EventParser and EpiTest

A User Guide

© JAN CHRISTOPH MEISTER

This User Guide gives a brief introduction to the concepts and functionality of the markup tool EventParser (currently version 4.37) and the analysis tool EpiTest (currently version 3.85). The underlying narratological theory of narrated action is discussed in detail in my book *Computing Action. A Narratological Approach*. Berlin, New York (De Gruyter) 2003. Please report any problems encountered with either program to mail@jcmeister.de . You are welcome to copy or change any of the program code for non-commercial purposes. Any commercial use or exploitation of EventParser and EpiTest or part thereof requires my written authorization.

Contents

1. The EventParser Program

The most important lesson to be drawn from our theoretical discussion of the concept of *action* in *Computing Action. A Narratological Approach* (2003) is that a fictive happening, even in its most basic form (*X and then Y*), never enters our consciousness in and of itself; it can do so only in the form of an interpretive construct. This basic construct of action logic, which we refer to as the EVENT, corresponds to an assertion by the narrator or an assumption by the reader that two causally or chronologically linked states represent a transformation in the properties of a fictive object. Points (a) to (d) below will serve as a brief reminder of how EVENTS are defined and an outline of how they are encoded (the construction of EVENTS is known as encoding in Event-Parser terminology). Note that for brevity's sake EventParser (as well as the EpiTest program to be described in chapter 2) use shortened terms and refer to the two types of STATE OF AFFAIRS (i.e., expositional vs. dispositional states) as two STATUSES, to a fully defined expositional STATE OF AFFAIRS as EXPOSITION and to its counterpart as DISPOSITION.[1]

(a) It must be possible to assign every predicate to a pre-existent predicate class.
(b) A FOCUS must be present, where a FOCUS is an identifiable, narratively based perspective of perception of an object in the narrated world.
(c) Under this FOCUS, the state of a fictional object of perception must be seen to be distinguished by predicate$_1$ at point t_1 in fictional time. Such a complex of object and predicate is referred to as a fictional STATE OF AFFAIRS. The first STATE OF AFFAIRS in an EVENT construct is termed the *expositional* STATE OF AFFAIRS (= EXPOSITION).
(d) Under the same FOCUS, or a different FOCUS which has an equivalent epistemological function (i.e. which fulfils the same epistemological truth conditions), the state of the identical object from (c) must be seen to be distinguished by predicate$_2$ at point t_2 in fictional time.[2] This is the *dispositional* STATE OF AFFAIRS (= DISPOSITION).

[1] Program specific terms appearing on the user interface of EventParser and EpiTest (in particular those defining windows and buttons) are generally identified by small caps in italics, e.g. *EVENT PARSING* window.

[2] The use of EventParser has to date been confined to OBJECT EVENTS; the following discussion therefore assumes the epistemological congruence of FOCUS and object. However, we must not ignore the theoretical distinction between the two, and nor should we forget the presence of the DISCOURSE EVENT as a theoretical concept.

Like all symbolic systems, narrating texts are characterised by a referential economy that suppresses information which is redundant or relatively easy for recipients to deduce from contextual evidence. To enable our theory to cope with this fact, we must formulate a supplementary rule which governs what kind of object can be involved in an EVENT construct:

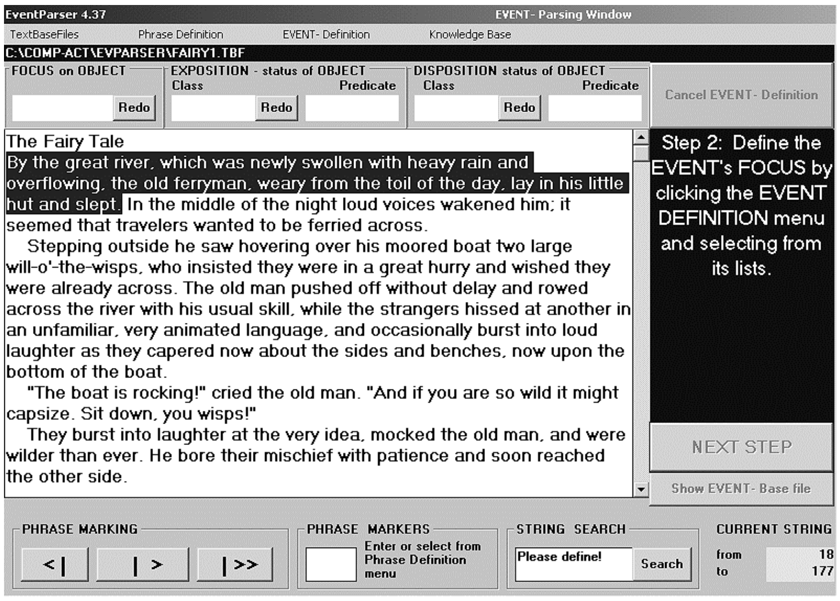
(e) The fictional object in an EVENT construct must fall into one of two classes. The first class consists of fictional objects which are referenced directly (e.g. by personal names or pronouns) or indirectly (e.g. by iterative or durative phrases such as *during* or *at once*) in a literal narrative assertion of their existence. The second, less obvious, class is composed of fictional objects whose existence at point t_1 in time can be hypothetically postulated on the basis of points of narrative indeterminacy by back-projecting it from t_2 (the dispositional STATE OF AFFAIRS).

It should be emphasized that rule (e) is not bidirectional: it cannot be inverted so as to allow us to look into the future beyond an expositional STATE OF AFFAIRS. In other words, the rules of interpretation allow us to assume that, if the text documents the present existence of an object, the object must also have existed virtually in the fictional past. We cannot, however, speculate about how, if at all, such an object will exist in the fictional future after the most recent description of the state of the fictional world in the narrative. (The only exception to this is when the narrative itself opens the fictional world to the future, for example, by making prophetic predictions.) We cannot test an assertion that epistemologically (i.e. in terms of FOCUS) transcends the interpreter's maximum possible stable knowledge state. It follows that we cannot falsify ACTION constructs whose formation depends on such an assertion; they are therefore beyond the scope of our approach.

We can now turn to the practical description of EventParser. Apart from the welcome window, which appears when it is opened, the program contains a total of four windows. These windows provide an interface with which the user can designate and mark up the EVENTS in a text and make declarations to elucidate the world knowledge on which his definition of EVENTS depends.

(1) The *EVENT PARSING* window is the heart of the text-processing interface.
(2) The *PREDICATE DEFINITION* window allows the user to define new predicates.
(3) The *EVENT BASE FILE* window allows the user to view and select encoded events.
(4) The *SEMANTIC DICTIONARY* window lists all the descriptive terms currently in use. The list represents the program's knowledge database, which expands dynamically as a text is processed.

Once the text to be processed has been loaded, it is displayed in the *EVENT PARSING* window. The user can then process the text by performing a prescribed series of definition operations. The help panel on the right-hand side of the window displays instructions which guide the user through each stage of the mark-up process. The first step in defining our first EVENT is to mark the text which contains its EXPOSITION. We do this by clicking and dragging with the mouse to select the appropriate range of characters as illustrated in figure 1.1.[3]



The Fairy Tale
By the great river, which was newly swollen with heavy rain and overflowing, the old ferryman, weary from the toil of the day, lay in his little hut and slept. In the middle of the night loud voices wakened him; it seemed that travelers wanted to be ferried across.
Stepping outside he saw hovering over his moored boat two large will-o'-the-wisps, who insisted they were in a great hurry and wished they were already across. The old man pushed off without delay and rowed across the river with his usual skill, while the strangers hissed at another in an unfamiliar, very animated language, and occasionally burst into loud laughter as they capered now about the sides and benches, now upon the bottom of the boat.
"The boat is rocking!" cried the old man. "And if you are so wild it might capsize. Sit down, you wisps!"
They burst into laughter at the very idea, mocked the old man, and were wilder than ever. He bore their mischief with patience and soon reached the other side.

