Semantic Logging of Repeating Events in a Forward Branching Time Model

Valerian Ivashenko
Intellectual Information Technologies Department
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
ivashenko@bsuir.by

Nikita Zotov
Intellectual Information Technologies Department
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
nikita.zotov.belarus@gmail.com

Maksim Orlov
Intellectual Information Technologies Department
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
orlovmassimo@gmail.com

Abstract. **The tasks of knowledge logging in the form of semantic networks of the model of the unified semantic knowledge representation are considered. The formal model of a semantic log of repeating events in knowledge processing and algorithms for adding and retrieving logged events from the log are presented. The spatial-time structure of logged processes should satisfy a forward branching time model.**

Keywords: **semantic logging, repeated events, semantic networks, branching time, full persistence, unified semantic knowledge representation model, multi-agent system, cognitive architecture**

I. Introduction

Semantic logging (SL) is a mechanism designed to provide the intellectual system with introspective capabilities in order to endow it with the qualities of an artificial consciousness, including the ability to explain one's own work and its results [1]. It should take into account the non-factors of knowledge [3] including its incompleteness, uncertainty, hypotheticality as well as requirements for working in real time [10]. SL consists in recording in the knowledge representation language, for example, in the form of a semantic network, a knowledge about an order (system) of actions and events (phenomena) occurred in processes of knowledge processing.

II. Overview

SL can be considered as one of the approaches to a process introspection [6]. SL approach was descrbied in several previous works [4, 5, 9–11, 16]. From the point of view of the becoming structure or structure of time, there are several types of process time models: linear time, tree or branching time, directed acyclic time structure, arbitrary time structure [2, 9–11, 13]. From the point of view of the data structure processing there are also several types of persistent data structures (DS): partially persistent DS, fully persistent DS, confluently persistent DS [7, 9]. One of the approaches to achieve DSs persistence (structures when some changes are made to them retain all their previous states and access to those states) is using CoW data structures [8].

There are also several points of view for knowledge processing (KP) in multi-agent an AI systems: concurrent and distributed KP [6] without a certain global state or common knowledge base such as in actor model; coordinated and partially synchronized KP via such means as blackboard systems [15], fully synchronized KP using global states as in state space search models and algorithms (using various logics) [2].

Some algorithms and models have been proposed for acyclic process structures [9–11]. All considered models do not takeevents repetitions into account.

Their basic algorithms solve mostly SL generation and information retrieving tasks.

A. Basic procedures

1) Generation: The following log (pre-existing and possibly empty) generation algorithm (Fig. 1) pushes pre-created event identifiers with links providing full persistence of the structure of the semantic log as a set of its integrated versions.

```
⟨LE, log, e, n⟩ ←
  if (undefined (log))
  ⟨log, g⟩ ← createLeafLink (⟨LE, log, e, n, 2⟩)
  else
  l ← getFirstLink (log)
  c ← getCoefficient (⟨log, l⟩)
  ⟨log, g⟩ ← createLeafLink (⟨LE, log, e, n, 2/c⟩)
  while (hasNextLink (⟨log, l⟩) ∧ (c = 1))
    ⟨f, p, c⟩ ← ⟨l, getNextLink (⟨log, l⟩), 1⟩
    l ← p
    if (hasNextLink (⟨log, l⟩))
      l ← getNextLink (⟨log, l⟩)
      c ← getCoefficient (⟨log, l⟩)
    g ← createPair (⟨log, f, p, 2/c, g⟩)
  g ← appendLink (⟨log, l, g⟩)
  ← log
```

Fig. 1. Log link generation algorithm

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

2) Retrieving information: There are two basic searchalgorithms in such SL structures (Fig. 2, 3).

```
⟨log, i⟩ ←
⟨l, c, s⟩ ← ⟨getFirstLink (log) , 1, 1⟩
j ← getIdentifier (⟨log, l⟩)
while (hasNextLink (⟨log, l⟩) ∧ less (⟨i, j⟩))
| n ← getNextLink (⟨log, l⟩)
| ⟨d, k⟩ ← ⟨n, 1⟩
| while (k < c) ⟨k, d⟩ ← ⟨k + k, getDownLink (⟨log, d⟩)⟩
| j ← getIdentifier (⟨log, d⟩)
| if (¬less (⟨j, i⟩))
| | s ← s + c
| | c ← c ∗ getCoefficient (⟨log, l⟩)
| | l ← n
while (1 < c)
| l ← getDownLink (⟨log, l⟩)
| d ← l
| if (hasNextLink (⟨log, l⟩))
| | n ← getNextLink (⟨log, l⟩)
| | d ← n
| ⟨c, k⟩ ← ⟨c/2, 1⟩
| while (k < c) ⟨k, d⟩ ← ⟨k + k, getDownLink (⟨log, d⟩)⟩
| j ← getIdentifier (⟨log, d⟩)
| if (¬less (⟨j, i⟩)) ⟨s, l⟩ ← ⟨s + c, n⟩
if (less (⟨j, i⟩) ∨less (⟨i, j⟩)) ← s
← ⟨s, l⟩
```

