УДК 004.04

Application Density and Feasibility Checking in Real-Time Systems

Sergey Baranov (SPIIRAS, ITMO University),
Victor Nikiforov (SPIIRAS)

An approach to analyze the compatibility of real-time multi-task applications with various combinations of scheduling modes and protocols of access to shared resources when run on multi-core platforms is described. It is based on the recently introduced notion of application density derived from estimation of application feasibility for various values of the processor performance. The software architecture of a relatively simple simulation tool for estimation of the task response time (and therefore, application feasibility) is described, which provides more exact data compared to the known analytical methods when they are applicable. Results of running this tool on a number of benchmarks, including balanced Liu-Layland configurations, are presented along with their analysis and interpretation. The suggested approach allows to indentify an optimal combination of the scheduling mode and access protocol for the given application structure.

Keywords: *simulation, real-time, application density, application feasibility.*

1. Introduction

Software applications for real-time systems (RTS) are usually built as collections of prioritized tasks treated as sequential programs closed w.r.t. to control flow. During application runs the tasks may share common system resources: executive ones (processors and processor cores of multi-core processors) and informational ones – global data arrays, interface registers of peripheral devices, elements of human-machine interface, etc. Access to executive resources is governed by scheduling mode in use, while access to shared informational resources is controlled by access protocols.

Each task τ_i of an RTS application is characterized by its period T_i, its deadline D_i, and its weight W_i. The period and the deadline are specified in absolute time units (e.g., milliseconds) and are usually considered as "external constraints" for the application, while the weight is an "internal constraint". It determines the "amount of computational work" (and thus, the quantity of the executive resource) which is needed for the task to accomplish its function and is specified in the number of standard machine operations for the given task realization. Given a particular processor performance P, the task weight W_i may be converted into time units: $C_i = W_i/P$.

Application behavior is composed by consecutive task instances $\tau_i^{(j)}$ (called jobs) running in parallel and iteratively activated according to their periods T_i. Due to competition among running jobs for platform resources, execution of any job $\tau_i^{(j)}$ may be suspended and then resumed after some period of time. The task response time R_i is the maximal time interval between respective jobs $\tau_i^{(j)}$ starts and terminations which is called "job existence interval".

A key requirement to an RTS software application is called "application feasibility" and is formulated as: $\forall i\ Di \geq R_i$ in all acceptable scenarios of application communication with its environment at the run time. Application feasibility may be checked either through an analytical estimation of the response time for each application task, or through simulation of the application run with an appropriate software tool.

For an RTS designed to run on a single-core processor exact analytical estimations of its feasibility exist since early 70-ies [1-3]; however, for multi-core processors exact analytical estimates are still unknown, while suggested rough methods provide pessimistic results if compared with real RTS behavior[4,5]. Therefore, a software simulation tool is needed to obtain a more exact estimation of application feasibility for RTS on multi-core platforms under various combinations of scheduling modes and access protocols than the known analytical methods. The paper describes the architecture of such software tool [6] and new results of a series of experiments with it.

2. Application Density

A derivative parameter characterizing the task τ_i behavior for the given performance P of processor cores is its utility $U_i = C_i/T_i$, which is the portion of the task period used for computing; thus the overall utility U of the application is: $U = (\Sigma_{1< i <n} U_i)/k$, k being the number of processor cores in the given platform, each of the same performance P. The value $1–U$ specifies the portion of the processor time not used by the application (the processor is either idle or is loaded with calculations unrelated to the RTS processing). An increase of the processor performance leads to an increase of the processor idle time for the given RTS software application.

Let's consider an auxiliary task characteristic called hardness: $H_i = Ti/D_i$. If $H_i \geq 1$ then existence intervals of any two consecutive instances of the same task τ_i – jobs $\tau_i^{(j)}$ and $\tau_i^{(j+1)}$ do not intersect. The reverse condition $H_i < 1$ means that existence intervals of such jobs may intersect. If all application tasks are of the same hardness H, then H is called the application hardness. Sometimes the reverse value $H–1$ turned out to be more convenient for consideration.

In [7] the notion of application density as the maximal value of the overall utility of an application was introduced: $Dens = max_P(U)$ for all values P of processor performance which the application is feasible with. Obviously, any correct application is feasible with $P = \infty$ and is not feasible with $P = 0$. It is also obvious, that if an application is feasible with the performance values P_1 and P_2, where $P_1 < P_2$, then it is feasible for any $P \in [P_1, P_2]$. Therefore, the value *Dens* exists which corresponds to the minimal processor performance P_0 with which the application is still feasible: for all $P < P_0$ the application is not feasible and for any $P \geq P_0$ it is feasible.

The above consideration prompts an efficient algorithm of "catching the lion in desert" for calculating the value *Dens*. Starting with the interval $[a,b]$ of performance values, where $a = 0$ and $b = P_{max}$, P_{max} being large enough for the application to be feasible, application behavior is simulated with the processor performance $P = (b–a)/2$. The next interval will be either $[a,P]$ if this simulation run confirms application feasibility, or $[P,b]$ otherwise. The loop terminates at the interval $[P–\varepsilon, P+\varepsilon]$, P being the resulting performance value with an accuracy of ε. Application density is determined by external factors and structural features of the application, as well as by selection of the scheduling modes and protocols of access to shared informational resources and may be used as a criterion of efficiency of the selected combination.

Current studies were focused on finding dependencies between application hardness and density for various combinations of scheduling modes and access protocols. For that purpose two dissimilar prototypes of the feasibility checker were developed: one in C++/C#, the other in Forth [8, 9], along with a number of application benchmarks. Simulation results obtained with these dissimilar tools differ for less than 0.1 per cent, which is a strong evidence in their trustworthiness.

