Tools for Analyzing Log File Data

Ulf Kroehne & Frank Goldhammer

2020/06/18

Agenda

- Minimal Workflow: Pre-Processing (Log Data), Low-Level Feature Extraction, Analysis and Validation of Process Indicators
- Requirements for Feature Extraction
  - *C* ontextual dependency (R1)
  - *O* penness (R2)
  - *R* eproducibility (R3)
  - *E* xtensibility (R4)
  - *V*ersatility (R5)
- Review of Tools Regarding R1-R5
  - PIAAC LogDataAnalyzer
  - Analyzing Log File Data with R
  - Analysis of Log Data using Finite State Machines (LogFSM)
- Summary

Minimal Workflow

Workflow and Differentiation Between Pre-Processing (Log Data), Low-Level Feature Extraction, and Analysis and Validation

Phase	Steps and Purpose	Term
Pre-Processing	- Preparing and parsing log data (cleaning) - Generation of datasets (transformation) - Purifying log data (including anonymization) - Documentation of log data	Log Data ↓
Low-Level Feature Extraction	- Validation of log data (sequences, contexts) - Theoretical derivation of indicators - Extraction of low-level features - Combination of low-level features to process indicators	↓ Process Indicators
Analysis and Validation	- Documentation of indicators - Life cycle of process indicators validation	

Platform-Specific Pre-Processing

- Log data from technology-based assessments are *event based*, often stored in technical formats (JSON, XML, Databases)
- Specific format for log data exist, not widely used (yet), e.g.,
  - The eXtensible Event Stream Format (XES, XES Working Group, 2016)
  - The xAPI (former Tin Can API) Format
- Log data are platform specific, but typically follow general pattern
  - Each event has an event type (however, the number and meaning of event types is unkown)
  - Event type determines the meaning and interpretation of event-specific attributes

Parsing Log Data

Screenshot Showing Log Data Provided by OECD for PISA 2012 Digital Reading Assessment:

	time	event	event_value	event_detail
1	1066.2	START_ITEM	NULL	NULL
2	1077.5	click	noID	seraing_home.php?tok=&link=275
3	1110.3	click	noID	ccc_home.php?tok=&link=217
4	1119.8	click	noID	ccc_programme_by_date.php?tok=&link=16
5	1140.8	click	previousBtn	stimulus.php
6	1142.5	click	previousBtn	stimulus.php#
7	1146.3	click	era002q2o4	item2.php
8	1146.3	change	era002q2o4	NULL
9	1149.4	click	NextButtonPForm	unit.php?PROCESSURI=http%3A%2F%2F127.0.0.1...
10	1151.4	END_ITEM	NULL	NULL
11	1151.4	click	noID	unit.php?PROCESSURI=http%3A%2F%2F127.0.0.1...

- The selected page is part of column `event_detail` in the form of a relative link with so-called query string

Generation of Cleaned Datasets

At least two options

a) *Flat and sparse log data table*: One combined table containing all event data, including the event-specific data located in columns, with missing values for event-specific data not provided for an event of a particular type (because event-specific data are either optional or not defined for this event type)

b) *Universal log format*: Multiple tables, each table containing data for one particular event type only, with event-specific data located in columns and missing values only for optional event-specific data not provided for a given instance of an event

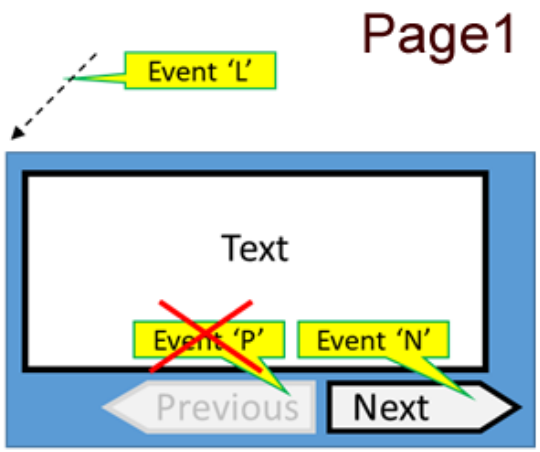
- Format is not of great importance, but tools need a study-independent structure (events; each event with person identifier, instrument identifier, timestamp, event type and event-specific attributes)

Contextual Dependency of Log Events (R1)

What is "special" about log data analyses?