Fig. 2. Get event by identifier algorithm

```
⟨log, n⟩ ←
⟨l, c, s⟩ ← ⟨getFirstLink (log) , getCoefficient (⟨log, l⟩) , 1⟩
while (hasNextLink (⟨log, l⟩) ∧¬(n < s + c))
| ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
| c ← c ∗getCoefficient (⟨log, l⟩)
c ← c/getCoefficient (⟨log, l⟩)
while (1 < c)
| ⟨l, c⟩ ← ⟨getDownLink (⟨log, l⟩) , c/2⟩
| if (hasNextLink (⟨log, l⟩) ∧¬(n < s + c))
| | ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
if (s = n) ← getEvent (l)
← nothing
```

Fig. 3. Get event by index algorithm

B. Applied tasks and questions

These algorithms are the platform to solve more complicated problems and applied tasks. There are several levels of such tasks and problems: level of relations between events and their repetitions [9]; level of spatial-time relations between phenomenons [13]; level of applied tasks of analysis and synthesis of external and internal phenomenons. The problems of the first level include problems of determining: event anteriority, primary event (Fig. 4) and last common event (LCE). For mentioned models, there are four cases of anterioity of two events: event coincidence, event alternativeness (synchronicity), the first event antecedence, the second event antecedence [2]. Tasks of the third level are the memorizing and supervising of internal and external processes, process mining [6] including history analysis and navigation, process reproduction (inductive programming). These tasks can arise in such areas as education, software development and interaction within control version systems, (digital) music composition and others [4, 6, 12, 15]. As for history navigation as a general applied tasks, let's consider the following approach. Let's consider an agent named "locator". This agent has five properties: two initial events (or its repetitions) – minor and major, two margin events (or event repetitions) – minor and major, chosen event (event repetition). Also this agent has behavior implemented by several procedures: initializing initial, margin and chosen events; transferring a chosen event to a user (another agent); updating minor or major margin event by the chosen event and choosing new event; updating minor margin event either by the major margin event or by the minor initial event and choosing new event; updating major margin event either by the minor margin event or by the majorinitial event and choosing new event.

```
log ←
⟨l, c, s⟩ ← ⟨getFirstLink (log) , 1, 1⟩
while (hasNextLink (⟨log, l⟩))
| c ← c ∗getCoefficient (⟨log, l⟩)
| ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
while (1 < c)
| ⟨l, c⟩ ← ⟨getDownLink (⟨log, l⟩) , c/2⟩
| if (hasNextLink (⟨log, l⟩))
| | ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
← s
```

Fig. 4. Primary event log index determining algorithm

III. PROPOSITION

A. Repetition logging

Previous models did not take into account the repetitions of events. Event (periodic) repetitions can be interpreted as event occurrences in (locally) cyclic spatial-temporal model. It is important to admit that all further events under consideration will have no more than one immediate predecessor. In the case of repeating events, the previous models are inappropriate. Thus, we need to identify event repetitions but not events. Therefore, we have many repetition identifiers for one event. ll of them should be enumerated and distinguished by an event repetition number. Another problem is that we might want to know, "Is there any event occurrence in the log or not?" without any interest in a particular event repetition. To solve the last problem (occurrence problem), global (event) occurrence identifiers (GOI) can be used. For an event, its first event repetition identifier may be used as a GOI. Thus, a global occurrence number is an event repetition number which equals 1 for each event. If we have a log

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

structure with partial persistence only then these are all essential differences.

The Fig. 5, 6 show the basic algorithms for KP under a log structure where event repetitions are taken into account. Algorithm for getting a last event repetition number can be obtained replacing *getCoefficient*(⟨*log, l*⟩) calls at Fig. 4 by 2.

```
⟨log, r⟩ ←
⟨l, c, s⟩ ← ⟨getFirstLink (log) , 1, 1⟩
while (hasNextLink (⟨log, l⟩))
| c ← c + c
| ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
⟨t, k⟩ ← ⟨l, c⟩
while (1 < c)
| ⟨l, c⟩ ← ⟨getDownLink (⟨log, l⟩) , c/2⟩
| if (hasNextLink (⟨log, l⟩))
| | ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
| else ⟨t, k⟩ ← ⟨l, c⟩
l ← newNextLink (⟨log, t⟩)
while (1 < k) ⟨k, l⟩ ← ⟨k/2, newDownLink (⟨log, l⟩)⟩
← appendRepetitionLog (⟨log, l, r⟩)
```