3. Feasibility Checker

The overall workflow of the simulator for checking application feasibility is presented in Fig. 1. Simulator initialization consists in selecting the desired combination of the scheduling mode and inheritance mode of the access protocol, setting the respective simulator constraints, reading the task description file, and forming the respective resource and task objects. Then the initial list of system events EventList is formed which consists in activation of the all tasks at the moments of system time defined by their phase shifts. Counts for their maximal response times are set to zero and all resources are set to be unlocked.

In the major simulator loop the first group of time-sake events in the ordered EventList is considered, the simulator system time is set to this time moment and all events from this first group are processed one-by-one according to the event type.

1. In case of activating a task, a new job is created from this task and is added to JobList with its priority and planned starting time equal to the current system time; also a new event is added to EventList – to activate the next instance of this task at the moment of time not less than the current time plus the task period T_i.

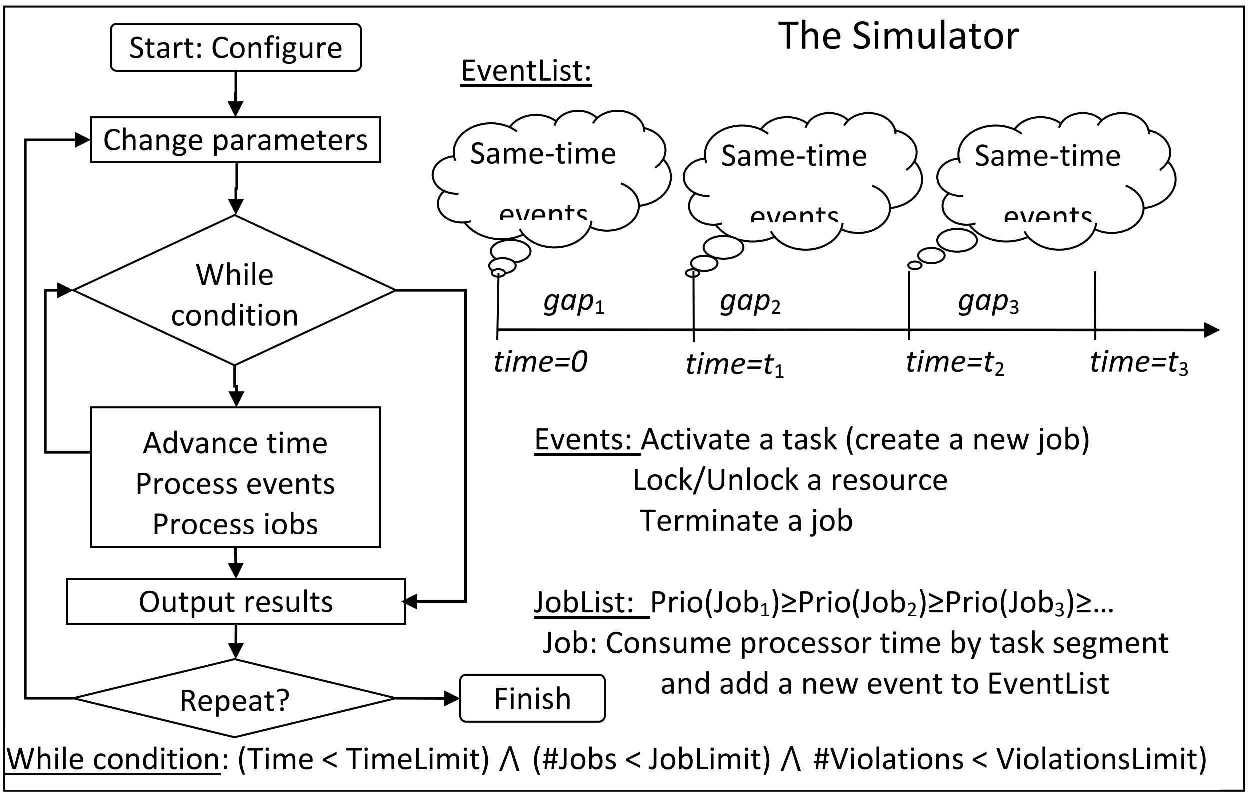


Fig. 1. The overall workflow of the feasibility checker

2. When terminating a job, the response time for the respective task is updated by the maximum of its current value and the existence time of this job. If the job existence time exceeds the task deadline D_i, then a violation of the task feasibility is registered. The considered job is deleted from the JobList.

3. In case of locking/unlocking a resource, if an already locked resource is being locked, the job is moved from JobList to the ordered list of jobs waiting for unlocking of this resource; otherwise the resource becomes locked by this job. When unlocking a resource, if the list of jobs waiting for it is not empty, then the first job form this list is moved back to JobList according to its priority and the resource becomes locked by this job; otherwise, the resource becomes unlocked.

Upon completion of the event processing, the considered event is deleted from EventList. After processing all time-sake events, JobList is considered (it may change as a result of event processing). If it is non-empty, its first element is selected and the time it consumed by this job is updated accordingly, probably generating a new event to terminated this job. If JobList is empty, this means the processor is registered to stay idle for the gap till the next time-sake event group. After that processing the major loop is reiterated. The loop terminates upon exhausting the time limit of the simulation session or when a specified number of created jobs is reached.

The results of simulation – maximum task response time, number of deadline violations, the application density, and other statistics data are displayed. A simulation log may also be displayed.