- Meaning of a single log event (e.g., clicking a button) can differ depending on which log events happen before (or after).
- This so-called *contextual dependency* of log events can be illustrated with a simple example

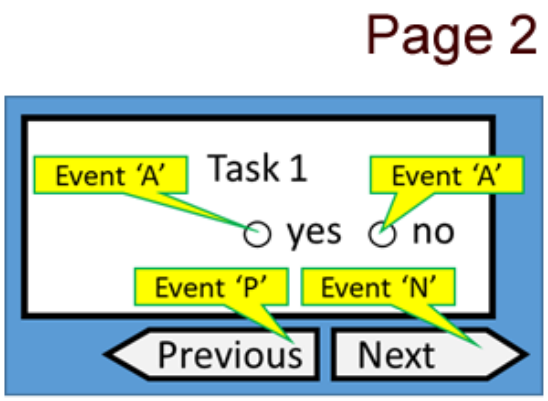
Hypothetical Unit with Three Pages (Page 1)



Consider a simple unit with 3 pages and 2 items. The first page contains a text, while the second and third pages contains the first and second items, respectively. Loading the unit is indicated by event 'L.' The pages contain two buttons, each of which triggers an event: The 'next' button triggers an event called 'N,' and the 'previous' button triggers an event called 'P.' If test-takers are not able to navigate to the previous page, event 'P' is invalid (as is the case, for instance, on the first page). Event 'N' identifies navigation to the next page.

Contextual Dependency of Log Events (R1)

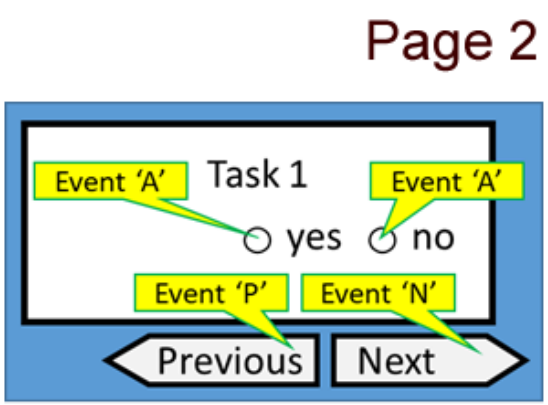
Hypothetical Unit with Three Pages (Page 2 und 3)



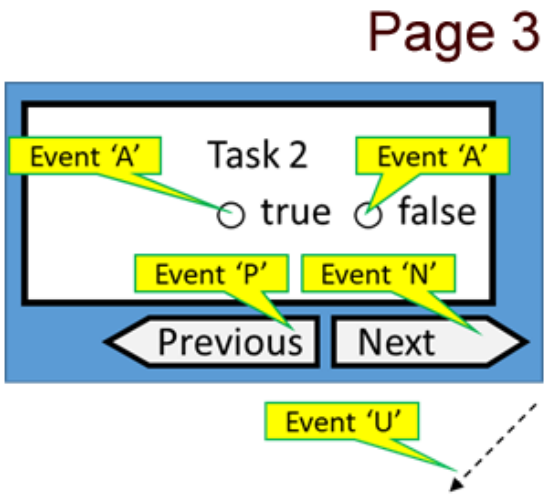
On the second page of the three-page unit, both the 'N' event and 'P' event can occur. The 'P' event indicates back-navigation to the text page, and the 'N' event indicates navigation to the last page containing the second item. However, the interpretation of 'P' and 'N' for Page 2 is different from Page 1. Hence, in order to interpret the events 'N' and 'P' correctly, it is necessary to incorporate the context (i.e., the current page, which can be identified from a valid sequence of 'N' and 'P' events). Events of type 'A' indicate an answer change; 'A' events related to Item 1 are valid only on Page 2.