Fig. 5. Algorithm of the queuing of an event repetition

```
⟨log, n⟩ ←
⟨l, c, s⟩ ← ⟨getFirstLink (log) , 1, 1⟩
while (hasNextLink (⟨log, l⟩) ∧¬(n < s + c + c))
| c ← c + c
| ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
while (1 < c)
| ⟨l, c⟩ ← ⟨getDownLink (⟨log, l⟩), c/2⟩
| if (hasNextLink (⟨log, l⟩))
| | ⟨s, l⟩ ← ⟨s + c, getNextLink (⟨log, l⟩)⟩
if (s = n) ← getOccurenceStructure (l)
← nothing
```

Fig. 6. Algorithm of the getting of occurrence structures by number of anevent repetition (getOccurrences)

B. Full persistence problems

However, if we work with full persistence SL structures then there will be a set of other problems. There will be: enumerating all repetitions of a certain event in the log (for partial persistence it can be efficiently solved with log-specific event repetition identifiers and additionally segregated enumeration queues), an occurrence problem that requires more complex solution than for a partial persistence SL structure.

C. Model

SL model of repeating events having no more than one immediate predecessor is defined by (1) and satisfies (2).

$$\langle e, d, l, i, g, o, r, h, n, p, P, R\rangle \in E_+^{I} \times S_+^{E\times \mathrm{N}} \times \times L^{S} \times I^{S} \times P^{E} \times \{\bot, \top\}^{S\times P} \times R_+^{S\times P} \times \times I_+^{L\times \mathrm{N}} \times \mathrm{N}_+^{L\times \mathrm{N}} \times \mathrm{N}_+^{L\times I} \times 2^{I} \times 2^{L} \quad (1)$$

where E – events, I – identifiers, S – descriptors structures with occurrence log sets (implemented as CoW critbit trees), L – logs, P – primary (occurrence) identifiers, R – repetition sublogs, h – event identifier by index mapping, n – repetition number by index mapping, p – index by identifier mapping.

$$g = (d \cap ((E \times \{1\}) \times S)) \circ i. \quad (2)$$

IV. Computation Complexity Analysis

It can be shown also by experimental confirmations that generation algorithms have $O(log(n)\, f(n))$ time complexity. Whereas time complexities of basic search algorithms do not exceed $O(log^2(n) * f(n))$, where $f(n)$ – time of the access to an element of logs.

V. Efficient Solvable Problems and Application

Anteriority and LCE problems for repeating events that satisfy given restrictions can be efficiently solved with followingalgorithms (Fig. 7, 8 and 9).

```
⟨e, n⟩ ←
log ← getRepeatedEventLog (⟨e, n⟩)
if (defined (log))
| ← getIdentifier (getFirstLink (log))
← emptyIdentifier
```

Fig. 7. Algorithm for getting an event repetition identifier (getRepeatedEventIdentifier)

```
⟨e, n, a, m⟩ ←
l ← getRepeatedEventLog (⟨e, n⟩)
g ← getRepeatedEventLog (⟨a, m⟩)
i ← getRepeatedEventIdentifier (⟨e, n⟩)
j ← getRepeatedEventIdentifier (⟨a, m⟩)
if (coincide (⟨e, a⟩))
|← coincidence
if (interdependent (⟨e, a⟩))
|← interdependence
if (less (⟨i, j⟩))
| if (retrieveREByIdentifier (⟨g, i⟩))
| |← antecedenceFirst
| else
| if (less (⟨j, i⟩))
| | if (retrieveREByIdentifier (⟨l, j⟩))
| | |← antecedenceSecond
| | else
| |← coincidence
← potentialAlternativeness
```

Fig. 8. Anteriority determining algorithm

These algorithms are primarily oriented to navigation task applications. Anteriority problem