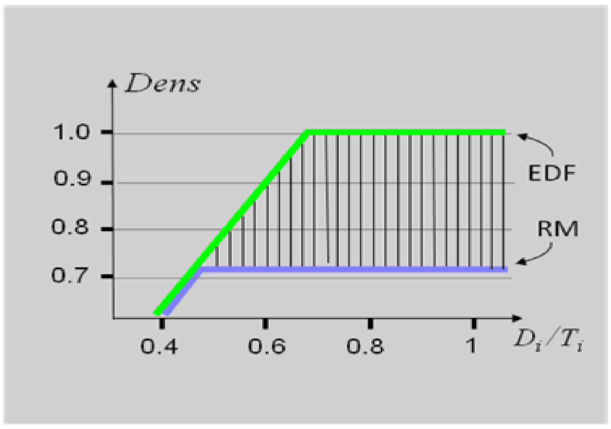
When any system event is processed, the respective time and other accompanying data are printed-out. All these data may be easily copied into MS Excel for a graphical representation of the obtained results and execution log.

4. Running Experiments with the Simulator

To run experiments, particular configurations of software applications to be analyzed are submitted to the simulator. As a rule, valuable configurations of real-time applications actually used in the practice of industrial programming are confidential proprietary; therefore, experiments were run on configurations of mainly methodological interest, the ones with uniform distribution of utility load and with logarithmical distribution of task periods being among them. Let's consider the known Liu-Layland configuration of 10 tasks which is both utility-uniform and with logarithmical periods of its tasks:

Task	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
T_i	100	107	114	123	132	141	151	165	174	187
C_i	7.2	7.7	8.3	8.9	9.5	10.2	10.9	11.6	12.4	13.2

The left diagram in Fig. 2 demonstrates how the application density *Dens* depends on the hardness $H=T_i/D_i$ (actually H^{-1}) of its tasks for two classical scheduling modes: Rate Monotonic (RM) and Earliest Deadline First (EDF) on a single core processor. The task hardness H is supposed to be the same for all 10 tasks, and such configuration is called "balanced".



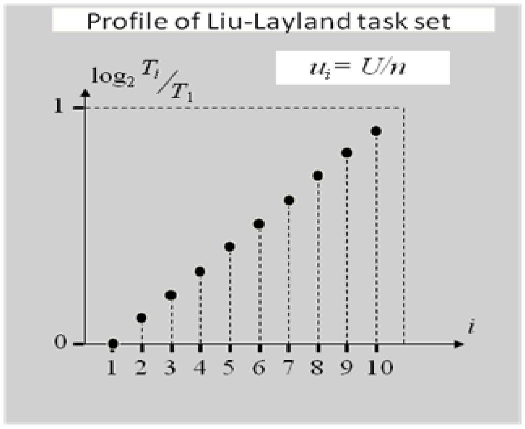


Fig. 2. Liu-Layland balanced configuration of 10 tasks on a single core platform

As one can see, the EDF and RM scheduling modes behave the same when deadline is much less than the period, but then EDF ensures the maximal density of 1 while RM cannot raise over 0.72. The right-side diagram demonstrates the logarithmic dependency of task periods which turns into a pure linear profile.

Results of another experiment with 5 independent tasks with balanced Liu-Layland configuration are presented in Fig. 3.

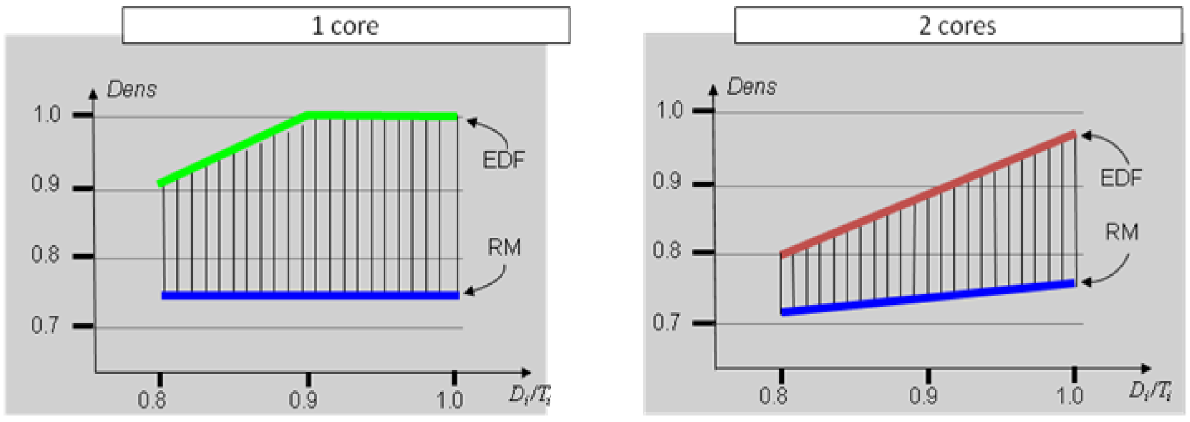


Fig. 3. Liu-Layland balanced configuration of 5 independent tasks

Here two analogous charts for 5 independent tasks with a balanced Liu-Layland configuration are compared when run on a single core and on a two-core platform:

Task	τ_1	τ_2	τ_3	τ_4	τ_5
T_i	1000	1150	1330	1520	1750
C_i	1000	1150	1330	1520	1750

One can notice that the utility load for each task is U_i=1 and the ratio U_i/U=0.2. With the optimal processor performance *P* using the EDF scheduling mode with application hardness approaching 1 ensures 100% processor load (*Dens*=1) on a single core processor, while on a two-core processor not only RM cannot ensure this efficiency, but EDF fails to reach it as well.

In Fig. 4 the same configuration as in Fig. 3 is studied, but this time the tasks share 5 common resources using mutexes to guard the respective critical intervals and the application runs on a single core platform.