Contextual Dependency of Log Events (R1)

Hypothetical Unit with Three Pages (Page 2 und 3)



On the second page of the three-page unit, both the 'N' event and 'P' event can occur. The 'P' event indicates back-navigation to the text page, and the 'N' event indicates navigation to the last page containing the second item. However, the interpretation of 'P' and 'N' for Page 2 is different from Page 1. Hence, in order to interpret the events 'N' and 'P' correctly, it is necessary to incorporate the context (i.e., the current page, which can be identified from a valid sequence of 'N' and 'P' events). Events of type 'A' indicate an answer change; 'A' events related to Item 1 are valid only on Page 2.



Process indicators are understood as characteristics of the test-taking process that can be derived from the sequence of activities, sequence of states, or presence of a particular state or action (i.e., sub-sequences of length 1). Process indicators can be stored in a wide format with one column per indicator. Process indicators can be qualitative, indicating whether an action or state was observed or a specified sequence occurred (dummy coding). Process indicators can also be quantitative, encoding, for instance, the number of occurrences of a particular action or the time required for a particular state or sequence of states.

Validation of Log Events

Contextual Dependency and Validation (of Sequence and Contexts)

- Context (i.e., the sequence in which events occur) must be taken into account in numerous log file analyses to validate the correctness of log data (Kroehne and Goldhammer 2018)
- Validation requires a kind of "memory" of information on previous log events
- Validation furthermore requires a profound understanding of the functioning of the interactive items
- Validation also requires documentation of interactive items and the platform as knowledge about the log data

Requirements (R1)

- *Contextual Dependency* (R1): The tool allows for the extraction of derived variables or new information (indicators), defined as prescriptive rules or algorithms for handling elementary log events, while taking into account that the same event can have different meanings depending on the context created by previous (and subsequent)[1] events.

- In my opinion, it can be concluded from this requirement, that the extraction of features from log events always requires some kind of algorithm.

[1] Taking into account not only previous events (i.e., involving some kind of memory) but also subsequent (future) events, as stipulated in R1, deserves special attention. While such a look ahead is possible when analyzing log data retrospectively, it is not available when log events are analyzed live during an ongoing assessment.

Requirements (R2-R5)

- *Openness* (R2): Extracted indicators can be embedded in a workflow that distinguishes among the phases of pre-processing, feature extraction, and analysis and validation of extracted indicators.

- *Reproducibility* (R3): The tool stores the exact specification of the algorithms used for feature extraction in such a way, that independent researchers can replicate the creation of process indicators.

- *Extensibility* (R4): The tool enables the extraction of user-defined indicators based on newly developed algorithms or modifying the algorithms used to extract existing indicators.

- *Versatility* (R5): The tool can be used with log data gathered from different assessment platforms or various data sources.

PIAAC Log Data Analyzer – Summary (R1-R5)[2]

- Contextual Dependency (R1): Indirectly considered when implementing the extraction in Java.
- Openness (R2): Exported aggregated variables and process indicators can be used for further analyses with the statistical software environment of the researcher's choice.
- Reproducibility (R3): Results are reproducable, as long as the particular versions of the PIAAC Log Data Analyzer are available.
- Extensibility (R4): Export of cleaned raw log events supported (can be used with statistical environments such as R and LogFSM)
- Versatility (R5): Restricted to PIAAC Round 1

[2] Details in the presentation by Carolin Hahnel.

The R Environment: Why?

R is in particular powerful because of packages

- R packges for (pre-) processing data
  - XML: `XML` / `xml2`
  - JSON: `jsonlite` / `tidyjson`
  - Databases: `DBI` / `sqldf`
  - Interop: `haven` / `foreign`
  - Encoding: `urltools`/ ...
- Specific packages for log data analysis
  - LOGAN (see first talk in this session)
  - Sequence analysis (e.g., TraMineR)
  - Reading XES data: `xesreadR`
- Packages with interface advanced methods
  - Natural language (NLP): OpenNLP / ...
  - Process mining: RWeka / bupaR / ...
  - Machine learning: ... (or phython?)