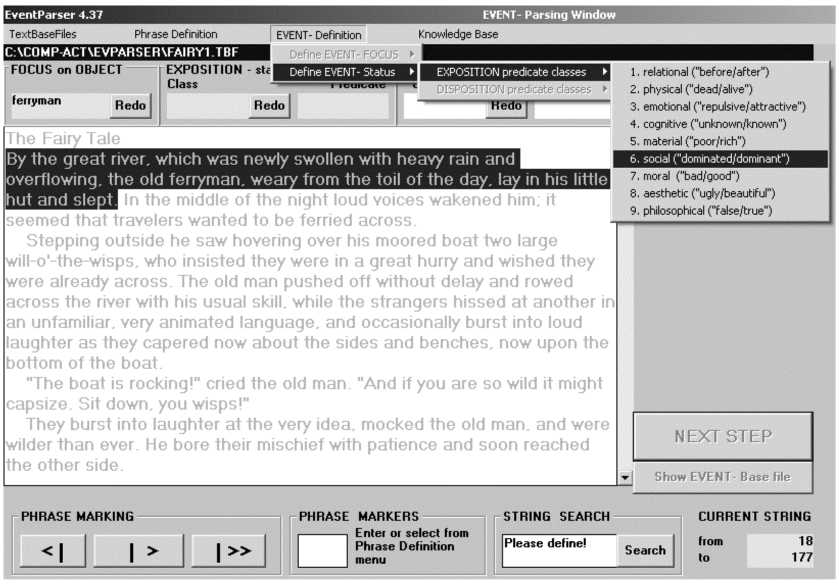
Step 2: Define the EVENT's FOCUS by clicking the EVENT DEFINITION menu and selecting from its lists.

Fig. 1.1: *EVENT PARSING* window

The help area now asks us to use the *EVENT DEFINITION* menu to define the FOCUS. The *EVENT DEFINITION* menu is the key user interface element involved in allocating descriptive terms. It displays lists from which, depending on the stage in the mark-up process, we can select focus descriptors or predicates that describe the expositional and dispositional statuses. When the user loads a text for the first time, the list of narrator- and actant-FOCUS descriptors is empty, so, as with all such lists, the user must enter the appropriate term manually in

[3] Our practical demonstration is based on the opening segment of Goethe's *Conversations of German Refugees* (1795) which is discussed in detail in *Computing Action. A Narratological Approach*.

the top field of the list. The program automatically saves terms entered in this way and displays them in the list the next time it is displayed. The STATUS options are slightly different, for they require one of the nine semantic classes to be selected before the class-specific lists can be viewed. As these latter lists are initially empty, we must enter the necessary predicate terms one by one when we start processing a new text.[4] The program does not restrict the type and scope of the user's predicates; thus, when using EventParser to encode a text, especially for a series of experiments, the researcher should ensure the comparability of the terms by establishing appropriate conventions in advance. When a descriptive term is assigned, it is displayed in the relevant field of the grey panel just above the text listing (as is already the case with our *ferryman* in figure 1.2). Isolated errors can be corrected by using the *REDO* buttons next to each field. Alternatively, the user can reject the entire EVENT definition by clicking the *CANCEL EVENT DEFINITION* button; each stage in the EVENT definition process must then be repeated from scratch.


EventParser 4.37
EVENT- Parsing Window
TextBaseFiles
Phrase Definition
EVENT- Definition
Knowledge Base
Define EVENT- FOCUS
Define EVENT- Status
EXPOSITION predicate classes
DISPOSITION predicate classes
1. relational ("before/after")
2. physical ("dead/alive")
3. emotional ("repulsive/attractive")
4. cognitive ("unknown/known")
5. material ("poor/rich")
6. social ("dominated/dominant")
7. moral ("bad/good")
8. aesthetic ("ugly/beautiful")
9. philosophical ("false/true")
Redo
Redo
Redo
NEXT STEP
Show EVENT- Base file
PHRASE MARKING
<|
|>
|>>
PHRASE MARKERS
Enter or select from Phrase Definition menu
STRING SEARCH
Search
CURRENT STRING

C:\COMP-ACT\EVPARSER\FAIRY1.TBF

FOCUS on OBJECT: ferryman

EXPOSITION - sta… Class

Please define!

from 18
to 177

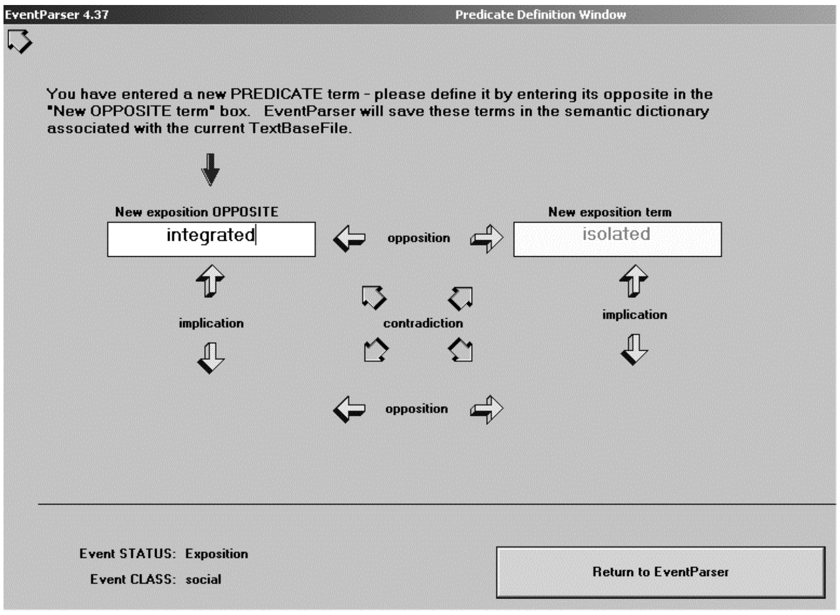
The Fairy Tale
By the great river, which was newly swollen with heavy rain and overflowing, the old ferryman, weary from the toil of the day, lay in his little hut and slept. In the middle of the night loud voices wakened him; it seemed that travelers wanted to be ferried across.
Stepping outside he saw hovering over his moored boat two large will-o'-the-wisps, who insisted they were in a great hurry and wished they were already across. The old man pushed off without delay and rowed across the river with his usual skill, while the strangers hissed at another in an unfamiliar, very animated language, and occasionally burst into loud laughter as they capered now about the sides and benches, now upon the bottom of the boat.
"The boat is rocking!" cried the old man. "And if you are so wild it might capsize. Sit down, you wisps!"
They burst into laughter at the very idea, mocked the old man, and were wilder than ever. He bore their mischief with patience and soon reached the other side.

Fig. 1.2: *EVENT DEFINITION* menu in operation

The screenshot in figure 1.2 shows the state of the program when the user has defined *ferryman* as an actant FOCUS, confirmed the definition by pressing *ENTER* or clicking the *NEXT STEP* button, and opened the list of nine expo-

4 Note, however, that predicate lists from previous sessions can be reused in new texts simply by copying the **.lbf* files.

sitional STATUS predicate classes in the *EVENT DEFINITION* menu. The class terms are prespecified; in the current version, they were chosen on a purely intuitive basis—one of the shortcomings of our theory and methods as they stand. In part 1, we decided that the *isolated* predicate should be assigned to the EXPOSITION selected in figures 1.1 and 1.2. If we click on the *social* option in the class list, an empty list appears (not shown in the screenshot). We enter our new term *isolated* at the top of this empty list and confirm the new entry by pressing *ENTER*. The *PREDICATE DEFINITION* window is then displayed as in figure 2.2.3:


You have entered a new PREDICATE term - please define it by entering its opposite in the "New OPPOSITE term" box. EventParser will save these terms in the semantic dictionary associated with the current TextBaseFile.

New exposition OPPOSITE: integrated

New exposition term: isolated

Event STATUS: Exposition

Event CLASS: social

Fig. 1.3: *PREDICATE DEFINITION* window (expositional predicate)

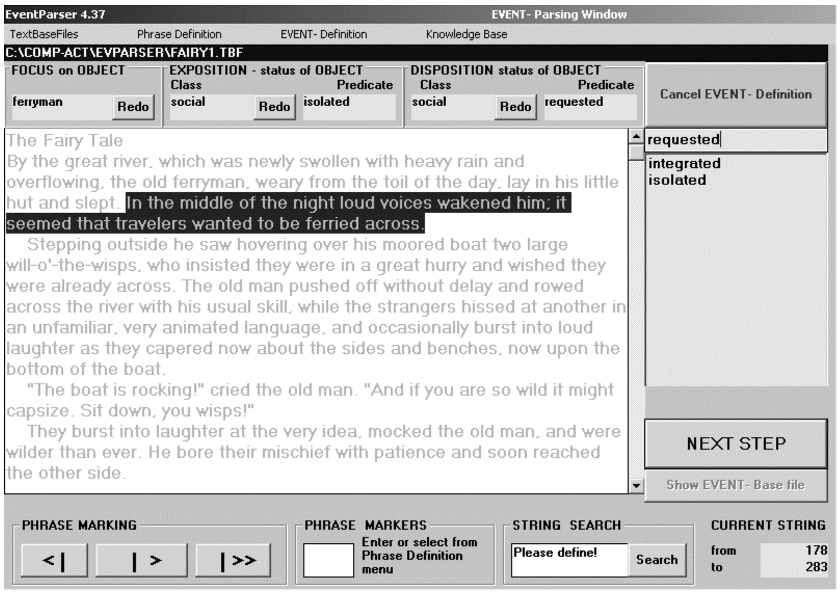
In the *PREDICATE DEFINITION* window, EventParser prompts the user to enter the opposite term which accompanies *isolated*. The layout of the window shows the upper axis of a semantic square in the making. We define *integrated* as the opposite term of *isolated* and by doing so give the program a first, small indication of the normative and cognitive frame of reference which applies to the EVENT being defined. After entering the new opposite term and clicking the *BACK TO EVENTPARSER* button, we are returned to the main *EVENT PARSING* window.