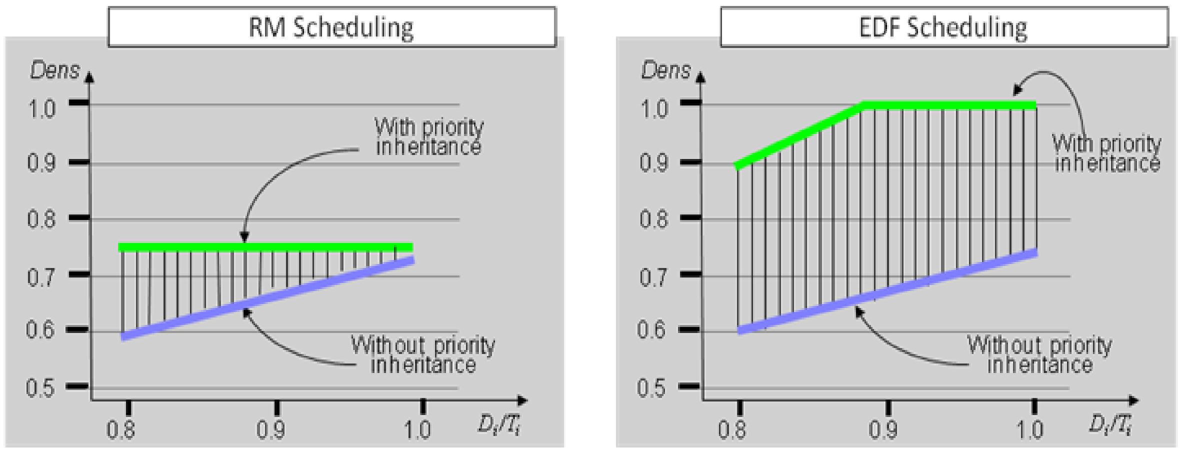


Fig. 4. Liu-Layland balanced configuration of 5 dependent tasks, single core

The charts represent the density/hardness dependency under RM and EDF scheduling modes with and without priority inheritance when tasks perform access to shared resources. Two unexpected facts are worth mentioning in this case:

- when the scheduling mode ignores priority inheritance, the density values for RM and EDF are exactly the same;
- when priority inheritance in any form is added, then density is the same as for independent tasks for both RM and EDF scheduling modes.

When the same application with a priority inheritance mechanism runs on a two-core processor, the advantages of the EDF scheduling mode over the RM one diminish in comparison with a single core platform, especially when the application hardness is close to 1, as one can see in Fig. 5.

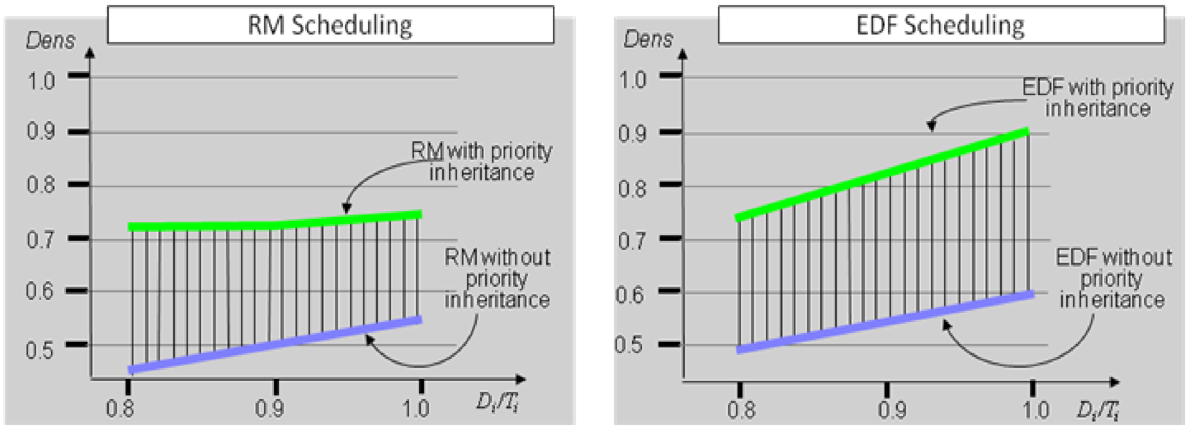


Fig. 5. Liu-Layland balanced configuration of 5 dependent tasks, 2 cores

In the next Fig. 6 one can see how for the same Liu-Layland balanced configuration of 10 independent tasks the difference between RM and EDF diminishes when the number of cores in the processor increases.

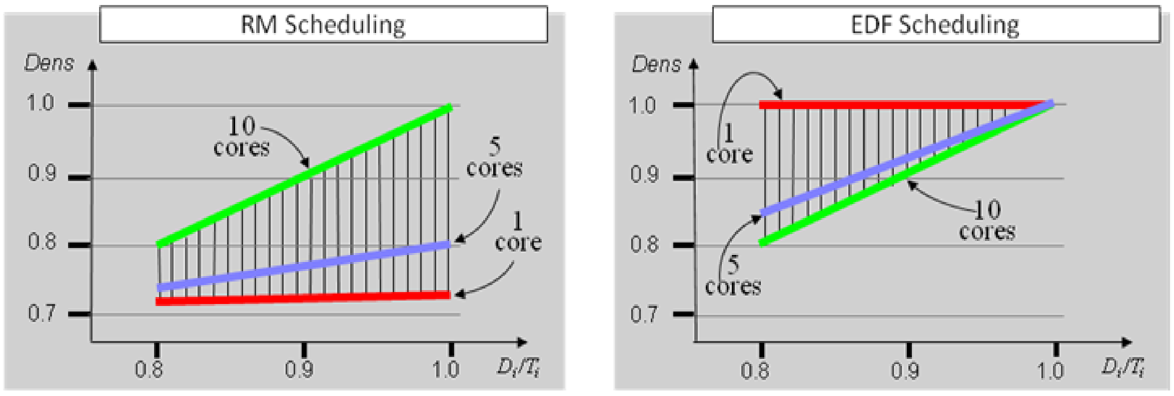


Fig. 6. Liu-Layland balanced configuration of 10 independent tasks, many cores

In Fig. 7 two 4-task configurations with non-uniform utility load are studied. Tasks in the left-hand part are all independent. In the right-hand part tasks 1 and 3 share resource 1, while tasks 3 and 4 share resource 2 and are therefore dependent.