The R Environment – Summary (R1 - R3)

- Contextual Dependency (R1):
  - Easy for a simple form of algorithms, using, for instance, the lead and lag operators of the `dplyr` package
  - (Efficient) Vectorized implementation of algorithms can be demanding for inexperienced users
  - Programming skills are required to implement sophisticated algorithms when there is no existing R package providing additional support.
- Openness (R2)
  - Extracted indicators can be integrated into specific workflows[3]
- Reproducibility (R3)
  - Achieving reproducibility in feature extraction requires to share R programming (syntax), for instance, using OSF.

[3] Even server side processing is possible using, for instance, the `OpenCPU` package

The R Environment – Summary (R4 and R5)

- Extensibility (R4)
  - Algorithms written in the R language (or any language when integrated as R package) can be used and extended
  - The more complex the algorithms become, the more the analysis of log data becomes programming
  - However, debugging R code can still be cumbersome
  - Main question: Is there a general scheme or pattern behind the algorithms for feature extraction?

The R Environment – Summary (R4 and R5)

- Extensibility (R4)
  - Algorithms written in the R language (or any language when integrated as R package) can be used and extended
  - The more complex the algorithms become, the more the analysis of log data becomes programming
  - However, debugging R code can still be cumbersome
  - Main question: Is there a general scheme or pattern behind the algorithms for feature extraction?
- Versatility (R5)
  - R can be used with log data gathered from different assessment platforms or various data sources

R package LogFSM[4]

- Package aims to implement the general framework for analyzing log data using finite state machines described in Kroehne and Goldhammer (2018)
- Algorithmic decomposition of interaction processes with *finite state machines* as a general method that can be applied to define and operationalize different indicators (as a general method for theory-driven feature extraction)
- In short
  - Use the power of R and packages for data pre-processing and the analysis of process indicators
  - Integrate feature extraction into the assessment framework by focusing on the decomposition of the test-taking process

[4] `https://github.com/kroehne/LogFSM`

LogFSM (Conceptual Distinction)

- Separation of the operationalization of *activities* and *states* based on platform-specific log events (lower part) and definition of features and process indicators using states and action (upper part)