The fields in the grey panel above the text listing now display the current state of the EVENT definition: we can see that the FOCUS is *ferryman* and that the expositional STATUS of the object is described by the *isolated* predicate, which is a member of the *social* predicates class. The definition of the selected segment of text as an EXPOSITION is now complete. Note that the numerical indices of the first and last characters in the selection (18 and 177 respectively) are displayed in the lower right-hand corner of the window under the heading *CURRENT STRING*.

The program's help panel now prompts us to repeat the process by selecting and defining the dispositional STATE OF AFFAIRS in a similar manner. It is up to the individual interpreter to decide where this second STATE OF AFFAIRS is located and how wide the scope of its definition should be. In chapter 1.4 of *Computing Action*, we decided that the DISPOSITION is contained in the following segment:

> In the middle of the night loud voices wakened him; it seemed that travellers wanted to be ferried across.
>
> Goethe 1989:70

We also decided to define a categorially homogenous dispositional STATUS. As illustrated in figure 1.4, we therefore provide the DISPOSITION with the *requested* predicate, which belongs to the same class (*social*) as the expositional predicate.



C:\COMP-ACT\EVPARSER\FAIRY1.TBF

FOCUS on OBJECT: ferryman
EXPOSITION - status of OBJECT: Class: social; Predicate: isolated
DISPOSITION status of OBJECT: Class: social; Predicate: requested

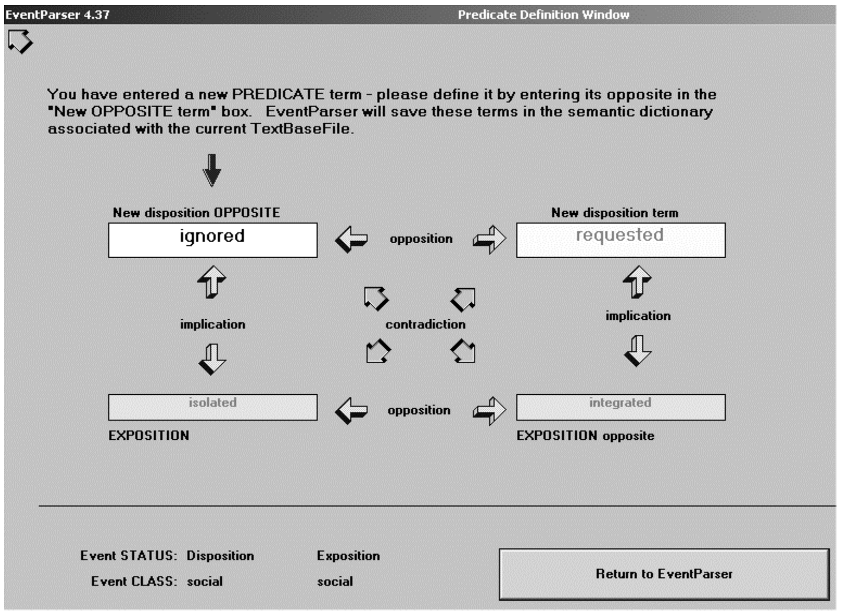
requested
integrated
isolated

The Fairy Tale
By the great river, which was newly swollen with heavy rain and overflowing, the old ferryman, weary from the toil of the day, lay in his little hut and slept. In the middle of the night loud voices wakened him; it seemed that travelers wanted to be ferried across.
Stepping outside he saw hovering over his moored boat two large will-o'-the-wisps, who insisted they were in a great hurry and wished they were already across. The old man pushed off without delay and rowed across the river with his usual skill, while the strangers hissed at another in an unfamiliar, very animated language, and occasionally burst into loud laughter as they capered now about the sides and benches, now upon the bottom of the boat.
"The boat is rocking!" cried the old man. "And if you are so wild it might capsize. Sit down, you wisps!"
They burst into laughter at the very idea, mocked the old man, and were wilder than ever. He bore their mischief with patience and soon reached the other side.

STRING SEARCH: Please define!
CURRENT STRING: from 178, to 283

Fig. 1.4: *EVENT PARSING* window with predicate list

As this involves the use of a descriptive term which the program has not encountered before, we have to enter it manually in the list box on the right-hand side of the main window, which now also contains *integrated* and *isolated*, the terms that we entered when defining the EXPOSITION. As before, we are prompted to define the new *requested* predicate in the *PREDICATE DEFINITION* window, which now displays a semantic square in which three positions have already been filled. The first pair of opposite terms, *isolated* and *integrated*, has moved to the bottom of the square; the new term, *requested*, occupies the top right-hand corner. By defining its opposite as *ignored* in the top left-hand field (see figure 1.5), we can complete our first semantic square. In the newly created square, the diagonal from *isolated* to *requested* represents the real event construct, while the diagonal from *ignored* to *integrated* represents our first virtual event construct (that is, an implicitly suggested EVENT which is possible according to our world knowledge and compatible with the narrated world as we have read it).


EventParser 4.37 Predicate Definition Window

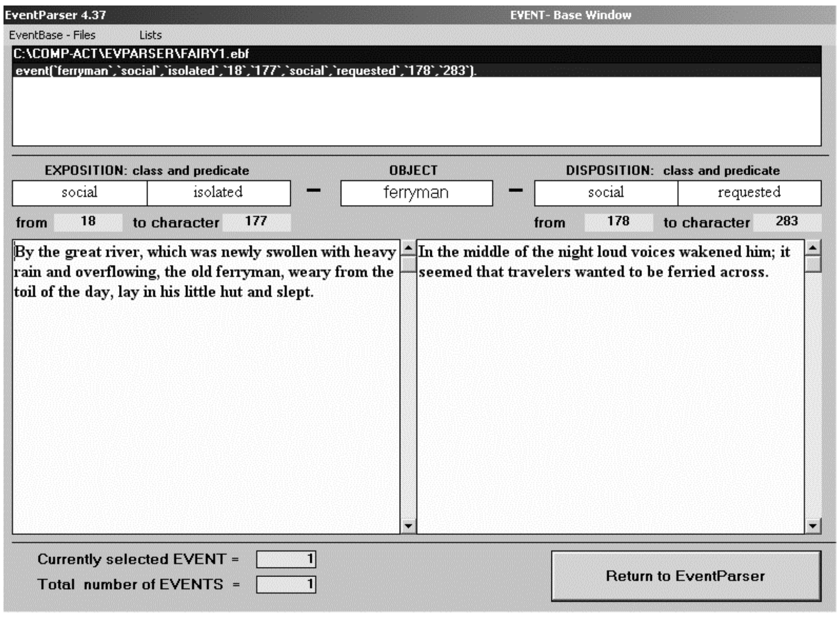
You have entered a new PREDICATE term - please define it by entering its opposite in the "New OPPOSITE term" box. EventParser will save these terms in the semantic dictionary associated with the current TextBaseFile.

