaum, Mahwah NJ, 2007.
3. Brock, G.N., Shaffer, J.R., Blakesley, R.E., Lotz, M.J., Tseng, G.C. Which missing value imputation method to use in expression profiles: A comparative study and two selection schemes. *BMC Bioinformatics*, 9:12, 2008. doi: 10.1186/1471-2105-9-12.
4. Gabadinho, A., Ritschard, G., Studer, M., Müller, N.S. Mining sequence data in R with TraMineR: A user's guide for version 1.1. Technical report, Department of Econometrics and Laboratory of Demography, University of Geneva, Geneva, 2009. URL <http://mephisto.unige.ch/traminer>.
5. Gauthier, J.A., Widmer, E.D., Bucher, P., Notredame, C. Multichannel sequence analysis applied to social science data. Manuscript, University of Lausanne, 2007. (Under review).
6. Hobbs, J.R., Pan, F. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing*, 3(1):66–85, 2004.
7. Karweit, N., Kertzer, D. Data organization and conceptualization. In Giele, J.Z., Elder, G.H., editors, *Methods of Life Course Research: Qualitative and Quantitative Approaches*, pages 81–97. Sage, 1998.
8. Little, R.J.A. Modeling the drop-out mechanism in repeated-measures studies. *Journal of the American Statistical Association*, 90(431):1112–1121, 1995. URL <http://www.jstor.org/stable/2291350>.
9. Ritschard, G., Oris, M. Life course data in demography and social sciences: Statistical and data mining approaches. In Levy, R., Ghisletta, P., Le Goff, J.M., Spini, D., Widmer, E., editors, *Towards an Interdisciplinary Perspective on the Life Course*, Advances in Life Course Research, Vol. 10, pages 289–320. Elsevier, Amsterdam, 2005.
10. Yamaguchi, K. *Event history analysis*. ASRM 28. Sage, Newbury Park and London, 1991.
11. Zaki, M.J. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.