- $n_{s_{l_v}} \in (\bar{\mathbb{N}} \cup \{0\})$ — the index of the last vacant segment in sc-memory;
- $n_{s_{l_r}} \in (\bar{\mathbb{N}} \cup \{0\})$ — the index of the last released segment in sc-memory;
- $m_s \in M$ — the object that synchronizes access to S, $n_{s_{l_e}}$ and $n_{s_{max}}$;
- $m^n_{s_{l_r}} \in M$ — the object that synchronizes access to $n_{s_{l_r}}$;
- *SSPI*={*engage*, *release*} — internal programming interface of sc-elements storage in sc-memory.

All allocated segments S may be free $S_f \subseteq S$ or engaged $S_e \subseteq S$. The set of free sc-memory segments S_f includes the set of vacant segments $S_v \subseteq S_f$ and the set of released segments $S_r \subseteq S_f$.

Cells in sc-memory segments E may be free $E_f \subseteq E$ and engaged $E_e \subseteq E$. The set of free sc-memory cells $E_f = \{{e^{ij}}_f |\ {e^{ij}}_f \in {s^i}_f,\ {s^i}_f \in S_f\}$ includes the set of vacant cells $E_v \subseteq E_f$ and the set of released cells $E_r \subseteq E_f$.

Consequently, the following statements hold for all sc-memory segments and cells in them:

- $E_f \cup E_e = E, E_f \cap E_e = \varnothing$,
- $E_v \cup E_r = E_f, E_v \cap E_r = \varnothing$,
- $S_f \cup S_e = S, S_f \cap S_e = \varnothing$,
- $S_v \cup S_r = S_f,\ S_v \cup S_r \subseteq S_f$.

For the sets of engaged and released cells, the corresponding transitions can be defined in the form of:

- operation of allocating a *engage* : $E_f \to E_e$, which changes the state of the cell from "released" to "engaged":

$$engage() = \begin{cases} e_{ij} \in E_e, & if \exists e_{ij} \in s_i, E_f \wedge n \leq n_{s_{max}}, \\ \text{Error}, & if\ n > n_{s_{max}} \vee |E_f| = 0; \end{cases}$$

- operation of releasing a cell in sc-memory segment *release* : $E_e \to E_f$, which changes the state of the cell from "engaged" to "released":

$$release(e_{ij}) = \begin{cases} e_{ij} \in E_f, & if\ e_{ij} \in E_e, \\ \text{Error}, & if\ e_{ij} \notin E_e. \end{cases}$$

The algorithm of the cell engaging operation in sc-memory segment *(engage)* can be described as follows:

- Step 1: Try to find any vacant segment S_i in the set S_v:
  - If such a segment exists, go to step 2.
  - If no such segment exists, skip to step 3.
- Step 2: Engage a new cell in the found vacant segment s_i:
  - Increase $n_{e_{le}}$ in segment s_i by 1.
  - Occupy the cell e_{ij} with index $n_{e_{le}}$ in segment s_i.
  - Return the address of the engaged cell e_{ij} and terminate.
- Step 3: Attempt to get a new segment from set S:
  - If the number of engaged segments $n_{e_{le}}$ is less than the maximum $n_{s_{max}}$, create a new segment s_i (set $n_{e_{le}}$ as 0) and add it to S.
    * Increase $n_{e_{le}}$ in segment s_i by 1.
    * Occupy the cell e_{ij} with index $n_{e_{le}}$ in segment s_i.
    * Return the address of the engaged cell e_{ij} and terminate.
  - If the maximum number of segments $n_{s_{max}}$ is reached, go to step 4.
- Step 4: Try to get the released segment s_i from the set S_r:
  - If there is no such segment, report an error and terminate.
  - If such a segment exists, proceed to step 5.
- Step 5: Engage a new cell in the found released segment s_i:
  - Get the last released cell e_{ij} by its number in segment n_{e_r}.
  - Occupy the cell e_{ij} with index n_{e_r} in segment s_i.
  - Update n_{e_r} for the next released cell.
  - Return the address of the engaged cell e_{ij} and terminate.

The algorithm of the cell releasing operation in sc-memory segment (*release*) can be described as follows:

- Step 1: Verify the correctness of the given address of cell e_{ij} :
  - If the cell address does not exist in sc-memory, terminate with an error.
  - If the cell address exists in sc-memory, proceed to step 2.
- Step 2: Using the cell address, determine the corresponding segment e_i of the cell in sc-memory and proceed to step 3.
- Step 3: Release the cell e_{ij} :
  - Update the information about the cell e_{ij} , mark it as released.
  - Update the number of the last released cell in segment n_{e_r}.
  - Go to step 4.
- Step 4: Update the information about the released segments.
  - If the released cell was the first released cell in the segment, update the information of the last released segment in sc-memory $n_{s_{lr}}$.
  - Go to step 5.
- Step 5: Terminate.