New disposition OPPOSITE: ignored
New disposition term: requested
EXPOSITION: isolated
EXPOSITION opposite: integrated

Event STATUS: Disposition Exposition
Event CLASS: social social

Fig. 1.5: *PREDICATE DEFINITION* window (dispositional predicate)

When we return to the *EVENT PARSING* window, EventParser gives us the opportunity to add the EVENT we have just defined to the EVENT database. We do this by clicking the *ADD TO EVENT BASE* button, after which we are presented with a list of all the EVENTS that have been defined (see figure 1.6). Clicking on any EVENT in the list displays its associated segments of text.



```
C:\COMP-ACT\EVPARSER\FAIRY1.ebf
event(`ferryman`,`social`,`isolated`,`18`,`177`,`social`,`requested`,`178`,`283`).
```

EXPOSITION: class and predicate: social, isolated — OBJECT: ferryman — DISPOSITION: class and predicate: social, requested

from 18 to character 177 — from 178 to character 283

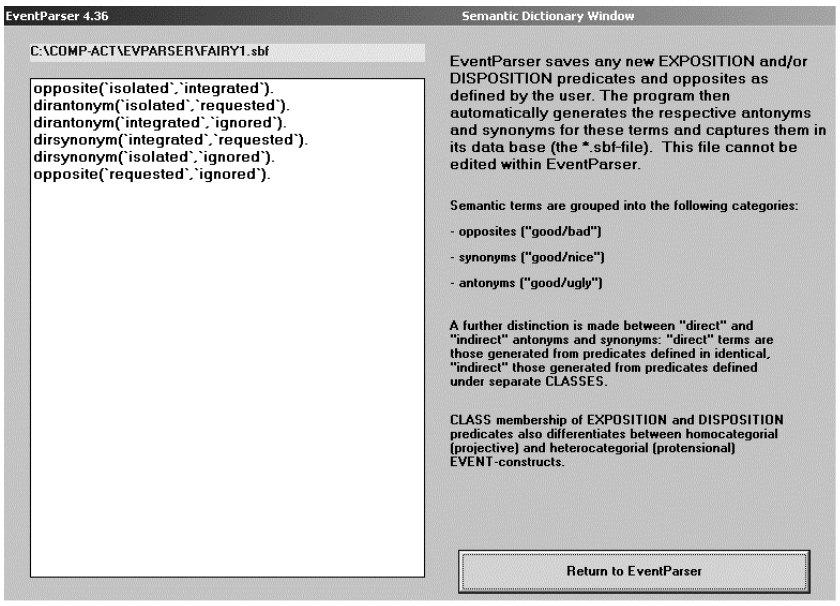
By the great river, which was newly swollen with heavy rain and overflowing, the old ferryman, weary from the toil of the day, lay in his little hut and slept.

In the middle of the night loud voices wakened him; it seemed that travelers wanted to be ferried across.

Currently selected EVENT = 1
Total number of EVENTS = 1

Fig. 1.6. *EVENT BASE* window

Clicking the *RETURN TO EventParser* button returns us to the main window, where a range of additional menus and buttons can be used to invoke further functions. In particular, the *KNOWLEDGE BASE* menu allows us to examine the knowledge database that is incrementally expanded as successive predicates are declared. The knowledge database is saved in a file with the *.sbf* suffix (sememe database file) and is displayed in the *SEMANTIC DICTIONARY* window, where semantic terms appear in the order in which they were entered or generated. In the example screenshot in figure 1.7, the knowledge database contains two pairs of opposites, two pairs of direct antonyms, and two pairs of direct synonyms. The antonyms and synonyms are *direct* because their elements belong to the same predicate class.



```
C:\COMP-ACT\EVPARSER\FAIRY1.sbf

opposite(`isolated`,`integrated`).
dirantonym(`isolated`,`requested`).
dirantonym(`integrated`,`ignored`).
dirsynonym(`integrated`,`requested`).
dirsynonym(`isolated`,`ignored`).
opposite(`requested`,`ignored`).
```

EventParser saves any new EXPOSITION and/or DISPOSITION predicates and opposites as defined by the user. The program then automatically generates the respective antonyms and synonyms for these terms and captures them in its data base (the *.sbf-file). This file cannot be edited within EventParser.

Semantic terms are grouped into the following categories:

- opposites ("good/bad")

- synonyms ("good/nice")

- antonyms ("good/ugly")

A further distinction is made between "direct" and "indirect" antonyms and synonyms: "direct" terms are those generated from predicates defined in identical, "indirect" those generated from predicates defined under separate CLASSES.

CLASS membership of EXPOSITION and DISPOSITION predicates also differentiates between homocategorial (projective) and heterocategorial (protensional) EVENT-constructs.

Fig. 1.7. *SEMANTIC DICTIONARY* window

We have now reached the end of our discussion of the main components and principles of the EventParser software.[5] The program allows us to define EVENTS in accordance with the theoretical and methodological criteria set out above and makes it possible to save formally consistent mark-up data (in event database, list database, and sememe database files) for subsequent differential and relational processing. The program creates the list database and sememe database files behind the scenes; they are then automatically combined to produce a master file with the suffix *.esf* (event-sememe file) in the format required for analysis in EpiTest.

Superficially, EventParser is little more than a simple mark-up tool, but it also has a far more important function: at every stage, it forces the user to explicitly define the semantic terms being used and thus make clear the symbolically represented world knowledge which influences the encoding

[5] A number of auxiliary functions (see the panel at the bottom of the *EVENT PARSING* window) have been added to assist the user in dividing texts into EVENTS. Searching for user-defined characters or strings (*PHRASE MARKERS*) takes some of the effort out of selecting segments of text (*PHRASES*). A second search option is provided by the *STRING SEARCH* field, in which the user can enter a string for which to search by scanning forward from the insertion point. Unlike the *PHRASE MARKERS* field, the *STRING SEARCH* option does not alter the selection; its primary function is to locate specific names, adjectives, or similar features.

process. In the next section of our discussion, we shall turn to EpiTest and the question of how to design an automated tool for analysing the combinatorial potential of EVENTS. What distinguishes a pair of EVENTS which can be linked to form an EPISODE? What kind of relations exist between two such EVENTS? And do interpreters prefer some ways of linking EVENTS to others? Only when we have considered these questions in a theoretical context will we be able to devise generative algorithms which answer them. The algorithms will be used in EpiTest to search for virtual EPISODE constructs and ACTION metaconstructs in EventParser record files. In the process, it will become clear that formally represented world knowledge, a side effect of EventParser mark-up, is the cornerstone of combinatorial analysis in EpiTest.

2. The EpiTest Program

The EpiTest program was developed with a specific task in mind: combinatorial analysis of **.esf* files produced by EventParser. The heart of the program consists of a set of algorithms which search saved databases of EVENTS for connections which correspond to the categories in our episode matrix. EpiTest was written in PROLOG (Programming in Logic), a programming language which has long been regarded as ideal for developing artificial intelligence models and is also one of the standard tools of computational linguistics.

The code listing of EpiTest version 3.85 (the version designed and used when writing the present study) runs to about 2,240 lines including comments.[6] The comments document the route taken through the code during execution and should help readers who are not familiar with PROLOG to understand how the program and its various models operate.

[6] Comments are sections of a source code listing inserted by the programmer to clarify his code to the reader—or indeed sometimes to himself! Comments are completely ignored during program execution; in PROLOG syntax, they are marked by a preposed % or enclosed in /*... */. – A file with the source code is part of the installation (see section 4 of this document).

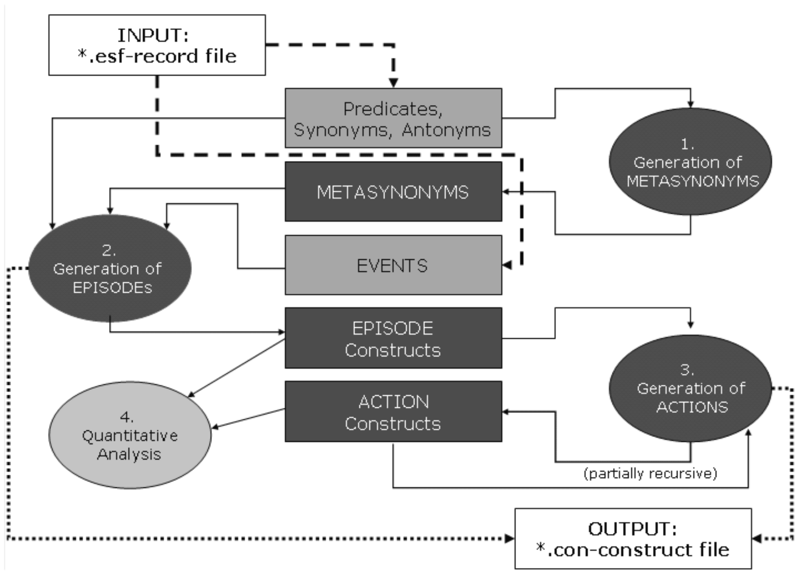


Fig. 2.1: EpiTest flowchart[7]

However, it will probably be easier to digest the flowchart shown in figure 2.1. With the help of the diagram, we can describe a complete execution cycle of EpiTest as follows:

0. Synonyms, antonyms, and EVENT constructs are read from the record file (broken lines).
1. METASYNONYMS are generated in Algorithm 1 (Module 6 in the program listing). They are then added to the program's dynamic knowledge database (solid line).
2. EPISODE constructs are generated by testing for combinatorial connections in Algorithm 2 (Module 4). The constructs are saved in the **.con* construct file and added to the dynamic knowledge database (dotted line).
3. ACTION constructs are generated by combinatorially connecting EPISODE constructs in Algorithm 3 (Module 5). The first ACTION

[7] The flowchart illustrates how our software exploits the combinatorial capabilities of PROLOG. EpiTest first builds a list of all the EPISODE constructs that fit the appropriate criteria and then initiates a second recursive process for building ACTION constructs. However, EpiTest is not really an intelligent program—although it adds newly generated factual knowledge (constructs and metasynonyms) to its knowledge database, it does not generate dynamic rule knowledge of any kind.

construct is saved in the dynamic knowledge database before the program recursively checks whether further EPISODES can be added to the ACTION chain. Complete ACTION constructs are saved in the **.con* construct file.