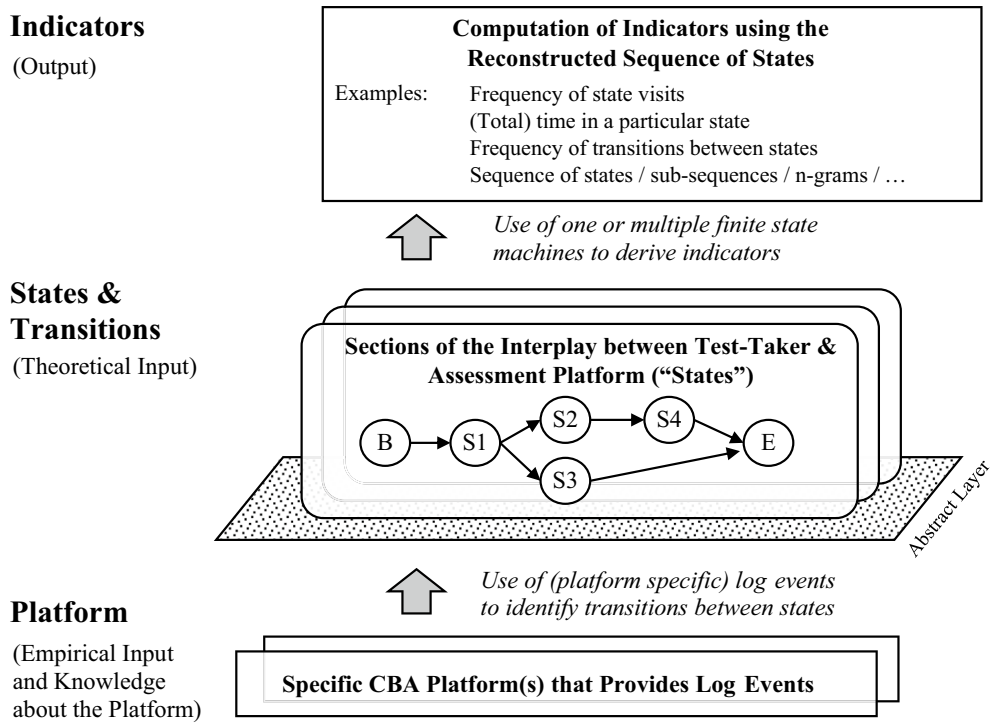
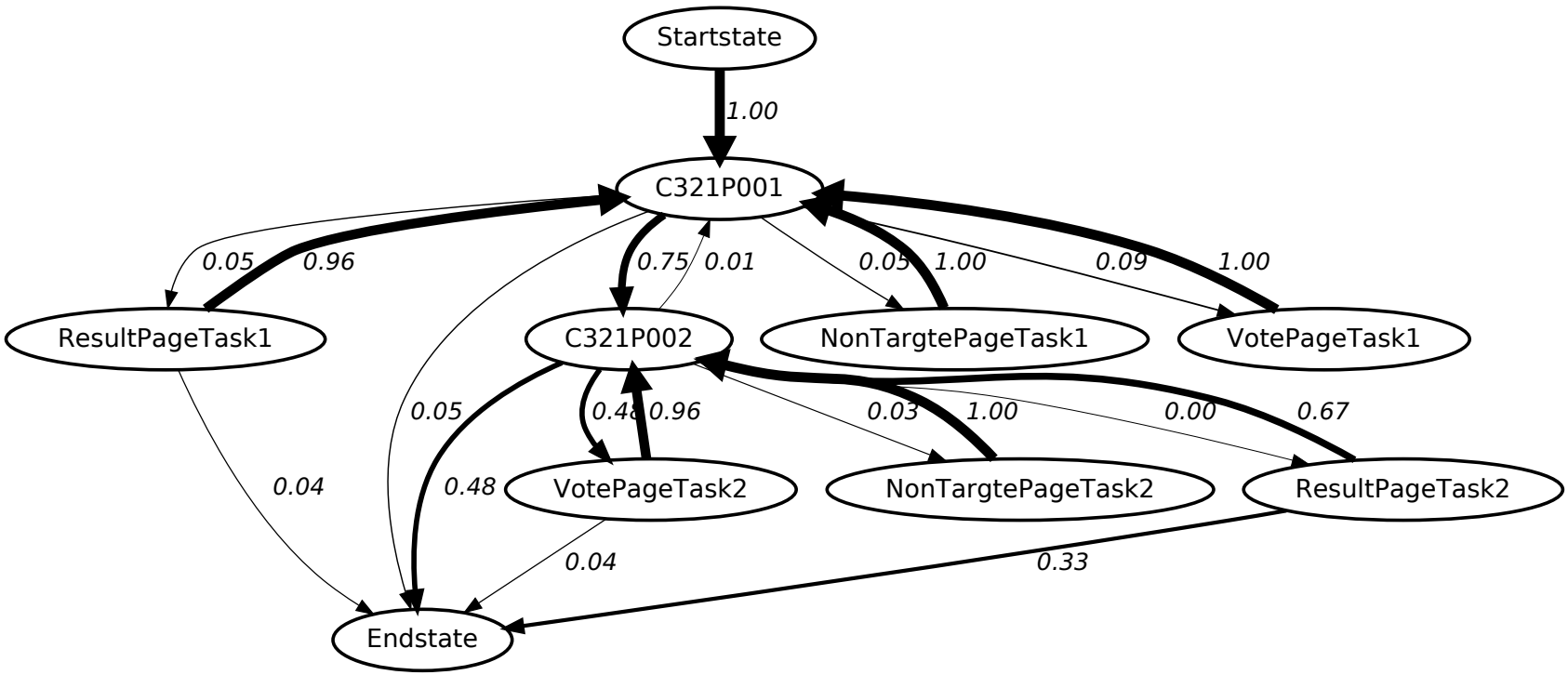