The described algorithms may include synchronization mechanisms to ensure data integrity during multithreaded sc-memory accesses. All synchronization operations in these algorithms can be performed using ap-

propriate synchronization objects m_s, m_e, ${m^n}_{s_{lv}}$. The model of the synchronization object will be discussed later.

Basically, the advantages of the described algorithms are due to the advantages of the cell engaging algorithm in sc-memory, which are as follows:

- The cell engaging algorithm in sc-memory tries to find a vacant segment before creating a new one. If there is no vacant segment, it tries to create a new one if the maximum number of segments has not been reached. This approach avoids wasting memory on creating unnecessary segments.
- By updating the number of the last released cell in a segment, the algorithm keeps track of which cells are available for reuse. This tracking ensures that the engaging process is fast and does not require searching the entire memory for a released cell.

To analyze the complexity of these algorithms, let us consider their main characteristics:

- In the algorithm of cell engaging operation in scmemory:
  - finding a vacant segment and engaging a cell in itrequires traversing many segments and checking their status, which can be accomplished in a time proportional to the number of segments;
  - creating a new segment and selecting a cell in it requires a fixed number of operations independent of the data size;.
  - finding a released segment also requires traversing multiple segments.
- In the algorithm of cell releasing operation in scmemory:
  - checking the correctness of the cell address and determining the appropriate segment can be accomplished in a time proportional to the number of segments, or faster if an efficient data structure is used to store the segments;
  - releasing a cell and updating segment information requires a fixed number of operations.

The complexity of the algorithms depends on the data structures used and how they are processed. Assuming that multiple segments are processed efficiently, for example using internal lists in released segments and released segment cells, the underlying complexity of the algorithms will be determined by the number of operations required to process each step. Thus:

- The algorithm of cell engaging operation in scmemory can have a complexity from $O(1)$ to $O(n)$, where n is the number of segments, depending on whether a free segment is found or a new segment needs to be engaged;
- The algorithm of cell releasing operation in scmemory basically has a complexity of $O(1)$, since most operations are performed in a fixed amount of time, except for address correctness checking, which may require $O(n)$ in the worst case

So, an sc-element storage is a set of cells, each of which can store some sc-element (be engaged) or can be empty (free):

$$(\forall e \in E : (e \in E_e) \vee (e \in E_f)).$$

Each cell $e_{ij} \in E$, $e_{ij} \in s_i$,$s_i \in S$ has a unique internal address $a = \langle i, j \rangle \in A$. That is, the following statement always holds:

$$(\forall e_{ij} \in E, \exists! a \in A : (e_{ij} \in N) \wedge (s_i \in S) \wedge (a = \langle i, j \rangle))$$

Each cell stores either an sc-node N or an sc-connector C:

$$(\forall e \in E : (e \in N) \vee (e \in EC)),$$
$$N \cup C = E, N \cap C = \varnothing.$$

It is assumed that if a cell stores some sc-element, it stores information characterizing this sc-element. Each cell in sc-memory $e \in E$ can be represented as a tuple:

$$e = \langle FI, EI, CI \rangle,$$

where

- FI — characteristics of the stored sc-element, including the type of sc-element and its states,
- EI — information about sc-elements incident with the given sc-element,
- CI — information about the number of incoming and outgoing sc-connectors for a given sc-element.

The characteristics of an sc-element F I can be represented as:

$$FI = \langle t, s \rangle,$$

where

- $t \in T$ is the syntactic type of a given sc-element (e.g., sc-node, sc-connector, ostis-system file,etc.),
- $s \in ES$ is the state of the cell (e.g.,"engaged","free", etc.).

$$T = T_n \cup T_c,$$

$$T_n = \{node, file\}, T_c = \{connector, arc, edge\},$$

where

- T is the set of all possible syntactic types of scelements;
- T_n is set of all possible syntactic types of sc-nodes, *node, file* — actual sc-node label and ostis-system file label, respectively;
- T_c is the set of all possible syntactic types of sc-connectors, *connector, arc, edge* — the actual sc-connector label, sc-arc label, and sc edge label, respectively;

$$ES = \{engaged, free\},$$

where

- ES is the set of all possible cell states;
- $engaged$ — the cell is "engaged";
- $free$ — the cell is "free".