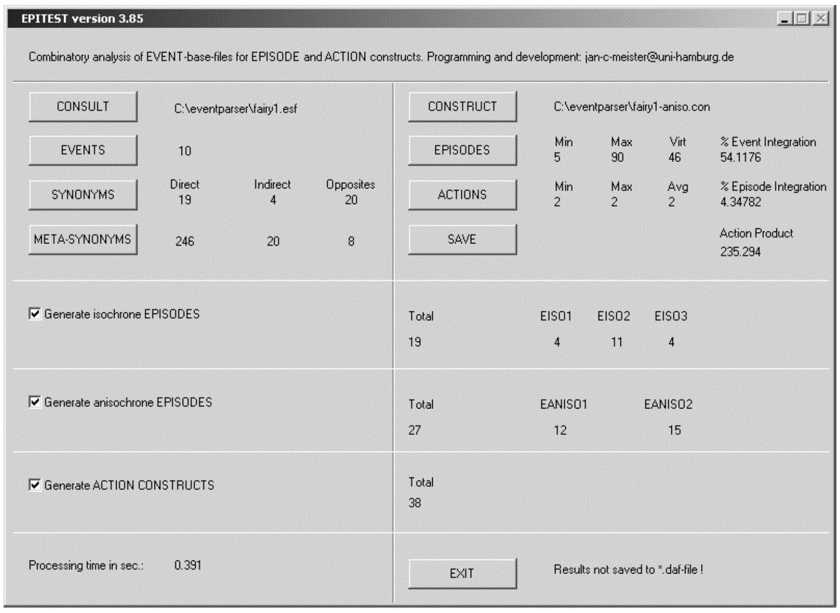
4. The constructs that have been generated are subjected to quantitative statistical analysis in Algorithm 4.

The reader may well ask why we should spend a considerable amount of time and effort designing these computational algorithms in order to implement what is already a highly abstract formal system of narrative theory and action logic. Some quantitative data may help illustrate the potential benefits.

In our illustration of the different types of EPISODE, we analysed the combinatorial potential of ten EVENTS and identified five acceptable EPISODES which can be produced from them. If we enter our ten basic events in the sample file *fairy1. esf* and analyse this file in EpiTest, the program generates the same five EPISODES as those we found manually. It also produces 246 METASYNONYMS which allow it to uncover hidden semantic connections and more than quintuple the size of its semantic database. On this extended basis, EpiTest generates forty-one additional, unpredicted EPISODE constructs, each of which fully satisfies the criteria of one of our five categories EISO_{1-3} and EANISO_{1-2}. Now, these results could have been worked out manually, given a few sharp pencils, plenty of paper, and even more patience—but not in the 0.33 seconds which our program needed to complete the task.[8]

When it has finished processing a file, EpiTest presents the display shown in figure 2.2.

[8] System specification: 2.66 GHz Pentium IV processor with 512 MB RAM, Windows XP.



EPITEST version 3.85

Combinatory analysis of EVENT-base-files for EPISODE and ACTION constructs. Programming and development: jan-c-meister@uni-hamburg.de

CONSULT C:\eventparser\fairy1.esf

EVENTS 10

SYNONYMS Direct 19 Indirect 4 Opposites 20

META-SYNONYMS 246 20 8

CONSTRUCT C:\eventparser\fairy1-aniso.con

EPISODES Min 5 Max 90 Virt 46 % Event Integration 54.1176

ACTIONS Min 2 Max 2 Avg 2 % Episode Integration 4.34782

SAVE Action Product 235.294

☑ Generate isochrone EPISODES — Total 19; EISO1 4; EISO2 11; EISO3 4

☑ Generate anisochrone EPISODES — Total 27; EANISO1 12; EANISO2 15

☑ Generate ACTION CONSTRUCTS — Total 38

Processing time in sec.: 0.391

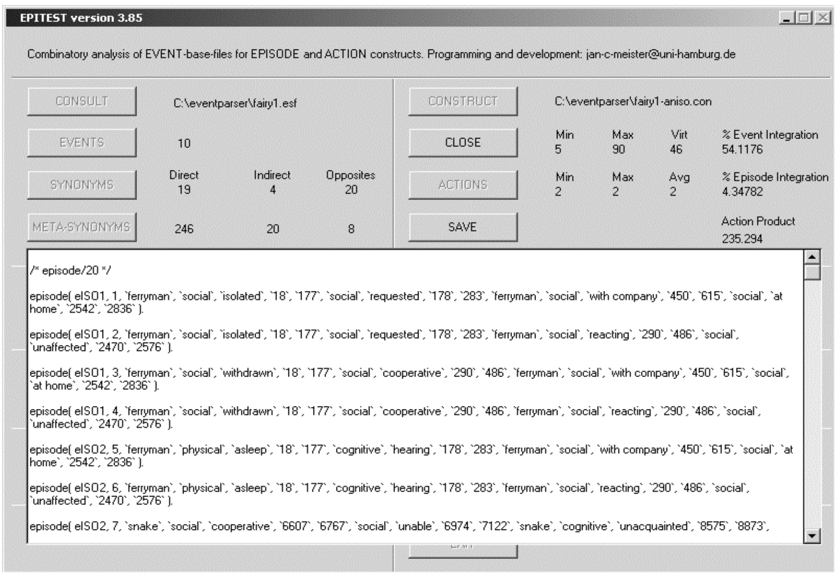
EXIT Results not saved to *.daf-file !

Fig. 2.2: EpiTest user interface

The various features of the program can be briefly described by considering the nine buttons in the screenshot reproduced in figure 2.2. The four buttons in the panel on the upper left-hand side relate to the database file being evaluated (in this case, it is *fairy1.esf*, which was previously produced in EventParser). *CONSULT* allows the user to open a new file; *EVENTS* lists the current file in a text field; *SYNONYMS* displays a similar list of categorially homogenous (direct) and categorially heterogenous (indirect) synonyms; *METASYNONYMS* calls the algorithm for generating METASYNONYMS (see above) and lists them in a text field. To the right of these last three buttons, the program displays the number of events that have been read and the number of user-defined and program-generated semantic terms.

After the program has read and prepared a **.esf* database, the user should specify what kind(s) of combinatorial analysis are to be performed. Depending on the checkboxes that have been selected, EpiTest will generate isochronous EPISODE constructs, anisochronous EPISODE constructs, or ACTION constructs. The buttons in the top right-hand corner concern the actual generation, saving, and display of these constructs. *CONSTRUCT* initiates the generation process and asks us to define the name of the results file (**.con* suffix). Once the program has completed its calculations and displayed the message 'Database analysis completed,' the results can then be displayed in text fields. *EPISODES* opens a list of the EPISODE constructs that have been found; *ACTIONS* shows the possible ways of linking these EPISODE constructs in the form of a PROLOG predicate *action(1, 2, 3,…, n*), where each of the numbers inside the parentheses is the index of an EPISODE which makes up the ACTION. *SAVE* allows the results of the various calculations to be stored in a separate **.daf (*data analysis) file. This file can be opened later in any text editor and contains a one-page statistical summary of the results. *EXIT* quits the program. The panel on the bottom right-hand side displays the number of EPISODES that were generated in each category and the total number of ACTION constructs they produced. In the lower left-hand corner, we can see the time taken by EpiTest to complete its calculations.

When EpiTest has finished generating EPISODE constructs on the basis of the *fairy1. esf* file and stored them in the *fairy1-aniso.con* results file, the user can click on *EPISODES* to display a list of the EPISODE constructs which were found, as shown in the screenshot in figure 2.3.