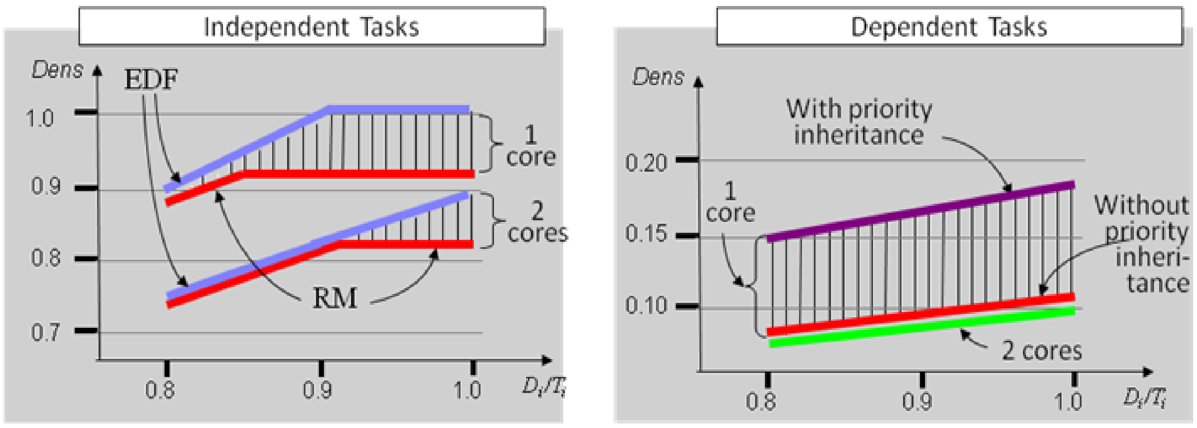


Fig. 7. RM vs. EDF for dependent and independent 4 tasks

For independent task with hardness close to 1, the EDF scheduling mode is much more efficient than RM on a single core platform as well as on a two core one. When deadline diminishes, this difference between EDF and RM disappears. Dependent tasks with shared resources RM scheduling, both with and without priority inheritance, demonstrate the same application density on a two core platform. However, on a single core platform priority inheritance provides substantial gain. The RM and EDF scheduling modes are optimal in their classes of applications when the latter run on a single core platform.

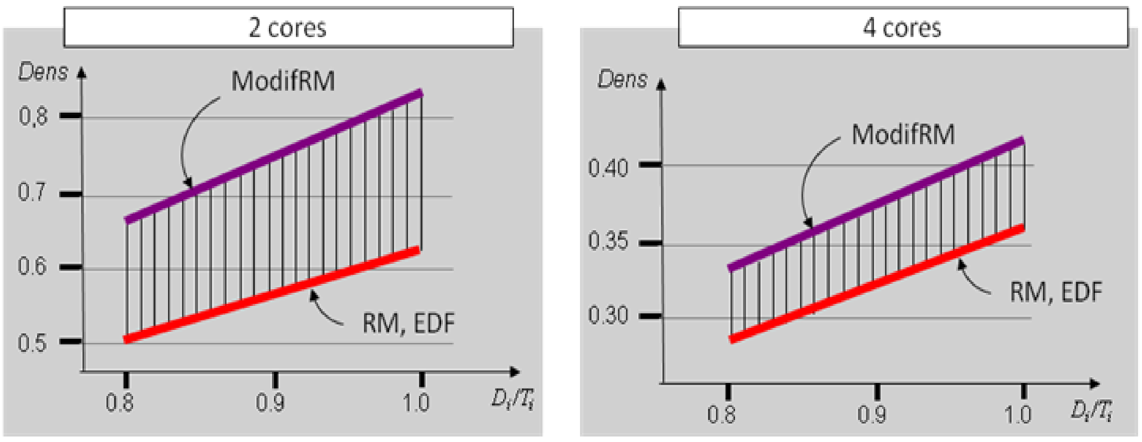


Fig. 8. RM vs. EDF for 5 independent tasks with shifted utility load

In case of multi-core processors, they are not optimal which is pretty well demonstrated by applications with shifted load; i.e., with substantially non-uniform distribution of the total load within the tasks. In the 5 task example in Fig. 8 60% of the total utility load is on task 1: it may be characterized as a "heavy" one, while tasks 2-5 may be called "light". In such cases it's reasonable to use a modified RM scheduling mode: the heavy task is assigned the highest priority, while scheduling among light tasks is governed in the regular RM way. Both charts in Fig. 8 demonstrate the advantages of this ModifRM (Modified Rate Monotonic) scheduling mode on a 2 and 3 core platform, while using RM and EDF scheduling modes is equally inefficient in this case.

5. Conclusions

Using the described methods of software simulation for estimating feasibility of real-time multi-task applications on multicore platforms allows to obtain objective data for selecting an optimal combination of scheduling mode and access protocols. It's noteworthy that analytical methods for such estimates exist for single core platforms only; if used for a multi-core platform they provide too pessimistic results. Therefore, simulation becomes an important tool for searching optimal application structures and platforms for real-time multi-task applications. The developed simulator may be used for feasibility checking of such applications.