Fig. 2 Illustration of states between log events and indicators computed using reconstructed sequences of states

LogFSM (States and Transitions)

Illusration based on a unit with multiple pages
(data exported from PIAAC Log Data Analyzer)



LogFSM (Syntax Input)

Syntax for the Decomposition of Test-Taking Processes:

LogFSM Syntax	Comment
`myfsmsyntax <-"`	Syntax is assigned to an R string variable.
`Start: Startstate` `End: Endstate`	Definition of start and end states.
`Transitions: Startstate -> C321P001 @ EventName=initialized&value=C321P001`	Unit start with an event named "initialized", and the attribute value must contain the item name ("C321P001").

`Transitions: C321P001 -> VotePageTask1 @ EventName=ButtonLogEntry&id=L306F01B01`
`Transitions: C321P001 -> NonTargtePageTask1 @ EventName=EmbeddedLinkLogEntry`
`Transitions: C321P001 -> ResultPageTask1 @ EventName=ButtonLogEntry&id=L306F01B02`
`Transitions: C321P001 -> C321P002 @ EventName=gotoFrame&value=C321P002`
	The name of the event to the right of the @ sign defines the state to which the machine transitions from state C23P001.

`Transitions: C321P002 -> ResultPageTask2 @ EventName=ButtonLogEntry&id=L306F01B02`
`Transitions: C321P002 -> C321P001 @ EventName=gotoFrame&value=C321P001`
`Transitions: C321P002 -> NonTargtePageTask2 @ EventName=EmbeddedLinkLogEntry`
`Transitions: C321P002 -> VotePageTask2 @ EventName=ButtonLogEntry&id=L306F01B01`
	Rules defined in this part apply only when the state machine is in state C321P002, since C321P002 is named as the "from-State" to the left of the "->".

`Transitions: NonTargtePageTask1 -> C321P001 @ EventName=ButtonLogEntry;EmbeddedLinkLogEntry`
`Transitions: NonTargtePageTask2 -> C321P002 @ EventName=ButtonLogEntry;EmbeddedLinkLogEntry`
`Transitions: ResultPageTask1 -> C321P001 @ EventName=ButtonLogEntry;EmbeddedLinkLogEntry`
`Transitions: ResultPageTask2 -> C321P002 @ EventName=ButtonLogEntry;EmbeddedLinkLogEntry`
	Events that can trigger the transition back to states C23P001 and C23P002 are listed, separated by ";".