Combinatory analysis of EVENT-base-files for EPISODE and ACTION constructs. Programming and development: jan-c-meister@uni-hamburg.de

```
C:\eventparser\fairy1.esf                C:\eventparser\fairy1-aniso.con

10                                       Min    Max    Virt   % Event Integration
                                         5      90     46     54.1176
Direct      Indirect    Opposites
19          4           20               Min    Max    Avg    % Episode Integration
                                         2      2      2      4.34782
246         20          8
                                                              Action Product
                                                              235.294
```

```
/* episode/20 */

episode( elS01, 1, `ferryman`, `social`, `isolated`, `18`, `177`, `social`, `requested`, `178`, `283`, `ferryman`, `social`, `with company`, `450`, `615`, `social`, `at
home`, `2542`, `2836` ).

episode( elS01, 2, `ferryman`, `social`, `isolated`, `18`, `177`, `social`, `requested`, `178`, `283`, `ferryman`, `social`, `reacting`, `290`, `486`, `social`,
`unaffected`, `2470`, `2576` ).

episode( elS01, 3, `ferryman`, `social`, `withdrawn`, `18`, `177`, `social`, `cooperative`, `290`, `486`, `ferryman`, `social`, `with company`, `450`, `615`, `social`,
`at home`, `2542`, `2836` ).

episode( elS01, 4, `ferryman`, `social`, `withdrawn`, `18`, `177`, `social`, `cooperative`, `290`, `486`, `ferryman`, `social`, `reacting`, `290`, `486`, `social`,
`unaffected`, `2470`, `2576` ).

episode( elS02, 5, `ferryman`, `physical`, `asleep`, `18`, `177`, `cognitive`, `hearing`, `178`, `283`, `ferryman`, `social`, `with company`, `450`, `615`, `social`, `at
home`, `2542`, `2836` ).

episode( elS02, 6, `ferryman`, `physical`, `asleep`, `18`, `177`, `cognitive`, `hearing`, `178`, `283`, `ferryman`, `social`, `reacting`, `290`, `486`, `social`,
`unaffected`, `2470`, `2576` ).

episode( elS02, 7, `snake`, `social`, `cooperative`, `6607`, `6767`, `social`, `unable`, `6974`, `7122`, `snake`, `cognitive`, `unacquainted`, `8575`, `8873`,
```

Fig. 2.3: List of EPISODES generated by EpiTest

In this case, the original *fairy1. esf* file contains only ten EVENTS, but, even so, the EPISODE list displays a total of forty-two virtual EPISODE constructs. Representing EpiTest output in a list format like this makes clear that a considerable number of EPISODES (theoretically possible ways of combining EVENTS) can be identified by using a computer to analyse reception records (*.*esf* files produced in EventParser). The output does not lend itself to qualitative analysis—it is not a reading whose merits can be evaluated by critics, and the methods behind it cannot be judged against the aesthetic standards of a hermeneutics of interpretation. It is obviously meaningless to ask which of the forty-six EPISODE constructs or thirty-eight ACTION constructs in our data is more correct, more satisfying, or even better in the sense of being more faithful to the original text. Qualitative evaluations, where the original text of the literary narrative is paramount, might therefore seem irreconcilable with the quantitative calculations on which our model is based. But this is not the case—by introducing the concept of virtual action potential, we will we able to distinguish individual texts or even entire corpora from one another by studying the values of this new statistic.

The action potential has only one constant factor: the literary text, a concrete object which consists of a finite set of symbolic signs that are arranged, so the empirical evidence tells us, according to certain syntactic and grammatical rules. The signs must be processed in a number of ways—through reading, interpretation, recombination, evaluation, comparison, and so on—before we can even begin to speak of concepts, let alone actions. The signs are processed under the influence of many variables—the reader's ability, expectations, intertextual and world knowledge, and aesthetic norms, to name but a few. These variables differ not only from reader to reader (i.e. from constructor to constructor) but also inside individual reading subjects and receptive processes, where closer observation shows that they can vary dynamically from text to text and even from sentence to sentence. With each successive sentence, the reader knows more, discovers new associations, and expects different things. Reception in general, and the reception of ACTIONS in particular, is fundamentally dynamic in nature and cannot be reconstructed in a supposedly ideal reading situation in order to facilitate the experiments of our computational approach.

However, readers do not evaluate literary narratives on a purely subjective basis by considering the richness of their action logic, the coherence and depth of their chains of fictive happenings, and the originality and elegance with which they combine events. Readers also take part in discussions with other readers. Theoretically, the semiotic process may continue into infinity as Eco has described, but, in practice, we will terminate it on the basis of our pragmatic needs. The same, I believe, applies to the discussion, be it naive or critical or philosophical, of the action potential of a narrative text. Only if the reading (receptive construction) of action can create synthesis, meaning, does it make sense to try to define action potential in theory and measure it in practice. The closer a receptively constructed ACTION clings to the referential framework of fictively represented actors, things, situations, and transformations, the more it is reduced to nothing more than a denotational function; but the more an ACTION distances itself from the fictive substance borne by a possible world and is condensed into a sequence of transformations which can be reconstructed as abstract formal propositions, the more it becomes a hollow, empty shell of meaning, the illusory meaning of a receptive happening that portrays itself as the only true action. From this perspective, the empirical reading and construction of ACTION can be seen as an attempt to find a way through the world of latent possibilities contained in a network of EVENTS. Our definition of the EPISODE as the smallest such ACTION construct provides us with a unit with which to measure and quantify this world of possibilities. We shall begin by considering the upper and lower theoretical values that can arise.

If only a single EVENT is present, it is impossible to form an EPISODE at all. Once two EVENTS, the minimum quantity needed for an EPISODE, are present, there are two ways of combining them. The definitions below assume that our two EVENTS are (1) *Egg from chicken* and (2) *Chicken from egg*.

(a) Hyperdetermination: every EVENT capable of entering into a connection can enter into one and only one such connection. Thus, we must read our two EVENTS either as (1) + (2) or as (2) + (1). The formula required in this case is simple: for n EVENTS, the minimum number of EPISODE constructs (Ep_{min}) is given by the formula $Ep_{\text{min}} = n / 2$.

(b) Hyperconnection: every EVENT can connect with every other EVENT to form a pair, but not with itself. The ontic primacy of chicken over egg is just as conceivable as the primacy of egg over chicken, and there are two possible EPISODES, (2) + (1) and (1) + (2) respectively. For n EVENTS, the maximum theoretical number of EPISODE constructs (Ep_{max}) is given by the formula $Ep_{\text{max}} = n(n - 1)$.

In a practical context, these simple formulae can be applied to obtain numerical values which describe the minimum and maximum levels of episodicity in a concrete text. They delineate the spectrum of EVENT combinations which are theoretically possible on the basis of a particular reading before syntagmatic and semantic criteria are considered. Putting the formulae into practice using the demonstration data of the ten EVENTS discussed above, we find that there is a theoretical minimum of $10 / 2 = 5$ EPISODES and a theoretical maximum of $10 * (10 - 1) = 90$ EPISODES. The actual number of virtual EPISODE constructs which can be produced will lie somewhere inside the theoretical range of five to ninety; in the present case, it is forty-six. Expressed as a percentage, this gives us an EVENT integration of just over 54 percent. In less formal terms, this means that (a) the EVENTS in the text can be combined into eighty-five theoretically possible distinct EPISODE constructs; and (b) of these eighty-five theoretically possible constructs, forty-six (i.e. 54.1 percent) can be virtually instantiated on the basis of the semantic and formal predications with which the EVENTS have been encoded. EpiTest calculates this value for us and displays it in the upper right-hand corner of the window under the heading *% EVENT INTEGRATION*.