Future research will be focused on extending the nomenclature of scheduling modes and access protocols and the profile of applications under study.

This work was partially financially supported by the Government of the Russian Federation, Grant 074-U01.

References

1. Liu, C., Layland, J. Scheduling Algorithms for Multiprocessing in a Hard Real-Time Environment. Journal of the ACM, 20(1), 46-61 (1973)
2. Andersson, B., Baruah, S., Jonsson, J. Static-Priority Scheduling on Multiprocessors. Proc. 22nd IEEE Real-Time Systems Symposium, 193-202 (2001)
3. Laplante, P.A. Real-Time Systems Design and Analysis. John Wiley & Sons, Inc., (2004)
4. Baker, T. Multiprocessors EDF and Deadline Monotonic Schedulability Analysis. Proc. 24th IEEE Real-Time Systems Symposium, 120–129 (2003)
5. Andersson, B. Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%. Proc. 12th International Conference on Principles of Distributed Systems. Luxor, Egypt, 73-88 (2008)
6. Baranov, S.N. Real-Time Multi-Task Simulation in Forth. Proc. 18th Conf. FRUCT, St.Petersburg, Russia, 17-22 (2016)
7. Baranov, S.N., Nikiforov, V.V. Density of Multi-Task Real-Time Applications. Proc. 17th Conf. FRUCT, Yaroslavl, Russia, 9-15 (2015)
8. Baranov, S.N. The Program RTMT for Simulation of Multi-Task Application Run. Certificate of official registration of a computer program No.2016613095, 16 March 2016 (RU), http://www1.fips.ru/wps/portal/Registers/ (in Russian)
9. Forth 200x, http://www.forth200x.org/forth200x.html (2016)

УДК 519.172

Verification of UCM Models with Scenario Control Structures Using Coloured Petri Nets

Vizovitin N.V. (A.P. Ershov Institute of Informatics Systems SB RAS),
Nepomniaschy V.A. (A.P. Ershov Institute of Informatics Systems SB RAS, Novosibirsk State University)
Stenenko A.A. (A.P. Ershov Institute of Informatics Systems SB RAS)

This article presents a method for the analysis and verification of Use Case Maps (UCM) models with scenario control structures – protected components and failure handling constructs. UCM models are analyzed and verified with the help of coloured Petri nets (CPN) and the SPIN model checker. An algorithm for translating UCM scenario control structures into CPN is described. The presented algorithm and the verification process are illustrated by the case study of a network protocol.

Keywords: *verification, translation, Use Case Maps notation, coloured Petri net, SPIN model checker, protected component, failure handling.*

1. Introduction

At early stages of software projects development during requirements capturing and analysis error prevention is of importance due to high cost on this stage. Use Case Maps (UCM) scenario-oriented graphical notation [10] allows users to formalize and analyze functional requirements. At the same time, it allows customers to monitor the system requirements. UCM models are general purpose. They are used for test case generation [3, 4], building test coverage criteria [2], and as a property specification language [8] for use with model checkers.

UCM model of a system depicts a set of scenarios as cause-and-effect relations between responsibilities. Responsibilities may be superimposed on the underlying components structure, reflecting the architecture of the system. UCM describes interaction of architectural entities focusing on causal relations and abstracting from some details of messaging and data processing.

However, tools for analysis and verification of UCM models are insufficiently developed. The UCM standard [10] defines an analysis procedure, which is implemented in the jUCMNav editor [11]. This analysis technique is rather primitive and it is hard to use. Since the standard describes the language semantics informally using traversal requirements for UCM, a number of papers are focused on providing a formal UCM semantics [6]. A few papers present a solution for verification of UCM models [7]. Verification methods for specific subject domains are also being developed. The paper [1] describes testing, analysis, and verification methods for telecommunication applications based on UCM models.

In previous papers [15, 16], we described an approach for general-purpose UCM models analysis and verification using coloured Petri nets (CPN). UCM models are translated into CPN models. The latter are then verified using the well-known SPIN model checker [9]. CPN models may also be analyzed directly using CPN Tools [5].

This article extends the scope of previously supported UCM constructs with protected components and failure handling constructs. It describes a method of analysis and verification for these UCM constructs.

2. Use Case Maps Notation Overview

The Use Case Maps notation is one of the languages defined in the User Requirements Notation standard [10]. The UCM visual notation is a high-level scenario-oriented modeling tool. It focuses on the causal flow of behavior, which is optionally superimposed on a structure of components. UCM models depict the causal interaction of architectural entities in a system while abstracting from message passing and data details. The notation simplifies modeling and analysis of functional requirements for distributed and concurrent systems while also allowing to reason about system architecture.

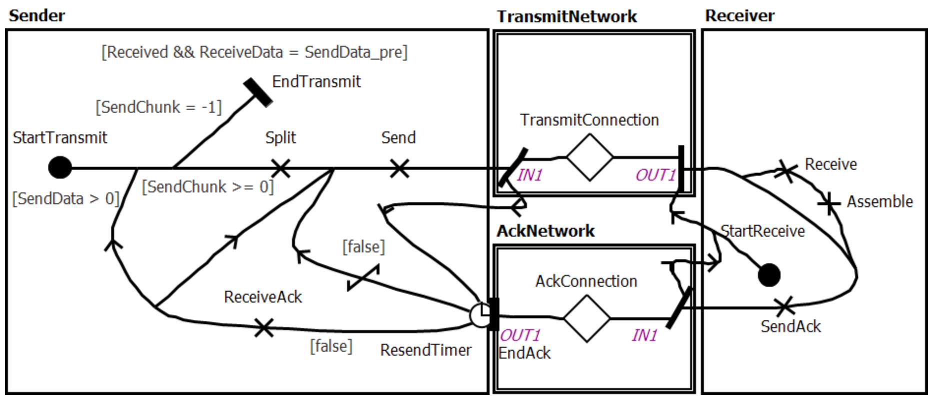


Fig. 1. Top-level map of the network protocol UCM model

Below we provide a short overview of basic elements of the UCM notation. Detailed language description including its graphical syntax is provided in [10] and [15]. A *map* (see Figure 1) contains any number of paths and components. *Paths* (depicted as connecting lines) express causal sequences and causal relationships between path nodes. Paths are directed. They may contain several types of path nodes. Paths start at *Start Points* (for example, `StartTransmit` on Figure