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

seems to be more complicated for cyclic process structures. That is particularly why, there are five cases of anterioity of two events: event coincidence, event alternativeness (synchronicity), the first event antecedence, the second event antecedence, interdependency (antecedence for both of events) (Fig. 8) [9–11, 13]. To solve anteriority problem for the last case, authors suggest to combine an incremental structures for merge-find set approach [14] and the proposed SL models.

```
⟨e, n, a, m⟩ ←
x ← getPrimaryRepeatedEventIndex (⟨e, n⟩)
y ← getPrimaryRepeatedEventIndex (⟨a, m⟩)
l ← getRepeatedEventLog (⟨e, n⟩)
i ← retrieveREByIndex (⟨l, x⟩)
i ← getRepeatedEventIdentifier (i)
g ← getRepeatedEventLog (⟨a, m⟩)
j ← retrieveREByIndex (⟨g, x⟩)
j ← getRepeatedEventIdentifier (j)
k ← getRepeatedEventIdentifier (⟨a, m⟩)
h ← getRepeatedEventIdentifier (⟨e, n⟩)
if (less (⟨i, j⟩) Vless (⟨j, i⟩) Vless (⟨k, i⟩) Vless (⟨h, j⟩))
|← nothing
if (less (⟨x, y⟩)) ⟨c, d, p⟩ ← ⟨⟨e, n⟩, ⟨a, m⟩, x⟩
   else ⟨c, d, p, l⟩ ← ⟨⟨a, m⟩, ⟨e, n⟩, y, g⟩

|⟨q, r⟩ ← ⟨1, 1⟩
while (r < p)
|  k ← retrieveREByIndex (⟨l, r⟩)
|  if (anteriority (⟨d, k⟩) = antecedentFirst) p ← r
|       else
|
|  |q ← r
|  r ←⌈(p+q)/2⌉
← retrieveREByIndex (⟨l, r⟩)
```

Fig. 9. Last common event repetition determining algorithm

VI. Conclusion

The proposed model and algorithms are able to provide prompt response to the main issues related to the study of the order of logged events and their repetitions. Basic algorithms were implemented using an integration platform which is the implementation platform for the reference-testing system [15]. The use of implemented algorithms is focused on supporting the automation of interaction with the knowledge base with reference materials and analyzing the history of responses by the user to test questions.

References

[1] D. McDermott et al. Mind and Mechanism. Cambridge (Mass), MIT Press. (xv +), 2001, pp. 262.

[2] J. F. Allen, Time and time again: the many ways to represent time, Intern. J. of Intelligent Systems, 6, 1991, pp. 341–355.

[3] A. S. Narinyani, Non-factors: inaccuracy and underdetermination – difference and interrelation. Izv. RAN (RAS). Ser. Teoriya i sistemy upravleniya, 5, 2000, pp. 44–56 (in Russian).

[4] Sorin Ilie, Mihnea Scafes, Costin Badica, Thomas Neidhart, Rani Pinchuk. Semantic logging in a distributed multi-agent system. 2010.

[5] G. Pavlin, M. Kamermans, and M. Scafes. Dynamic process integration framework: Toward efficient information processing in complex distributed systems. In Proceedings of 3rd International Symposium on Intelligent Distributed Computing – IDC'2009, vol. 237 of Studies in Computational Intelligence, Springer, 2009, pp. 161–174.

[6] W. Gaaloul, K. Gaaloul, S. Bhiri, A. Haller, M. Hauswirth. Log-based transactional workflow mining. Distributed and Parallel Databases 25, 2009.

[7] J. R. Driscoll, D. D. Sleator, R. E. Tarjan, Fully Persistent Lists with Catenation. J. ACM 41(5), 1994, pp. 943–959.

[8] O. Rodeh. B-Trees, Shadowing, and Clones. ACM Transactions on Storage 3 (4), 2008, 1.

[9] V. P. Ivashenko, Semantic logging of knowledge processing. Proceedings of the workshop "Information Technologies and Systems 2017 (ITS 2017)", Minsk, Belarus, 25 October 2017, pp. 110–111 (in Russian).

[10] V. P. Ivashenko. Algorithms for semantic logging of knowledge processing, BIG DATA & Advanced Analytics, Minsk, Belarus, 3-4 May 2018, pp. 267-273. (in Russian).

[11] V. P. Ivashenko, Semantic logging of knowledge processing based on binary generated. PRIP, 2019, pp. 172–177.

[12] D. A. Pospelov, Situational management: theory and practice, Nauka, Moskva, 1986, p. 288. (in Russian).

[13] V. P. Ivashenko. Ontological model of space-time relations for events and phenomena in the processing of knowledge. Vestnik BrGTU 5 (107), 2017, pp. 13-17. (in Russian).

[14] Z. Galil, G. Italiano. Data structures and algorithms for disjoint set union problems. ACM Computing Surveys 23 (3), (1991), pp. 319—344.

[15] V. P. Ivashenko, Reference and testing system based on the unified semantic representation of knowledge, ITS 2020, 2020, pp. 80–81. (in Russian).

[16] C. Ringelstein, S. Staab. Logging in distributed workflows. Proceedings of the ISWC'07 Workshop on Privacy Enforcement and Accountability and Semantics (PEAS 2007), 320, 2008, pp. 19–30, 2007.

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.