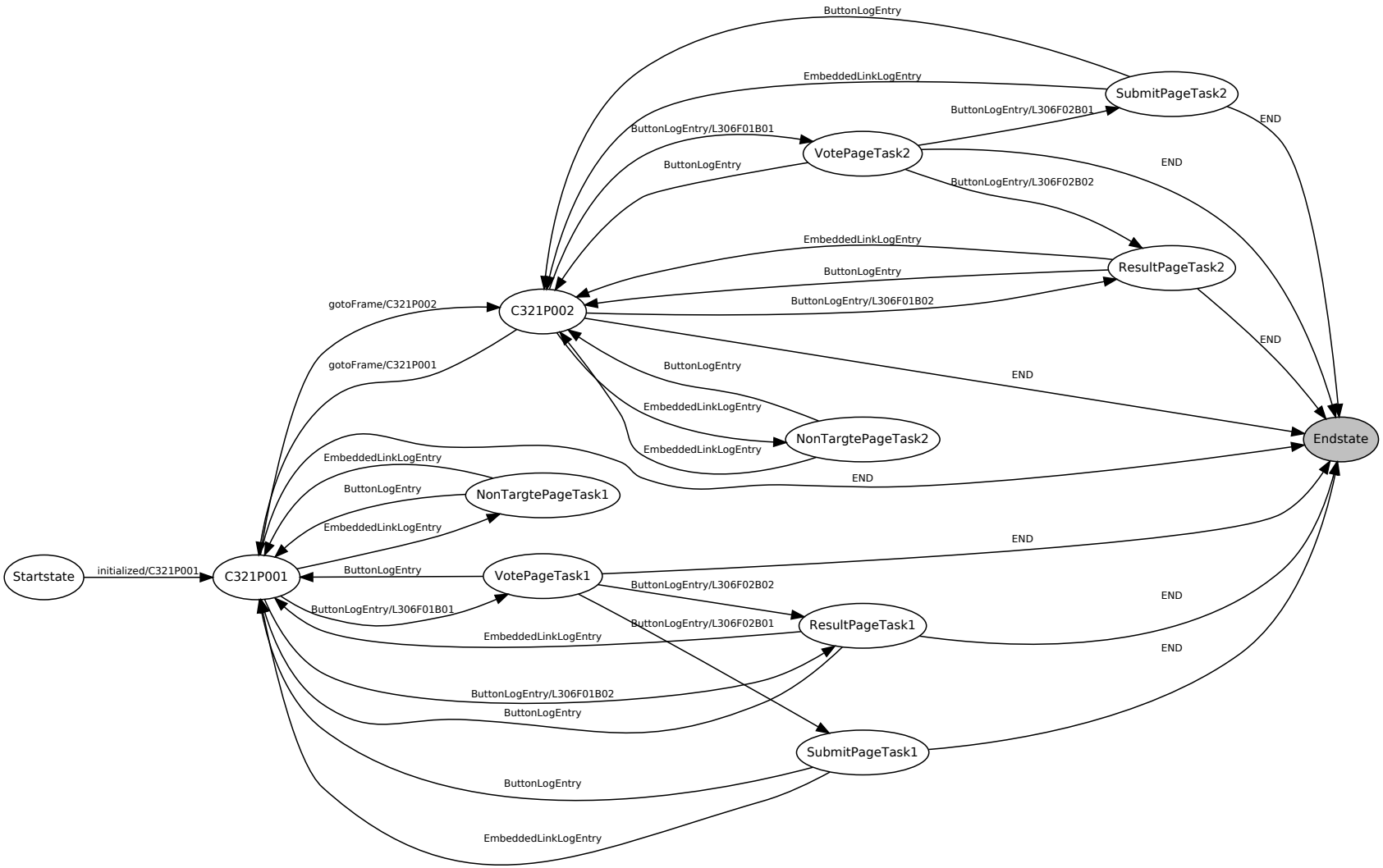
`Transitions: VotePageTask1 -> ResultPageTask1 @ EventName=ButtonLogEntry&id=L306F02B02`
`Transitions: VotePageTask1 -> C321P001 @ _`
`  EventName=ButtonLogEntry&cbaBackButton_101_4606665852447718`
	Each line contains a single rule (_ marks rules continued on the next line).

`Transitions: VotePageTask1 -> SubmitPageTask1 @ EventName=ButtonLogEntry&id=L306F02B01`
`Transitions: VotePageTask2 -> ResultPageTask2 @ EventName=ButtonLogEntry&id=L306F02B02`
`Transitions: VotePageTask2 -> C321P002 @ _`
`  EventName=ButtonLogEntry&cbaBackButton_101_4606665852447718`
`Transitions: VotePageTask2 -> SubmitPageTask2 @ EventName=ButtonLogEntry&id=L306F02B01`
	Multiple conditions, e.g. EventName= ButtonLogEntry and id=L306F02B01, can be combined using the "&" sign.

`Transitions: SubmitPageTask1 -> C321P001 @ EventName=ButtonLogEntry;EmbeddedLinkLogEntry` `Transitions: SubmitPageTask2 -> C321P002 @ EventName=ButtonLogEntry;EmbeddedLinkLogEntry`	Note that the event ButtonLogEntry can trigger a transition to different to-states depending on the from-state.
`Transitions: C321P001;C321P002;ResultPageTask1;ResultPageTask2;VotePageTask1; _` `  VotePageTask2;SubmitPageTask1;SubmitPageTask2 -> Endstate @ EventName=END"`	For all defined states (C321P001, …, SubmitPageTask2), the END event results in a transition to the end state.