1) and end at *End Points* (`EndTransmit`). These nodes define triggering and resulting conditions respectively or pre-conditions and post-conditions (shown in square brackets). Start Points also may denote the beginning of scenarios for failure and exception handling. Such Start Points are called *Failure Start Points* and *Abort Start Points* respectively. Start Point type is defined by a `failureKind` attribute, while a `failureList` attribute specifies a list of failures it may respond to. Abort Start Point is a Failure Start Point that in addition cancels all scenario behaviors in its *abort scope* – map of the Abort Start Point as well as all lower level maps as defined by Stub hierarchy (see below). *Responsibilities* (`Split`) define steps or actions required to fulfill a scenario. *Or-Forks*, possibly including conditions for outgoing path selection (shown in square brackets), and *Or-Joins* are used to model alternatives and loops. *And-Forks* and *And-Joins* express concurrency. *Waiting Places* and *Timers* (`ResendTimer`) denote points on the path where a scenario stops until a condition is satisfied or a triggering signal arrives. Scenario may also continue past the Timer using the timeout path. *Connect* nodes and *Empty Points* are used to connect two paths synchronously or asynchronously. *Failure Points* represent points on a path where the continuation of a scenario depends on the occurrence of a failure or exception. Each failure point has an associated triggering condition, as well as a failure name, which indicates the failure or exception that happened. Failure name effectively defines Failure or Abort Start Points used to continue scenario execution in case triggering condition is true. UCM models can be hierarchically decomposed using *Stubs* (`TransmitConnection`) that contain reusable units of behavior and structure called *plug-in maps*.

Components (`Sender`) are used to specify structural aspects of a system. Path nodes that reside inside a component are said to be *bound* to it. UCM models without components are said to be *unbounded*. Components may contain sub-components and have various types. However, most of them do not influence model semantics and serve only to convey architectural aspects of a system. Exceptions include components of kind *Object* that force interleaved traversal of path nodes of parallel branches that are bound to the component and *protected* components (`TransmitNetwork`) that restrict the amount of concurrent scenarios inside them. In the URN standard, maximum amount of concurrent scenarios inside a protected component is always 1. Therefore, protected components work as a mutual exclusion mechanism for concurrent scenario execution.

3. UCM Models to Coloured Petri Nets Translation Method

To analyze and verify UCM models we translate them into coloured Petri nets [12]. Input and output models are represented as hierarchical directed graphs with additional information associated with vertices and arcs.

3.1. Input UCM Model Restrictions

The following important restrictions are imposed on the input UCM models. All model elements have unique names and direction of all paths is defined. Special traversal semantics for the components of type Object is not supported. UCM models with Abort Start Points are rejected. These path nodes are rarely used and cause state space explosion upon translation due to scenarios termination semantics. Therefore, we do not translate them to CPN.

For the translation of protected components, we also impose an additional restriction on Or-Fork, Timer, and Or-Join nodes. Each path node of these types should be either bound or not to a given protected component together with all of its adjacent path nodes. This limitation is not fundamental since it could be achieved by simple UCM model modification. In case the system detects that this limitation is not held, the user is offered to modify the UCM model by either introducing additional Empty Point nodes adjacent to the problematic path nodes or in any other way that ensures that the limitation is held.

The listed restrictions do not limit significantly the set of supported UCM models since the most used elements and their use cases are supported.

3.2. UCM to CPN Translation Algorithm Overview

On the top level, UCM to CPN translation algorithm consists of five steps. On the first step pre-processing of an input UCM model is performed. The first step includes simple conversions of an input model as well as checks of input model constraints. The second step creates various CPN ML language definitions common to the entire CPN model. On the third step, additional vertices are added to the UCM model graph to simplify its conversion to a bipartite graph. On the fourth step, path node vertices with their immediate vicinity are translated independently of each other according to their types. The fifth step combines CPN fragments produced on the previous steps into a single CPN model. The translation algorithm is described in detail in [15, 16].

On the first step, UCM model is pre-processed. As part of this process the model is converted to an unbounded one, i.e. all components are removed. Information about protected components is stored in the attributes of each path node bound to the given protected component. All initial values for variables in the UCM model are determined. Algorithm constraints for input models are checked. The system notifies the user about any implicit conversions during this step.

The second step defines colours, constants, and some variables. A number of auxiliary colours are introduced, including `UNIT` – standard "base" colour with only one possible value `()`. Tokens with the colour `UNIT` are normally used to model signal transmission or scenarios execution.

On the third step, graph arcs that are not incident to vertices representing Connect path nodes are partitioned. Each arc is partitioned into two arcs using new helper vertices i.e. vertices of the new type *FakePathNode*. The resulting arcs preserve directions as well as annotations on the arcs outgoing from non-helper vertices.

On the fourth step, each path node vertex with its immediate neighborhood defined by adjacent Connect and helper vertices is handled separately. Each path node and its neighborhood are translated into a CPN model fragment – an annotated graph with additional definitions in CPN ML language. For each failure name, a CPN model fragment is also generated. During translation, helper vertices become places of type UNIT.