The action potential of a text, which basically measures the degree to which a given reading can be translated into overall ACTION constructs, can now be evaluated in terms of the combinatorial affinities between atomic EVENT constructs and molecular EPISODE constructs. However, we shall not stop here; our quantitative measure of action potential will also consider the scope for EPISODES themselves to be joined together in action logic. This means that, having considered local (EPISODE-forming) EVENT synthesis above, we must now consider global (ACTION-forming) EVENT synthesis as well. Ideally, we would define a series of semantic criteria similar to those used in the EPISODE matrix; this would allow our theory to represent our intuitive knowledge that the scope of a particular action narrative in the form of an ACTION construct depends primarily on the presence of a dominant semantic category, or theme. However, such a refinement is beyond the remit of the present study, and we must be content with a provisional, pragmatic criterion instead. EpiTest's combinatorial algorithm is therefore based on the following plausibly simple, broad rather than narrow, definition of the ACTION construct: two EPISODES form an ACTION if and only if they are sequentially ordered in the semiotic continuum of the text. This test therefore does no more than provide a solution to the following question: assuming that EPISODE α is the first in an unbroken chain, what EPISODES (β, γ, δ,…) can be linked to α in an isochronous order? The answer obviously depends on the connective potential of the initial construct α in each case.[9] In the case of our

[9] Successive new connections will obviously become possible as we move through the second, third, and subsequent positions in the list. The corresponding ACTION lists are represented as *actlist (1, 2, 3, ..., n)* in the program syntax. However, EpiTest generates them with an algorithm that is partially deterministic rather than fully recursive: it takes from the database only the first EPISODE construct which isochronally follows its predecessor on any occasion. A fully recursive algorithm would generate a huge number of combinatorial variants, and we are not yet in a position to place such a burden on reader or computer. Practical tests have shown that it takes long enough to calculate deterministic ACTION constructs alone—of the 0.39 seconds needed to process the *Matrix.esf* file, only about 0.05 seconds were required to assemble the EPISODE constructs; the rest of the time was spent building ACTION lists. Processing *Unterhaltungen.esf*, the complete mark-up file for the *Conversations of German Refugees* (our example text in part 3), took over four hours on a 266 MHz Pentium II processor. (Advances in computer technology since the time of writing will have reduced this figure considerably by now.) If nothing else, these figures illustrate the magnitude of the

demonstration file, we obtain a striking result: despite the high EVENT integration of 54 percent, the forty-six virtual EPISODE constructs cannot produce an ACTION list with more than two elements.

Consider again the second row of figures in the top right-hand panel of the EpiTest window in figure 2.3. On the right of the *ACTION* button, we can see numerical values for the length of the shortest ACTION chain generated, the length of the longest ACTION chain generated, and the average length of all generated ACTION chains. In the present example, the value of each statistic is two. Under the heading *% EPISODE INTEGRATION*, we can see the percentage of virtual EPISODE constructs which are combined in the longest generated ACTION chain. In our example, we have obtained a very small value: 4.3 percent. Although this result may seem somewhat disconcerting at first, it is easily explained. Because our example EPISODES reused the same segment of text several times, it follows that the possibilities for building extended ACTION chains must be severely limited. We are concerned with a narrative text that has been read as having ten EVENTS, on which small basis a considerable number of possible EPISODES can be formed, but the text remains episodic in the non-technical sense of the word. When we consider the opposite situation, it becomes even more apparent that the episodicity of a text is related to the number of EVENT, EPISODE, and ACTION constructs which it contains. Texts with a low level of EVENT integration but a high potential for ACTION synthesis are weak in action, not because they are episodic like the above example, but simply because they lack available EPISODES. If a text has low levels of both EVENT integration and ACTION synthesis, it will lack connectable EVENT constructs and therefore the raw material of a proper story in the first place. The final case consists of texts which when read produce a high level of EVENT integration and a high level of virtual EPISODE integration; the most extreme example of such a text is one in which all the constituent EVENTS are arranged in an uninterrupted succession which consists of *and then* connectors.

We are now in a position to abandon the interim terms ‘action potential’ and ‘episodicity’. Each of the four representative text types described above can be quantitatively represented as the product of the EVENT integration and EPISODE integration percentage values. In the demonstration file, this value, which we shall refer to as the virtual ACTION PRODUCT, is 54.117 * 4.347 = 235.294. The EpiTest window displays this figure under the final heading in the top right-hand panel. EPISODE constructs support an almost unlimited range of possibilities; the ACTION PRODUCT is a measure of how much of this potential can be synthesized logically in a global ACTION construct (the latter is, as noted above, defined in very broad formal terms).

cognitive task which humans perform so easily and find so deceptively simple when they read action.

It should be emphasized again here that our final quantification of the ACTION PRODUCT is in no way meant to be an objective measure of the coherence of a given narrative text. This is obvious not least because in itself a number like 235.294 says absolutely nothing. Is 235.294 a lot? Is it a little? The question can be answered only in differential terms, never absolutely. For the time being, then, this modest number is nothing more than the result of a cognitive formula which is evaluated when we form EPISODE and ACTION constructs during the comprehension of texts and the reading of ACTION. It is a formula that involves many different variables and dynamic linking processes of which we can be sure that only a fraction are known to us. Nevertheless, it is a fact that we can obtain results perfectly compatible with human intuition by applying our theoretical arguments and the practical model which culminates in the ACTION product which correlates real (empirically identified) EVENT constructs and algorithmically derived virtual EPISODE and ACTION constructs. The reader will see this for himself in the final part of our study, and if that fails to convince him, he can experiment further with EventParser and EpiTest on data of his own choosing.

He might, for example, follow the author and decide to test the programs by processing an extract from Lewis Caroll's *Alice in Wonderland* which was deliberately encoded to produce a narrative with a single uninterrupted strand. This extract represents the opposite of the above extracts from the *Fairy Tale*. Because of its typically episodic combination of high EVENT integration and low EPISODE integration, the *Fairy Tale* example yielded an ACTION product of 235.294. Compare this with the results of processing the *Alice.esf* file, which yields an ACTION product that is almost ten times higher than that of *Matrix.esf*. Like the latter, *Alice.esf* contains just ten EVENT constructs; but it permits the formation of an ACTION chain which contains a maximum of nineteen isochronally joined EPISODES.[10]

Practical Analysis Using EpiTest

In the preceding pages, we have described a first attempt to put the theory developed in our book *Computing Action. A Narratological Approach* into practice. Part 3 of that book describes the full-scale application of our software to a real text, Goethe's *Conversations of German Refugees*. However, it is well to remember that our model and methods are not without their limits. These weaknesses are particularly obvious when we come to deal with the anisochronous EVENT types. Readers can process the semantics of

[10] *Alice.esf* is included with the downloadable files.

anisochronous EVENT narratives effortlessly and intuitively because they use the framework of a prespecified comprehensive and well-structured knowledge context. This is very different from the incrementally assembled set of axioms which EventParser stores in its **.sbf* files..

The case of an anisonchronous EPISODE can illustrate the difficulties which can result. Assuming that narration (and therefore reading) is strictly sequential and does not employ embedding techniques, the prehistory supplied by EVENT_1 must be encountered after EVENT_2. The order in which the acts of reception (i.e. EVENT construction) take place is irrelevant in our theory and the EpiTest algorithm with which we are concerned. So far, so good. Now, EVENT_1 and EVENT_2 must have explicitly connectable predicates if they are to be linked semantically by EpiTest. The practical problem is that recipients tend to provide the EVENTS with such predicates only if they have mentally arranged the analeptically narrated EVENTS in the order of the fictive ordo naturalis beforehand. The EANISO_1 example encourages this, to be sure, but the bracketing and assistance it provides are the exception rather than the rule.

Narratologists typically treat the practical receptive necessity of returning to the ordo naturalis of the fabula as evidence that the ability to reconstruct a quasi-natural chronological order at the diegetic level is a necessary prerequisite for identifying a consistent action order. This may well be reasonable from the perspective of an intentional or causal definition of the concept of action, but things are different if EVENT, EPISODE, and ACTION are defined semiotically as dynamic constructs of reception. Every STATE OF AFFAIRS that we encounter in the empirical or fictional world is not just projectively configured in anticipation of a possible change which it could undergo in the future; it is also the base of a retrospective semantic protension which can make it the result of just such a change which has already taken place. Between the theoretical possibility of this semantically preconfigured construct and its concrete instantiation there lie many worlds, a marvellously comprehensive knowledge of the semantic relations which can accumulate through the symbolic representation of the phenomena of at least one possible world. In its present state, EpiTest has no access to the complex heuristic world knowledge of the semantic *ordo naturalis* which is constantly available to natural readers. Until this shortcoming is rectified, the rare occasions when EpiTest does combine anisochronous EVENTS into EPISODES and ACTIONS will be nothing more than flukes. In order to overcome this obvious weakness in EpiTest and the theory behind it, we would have to commit ourselves to using concepts from cognitive theory such as the script and the frame. These concepts postulate a schematically represented mental knowledge of standardized situational and pragmatic patterns which is constantly accessible to recipients; the knowledge comes into its own when the recipients need a heuristics with which to explain protensionally (anisochronally) arranged semantic EVENT terms.

In contrast to this ideal solution, we have employed something of a stopgap measure in the practical application of the EventParser and EpiTest programs to our example texts, the six individual narratives in Goethe's *Conversations of German Refugees*. A maximum of ten pairs of opposite terms have been formulated in advance for each of EventParser's nine heuristic categories. Taken together, these terms are an intuitively plausible model—albeit an admittedly crude one—of the knowledge context which I believe to be of crucial importance in the identification of anisochronous EPISODES. The results of the analysis are described in the third and final part of *Computing Action* after a discussion of the critical background against which our narratological study in literary computing makes sense.