LogFSM (FSM Input as Diagram)

UML State Chart for the Decomposition Defined Using the LogFSM



LogFSM – Summary (R1)

- Contextual Dependency (R1)
  - Events used to trigger transitions of a finite state machine, in a particular state
  - If the state machine is (because of previous events) in a different state, the meaning the event can be different (i.e., each transition is defined, conditional on a particular state)
  - Events are identified by the event type and by a specific pattern of event-specific values
  - Can be applied, as soon as event-specific values are parsed into separate columns
  - There is only one restriction, namely that the definition must be deterministic (i.e., a maximum of one transition per state can be active for a given combination of event type and event-specific values).
  - Specific operators allow look-ahead (i.e. contextual dependency by context to be created by subsequent events)

LogFSM – Summary (R2 an R3)

- Openness (R2)
  - LogFSM is used to reconstruct the sequence of states (or actions) for each test-taker
  - Summaries of this reconstructed sequence can be used to operationalize indicators
  - Indicators can be used in various ways within R or exported to be used in external tools

LogFSM – Summary (R2 an R3)

- Openness (R2)
  - LogFSM is used to reconstruct the sequence of states (or actions) for each test-taker
  - Summaries of this reconstructed sequence can be used to operationalize indicators
  - Indicators can be used in various ways within R or exported to be used in external tools
- Reproducibility (R3)
  - Separating the decomposition of the test-taking process into sequences of states and operationalization of indicators can increase reproducibility
  - Same state decomposition is possible with event-specific log data from different platforms
  - Visual presentation of state chart (created by default by the LogFSM package)

LogFSM – Summary (R4 and R5)

- Extensibility (R4)
  - The focus of LogFSM is feature extraction, based on the decomposition of test-taking processes
  - The specific finite-state machine is defined as syntax with some simple syntax rules, i.e., all algorithms that can be formalized with one or multiple state machines can be used
  - Some useful aggregates (i.e., the n-grams of states, time on each state and each sub-sequence of states etc.) are computed automatically
  - However, there the augmented raw log data table is provided, i.e., many different operationalizations are possible

LogFSM – Summary (R4 and R5)

- Extensibility (R4)
  - The focus of LogFSM is feature extraction, based on the decomposition of test-taking processes
  - The specific finite-state machine is defined as syntax with some simple syntax rules, i.e., all algorithms that can be formalized with one or multiple state machines can be used
  - Some useful aggregates (i.e., the n-grams of states, time on each state and each sub-sequence of states etc.) are computed automatically
  - However, there the augmented raw log data table is provided, i.e., many different operationalizations are possible
- Versatility (R5)
  - LogFSM can be applied to all log data that can be transformed into the flat and sparse log data table (as described above)
  - R can be used to pre-process data (using the different packages, as listed above)

Summary

The "CORE V" requirements were illustreated

- C: Tools need to tackle the specific nature of log data (i.e., the *contextual dependency* of log events)
- O: Tools should support different workflows, including those that allow secondary analysis using log data (*openness*)
- R: Of particular importance should be the requirement that tools should promote the traceability of log data analysis (*reproducibility*)
- E: To allow the development of new, innovative approaches, tools might not be closed (*extensibility*)

- V: From a researchers perspective tools should be *versatile* (i.e, not necessarily tools for specific studies only) and *extensible* (i.e., not only the implementation of already investigated indicators)

Thank you.

Questions and Comments: kroehne@dipf.de

PIAAC Log Data Analyzer:

- https://www.oecd.org/skills/piaac/log-file/

LogFSM R Package:

- www.logfsm.com / https://github.com/kroehne/LogFSM

Reference:

Kroehne, Ulf, and Frank Goldhammer. 2018. "How to Conceptualize, Represent, and Analyze Log Data from Technology-Based Assessments? A Generic Framework and an Application to Questionnaire Items." *Behaviormetrika*. https://doi.org/10.1007/s41237-018-0063-y.