Information about incident sc-connectors EI can be represented as:

$$(\forall n \in N : (n \ni EI = \langle b_o,\ b_i \rangle)),$$

$$(\ \forall\ \mathrm{n} \in C :(c \ni EI =$$

$= \langle\ b_o,\ b_i,\ \mathrm{b},\ \mathrm{e},\ n_{bo},\ p_{bo},\ \ n_{bi},\ p_{bi},\ n_{eo},\ p_{eo},\ \ n_{ei},\ p_{ei}\ \rangle))$, where

- $b_o \in A$ is the sc-address of the initial sc-connector outgoing from the given sc element,
- $e_i \in A$ is the sc-address of the initial sc-connector incoming into the given sc-element,
- $b \in A$ is the sc-address of the initial sc-element of the sc-connector,
- $e \in A$ is the sc-address of the final sc-element of the sc-connector,
- $n_{bo} \in A$ is the sc-address of the next sc-connector outgoing from the initial sc-element,
- $p_{bo} \in A$ is the sc-address of the previous scconnector outgoing from the initial sc-element,
- $n_{bi} \in A$ is the sc-address of the next sc-connector incoming into the initial sc-element,
- $p_{bi} \in A$ is the sc-address of the previous scconnector incoming into the initial sc-element,
- $n_{eo} \in A$ is the sc-address of the next sc-connector outgoing from the final sc-element,
- $p_{eo} \in A$ A is the sc-address of the previous sc-connector outgoing from the final sc-element,
- $n_{ei} \in A$ A is the address of the next sc-connector incoming into the final sc-element,
- $p_{ei} \in A$ is the address of the previous sc-connector incoming into the final sc-element.

In this case, the following statements are true:

$$(\forall e \in E, \exists! b_o, b_o \in A : (Inc(e, b_o) \wedge Inc(e, b_i))),$$

$$(\forall\ e \in C, \exists\ !\ n_{bo}, p_{bo}\ ,\ n_{bi},\ p_{bi} \in A :$$

$$((Inc(e,\ n_{bo}) \wedge (Inc(e,\ n_{bi}) \wedge (Inc(e,\ n_{bi}) \wedge (Inc(e,\ p_{bi})))),$$

$$(\forall\ e \in C, \exists\ !\ n_{eo}, p_{eo}\ ,\ n_{ei},\ p_{ei} \in A :$$

$$((Inc(e,\ n_{eo}) \wedge (Inc(e,\ p_{eo}) \wedge (Inc(e,\ n_{ei}) \wedge (Inc(e,\ p_{ei})))),$$

where Inc is the binary relation of incidence of two sc-elements.

Information about the number of incoming and outgoing sc-connectors C can be represented as:

$$CI = \langle c_{in}, c_{out} \rangle$$

where

- $c_{in} \in (\bar{\mathbb{N}} \cup \{0\})$ is the number of incoming sc-connectors in a given sc-element,
- $c_{out} \in (\bar{\mathbb{N}} \cup \{0\})$ is the number of outgoing sc-connectors from a given sc-element.

This information can be used to optimize the isomorphic search for sc-constructions over a given graphtemplate [2].

The model of storage of sc-elements in sc-memory provides:

- storage of sc-constructions, their sc-elements, characteristics and incident relations between them;
- ability to create, modify, search and delete sceleements.

The advantages of this model are as follows:

- it provides efficient fragmentation and defragmentation of cells;
- algorithms for allocating and freeing a memory segment have asymptotic complexity from $O(1)$ to $O(n)$, where n is the number of segments that must be traversed to find a free segment.

B. Model of storage of external information constructions in sc-memory

The model of storage of external information constructions in sc-memory can be represented as

$$FS = \langle\ CH,\ M_s,\ n_{ch_{le}},\ n_{ch_{max}},\ m_{ch},\ tr$$

$$TSO,\ SOF,\ FSO,\ FSPI\ \rangle, \text{ where}$$

- $CH = \langle ch_1,\ ch_2,\ ...,\ ch_i,\ ...,\ ch_n,\ \rangle,\ i = \overline{1, n}$ is the sequence of dynamically allocated file segments in sc-memory of fixed size n;
- $ch_i = \{\ \langle\ \langle\ l_{s_{i1}},\ s_{i1}\ \rangle,\ ...,\ \langle\ l_{s_{ij1}},\ s_{i1}\ \rangle,\ ...\ \langle\ l_{s_{im}}, e_{im} \rangle \rangle,\ n_{sl},\ m_s \}$, $j = \overline{1, m}$ — the i-th file segment of fixed size m, consisting of cells – pairs of string lengths $s_{s_{ij}}$ and strings themselves $s_{ij} \in STR$,
- $n_{s_l} \in (\bar{\mathbb{N}} \cup \{0\})$ — the index of the last engaged cell in the file segment ch_i,
- $m_s \in M$ — the object that synchronizes access to $n_{s_{le}}$;
- $M_s \subseteq CHS \times M$ — a dynamic oriented set of file and cell pairs and their corresponding synchronization objects;
- $n_{ch_{le}} \in (\bar{\mathbb{N}} \cup \{0\})$ — the index of the last engaged file segment in sc-memory ($n_{ch_{le}} = n$),