In principle there are at least four different combinations of text and reader in which EventParser and EpiTest can be used to mark up and combinatorially analyse record files. Together, the four combinations represent a typology of action analysis; each of them concerns a different methodological problem:

(a) One text and one reader. In this relatively simple situation, we are usually concerned with differential analysis of the components of a single more complex text. A typical topic of study would be: what are the differences between the ACTION PRODUCTS of a set of sub-narratives embedded in a single overall narrative?

(b) One text and *n* readers. Research of this type usually analyses statistical patterns in how different readers interpret a single text. A possible research topic might be: given a single text and two groups of readers, how and why does the mean ACTION PRODUCT of the readers in one group one differ from that of the readers in the other?

(c) *N* texts and one reader. In this case, we have a single reader whose individual way of reading action provides a means of analysing historical developments in a genre or the works of a given author. It is basically a variant of (a); a typical theme might be: how do differences in ACTION PRODUCT relate to the early and late novels of a particular author?

(d) *N* texts and *n* readers. This case involves the differential analysis of a text sample using multiple readers. The approach might be most profitable in a study where action is just one aspect of a broader empirical analysis of the style of an individual author or movement.

As can be seen, the four different types share the use of differential analysis to explore empirical data. In *Computing Action. A Narratological Approach* I argue that it is impossible to find an essentialist definition of action, be the latter practical or aesthetic. There is no point in asking what action, even literary action, actually is. But it is certainly worthwhile asking how and why we believe that a certain amount of ACTION is present in a given narrative. It is this question which the programs EventParser and Epitest can help us to answer.

3. Installation Guide

EventParser and EpiTest have been designed for PC-DOS computers running Windows operating software. EventParser 4.37 has very modest system requirements and will normally work on a 486 Hz cpu machine with 8 MB RAM and an operating system from Windows 3.11 upwards (tested on 95, 98, 2000 and XP; on an ME platform the program was found to be instable.)

EpiTest 3.85 is decidedly more demanding. The minimum configuration on which it tested positively was an Intel Pentium P I 166 MHz, 32 MB RAM, Windows 95.2 (also tested successfully under Windows 98, NT, 2000 und XP on PII, III and IV machines). Depending on the hardware configuration certain restrictions apply in terms of the complexity of **.esf-files* handled and the length of episode- and action-lists available for inspection in the user interface. Under Windows 98 an attempt to read in **.esf-files* of more than 50 kb (i.e., more than approx. 50 events) or to inspect output files which may have been successfully generated by EpiTest, but exceed the display box memory limitation can lead to a program crash. However, no loss of data is to be expected. The EpiTest analyses discussed in part 3 of *Computing Action. A Narratological Approach* were initially run on a P II configuarion (266 Mhz, 32 MB RAM, Windows 2000) and then re-run on an AMD Thunderbird 800 MHz, 128 MB RAM and a P IV, 2.66 Ghz under Windows XP.

3.1 The Installation File *compact.exe*

The installation file *compact.exe* is a self-extracting file available at

http://www.jcmeister.de/downloads/software/comp-action/compact.exe

Clicking this file will start the automatic installation process (note that no registry entries are made). The default installation directory is

c:/compact

A detailed listing of all files extracted and stored during this process is found in section 3.3 of this installation guide.

Assuming that the default settings have been accepted the two programs will be extracted into the following directories:

- EventParser 4.37: saved as *ep437.exe* in **c:\compact\eventparser\**
- EpiTest 3.85: saved as *epitest385.exe* in **c:\compact\epitest\**

Desktop links and icons for both programs should be created manually. Note that after installation all extracted files must have the read and write properties activated – copying the files between a hard drive and a CD-ROM can lead to these properties being disabled in which case neither program will work properly.

3.2 Running EventParser 4.37

Having been written in a by now outdated version of Visual Basic (3.0) EventParser 4.37 can only handle *.tbf – text files of up to 45 kb size. Larger files will therefore have to be segmented. EventParser is started by executing the file named *ep437.exe*.

Known bugs:

1. Activating the option „Include pre-defined terms" in the KNOWLEDGE-BASE-menu *prior* to the first processing of a new **.tbf*-file leads to a conflict with the succeeding automatic creation of a new **.sbf*-file. – Solution: create an empty **.sbf*-file prior to invoking EventParser using a simple text editor (Notepad etc.) and save it in text-only format into the same directory as the associated **.tbf*-file which you wish to process. The **.sbf*-file root name must be identical to that of the **.tbf* ; e.g. *myfile.tbf* and *myfile.sbf*.

2. If during the declaration process for a new EVENT only a partially new definition of semantic terms is done – i.e., if the dispositional term is taken from an exiting term list and then combined with a newly introduced expositional term in the PREDICATE DEFINITION window that same term will reappear in the bottom right box when next opening the PREDICATE DEFINITION window. However, this is merely a display error; the old term will not be saved in the course of the second term declaration.

3. Whenever any character is entered into the text box on top of the term lists the program will interpret this as the definition of a meaningful new semantic term. Unless you manage to exit from the PREDICATE DEFINITION window the only way to identify nonsensical or accidental term entries is to attach an easily identifiable counter term ('x, WOW') allowing you to prune the **.esf*-file manually with a text editor at a later stage.

4. Under certain conditions the predicate definition routine can result in an underdefined semiotic square. This can only be corrected manually by checking the resultant *.esf-file with a text editor prior to processing the protocol file in EpiTest.

3.3 Running EpiTest 3.85

EpiTest 3.85 is started by running the file *epitest385.exe*. The program was written in LPA Prolog which conforms to the Edinburgh Standard. For details on this implementation of Prolog see

http://www.lpa.co.uk/abt.htm

After installing EpiTest you will find an initialisation file *epitest385.ini* in the relevant diretory. The command line in this file reads

```
command=/h4096 /i2048 /t1024 /D1 /01024 /p4096
```

and contains so-called 'program switches' that define details of Prolog memory management and dynamic assertion and manipulation of clauses during recursion loops (‚safe dynamic mode’). Please consult the LPA manual for details on how to manipulate these parameters.

Known bugs:

1. When accidentally skipping the routine for generating metaterms (which is normally executed by clicking the METASYNONYMS button) the program may be unable to generate episode or action constructs. This happens whenever the synonyms declared in EventParser lack explicit semantic connectivity. Usually the higher the overall number of semantic terms declared, the more likely they will possess explicit connectivity. By contrary, a low amount of terms may seem perfectly related to each other for the human reader who has access to metaphorical and world knowledge, but not to the machine which relies on explicit definitions only. – This characteristic of EpiTest is intended.
2. On termination of the program an accidental error message can sometimes appear. This can be ignored.
3. Within EpiTest the automatically generated **.con* and **.daf* output files can only be saved into the same directory as the **.esf* input file.

4. Installed files

After a successful installation the following files should be available:

Program	Directory	File name	Function
EventParser	\eventparser	**ep437.exe** Cmdialog.vbx Crystal.vbx Mscomm.vbx Msole2.vbx Threed.vbx	Program files These files must all be in the same directory. Some additional dll-files will be saved here as well.
	\Alice \Fairytale \Matrix \Unterhaltungen	Standard.lbf Standard.sbf	Standard seme term files These contain default semantic terms and may be changed or augmented.
		*.tbf	TextBaseFile Text files to be processed; max size is 45 kb.
		*.sbf	EventParser files which are automatically created.
		*.lbf	
		*.ebf	

EpiTest		*.esf	Integrated protocol file which EventParser creates automatically by combining *.sbf and *.ebf-files.
EpiTest	\listings	ep generator3.85.pl epitestlisting.doc	Uncompiled PROLOG-file and source code listing.
EpiTest	\epitest	**epitest385.exe** epitest385.ovl	Program files
		epitest385.ini	Initialisation file
	\Alice \Fairytale \Matrix \Unterhaltungen	*.esf	Integrated protocol file created by EventParser for subsequent processing with EpiTest.
		metaterms.msf	Temporary file listing newly generated meta-terms.
		*.con	Construct file listing all EPISODE und ACTION-constructs generated by EpiTest.
		*.daf	Data analysis file containing the statistical results of the combinatory exploration of an *.esf-file with EpiTest.

Please note: when manipulating any *.sbf, *.lbf, *.ebf or *.esf-file make sure to insert a single line feed at the end of the file. This is a crucial syntax requirement for list files – ignoring it will result in a program crash!