The fifth step combines CPN model fragments produced on the fourth step into a single resulting CPN model. Model elements with same names are either merged or represented as fusion places if necessary.

4. Translation of Path Nodes Bound to Protected Components

To verify UCM models efficiently using CPN, number of scenarios being executed at a given point of the model should be limited. Otherwise, translated CPN model will have places with unbounded place capacity since places are used to model signal transport.

The UCM standard provides a method for modeling mutual scenarios exclusion for a subset of the UCM model paths. Protected components depicted with a double outline fulfill this purpose. All UCM model path nodes bound to the protected component are affected by it. Execution of any scenario may continue inside a protected component only if no other scenario is already being executed inside of it.

However, the semantics of the protected components offered by the standard is too restrictive to represent a wide variety of scenarios interactions while keeping the capacity of CPN places in the translated model limited. Thus, we propose to extend the standard by allowing to specify a maximum amount of concurrent scenarios within a protected component. This could be implemented either by adding a new integer attribute `scenarios` into the *Component* class of UCM abstract grammar or by using comment elements attached to a given protected component. The latter approach may be used to avoid modifying existing UCM editors.

Below we will describe translation of UCM components with attribute `protected = true` and positive values of the `scenarios` attribute, as well as other UCM path nodes which are bound to such components. Note that if `scenarios = 1` then the protected component has the same semantics as in the UCM standard.

As any other UCM element, protected components have unique names. We will also impose an additional restriction for Or-Fork and Or-Join nodes. Each path node of these types should be either bound or not to a given protected component together with all of its adjacent path nodes. This limitation is not fundamental since it could be achieved by a simple UCM model modification. However, it avoids semantics ambiguity for such UCM models, as well as significant complication of the translation algorithm.

The example of protected components translation from Figure 1 is provided in Section 7 and a more detailed description of it is given in [14].

On the first step of the translation algorithm, we additionally check that `scenarios > 0` for all protected components. Otherwise, UCM model is deemed incorrect. The additional limitation on Or-Fork and Or-Join nodes is checked as well. If it does not hold user is advised to modify the UCM model by adding new Empty Points on the arcs incident to the problematic path nodes and adjusting protected components. Information about each protected component is stored in the attributes of path nodes bound to it. Each path node may be bound to multiple protected components.

The second and third steps of the translation algorithm have nothing specific for protected components. They are considered in Section 3.2.

Protected components are modeled in CPN using anti-places. Anti-place is a common CPN modeling pattern used to limit the amount of tokens in a given fragment of CPN. Initial marking of an anti-place usually holds the amount of UNIT tokens equal to the limit. When other tokens are created in a given CPN fragment an equal amount of tokens from the anti-place should be consumed. When other tokens are removed from a given CPN fragment, an equal amount of tokens should be put back to the anti-place.

On the fourth step, each path node vertex and its adjacent vertices is translated into a CPN fragment. An anti-place is created for each protected component a vertex has information about in its attributes. The anti-place has a colour UNIT and an initial marking with the same amount of tokens as the value of the `scenarios` attribute was for the protected component. Only the nodes that are capable of starting (forking) or terminating (joining) scenarios that flow through them and a protected component will actually create additional anti-places and arcs. Anti-places are named after the corresponding protected components, so they are uniquely identifiable as well.

The fifth step of the translation algorithm stays the same – all additional anti-places will be joined according to their names in the same way other places of CPN fragments are joined, using fusion places if necessary.

Translation of separate UCM path nodes is described below. For each path node and each protected component, we may define whether this path node and any of the path nodes adjacent to it are bound to the component. For Or-Fork and Or-Join nodes the path node itself as well as other path nodes adjacent to it are either all bound or not to a given protected component. They do not create or terminate scenarios. Therefore, CPN transitions corresponding to Or-Fork and Or-Join nodes never need to be connected to anti-places.

Let us consider the translation of path nodes that may create or terminate scenarios. These include And-Forks, And-Joins, Start Points, and End Points. If a path node is bound to a protected component, a difference between the number of outgoing and incoming arcs is calculated. In case it is positive, an arc is added to the resulting CPN fragment from the anti-place to the transition corresponding to the path node. In case it is negative, an arc is added in the reverse direction. In both cases, arc inscription equals to the () times the absolute difference value. Note that difference value cannot be zero – otherwise, there is no scenario creation or termination.

Let us consider the translation of path nodes bound to a protected component that have adjacent path nodes not bound to the component. We calculate a balance value for a path node. Starting balance value is 0. Each outgoing arc that leads to a path node not bound to the component decreases balance by 1. Each incoming arc from a path node not bound to the component increases balance by 1. After considering all incident arcs for the given path node we have a balance value of this path node in relation to the protected component. In case the balance value is positive, an arc is added to the resulting CPN fragment from the anti-place to the transition corresponding to the path node. In case it is negative, an arc is added in the reverse direction. In both cases, arc inscription equals to the () times the absolute balance value. In case the balance is zero no new arcs are added.

The additional arcs described above may be added independently of one another. In this case, their inscriptions are combined in a natural way. If a stub is bound to a protected component then the described procedure is applied to all path nodes on child diagrams of the stub as well, accounting for Start Points and End Points that have bindings to the stub, which do not create or terminate scenarios.

5. Translation of Failure Handling Path Nodes

Failure handling in UCM is modeled using Failure Point and Failure Start Point path nodes. Failure Point represents a point in scenario behavior where the continuation of the scenario depends on the occurrence of failure or exception. Failure Start Point denotes the beginning of a scenario behavior in response to failure or, in other words, a start of a failure handler.

Failure Start Point nodes have a list of failure names they are supposed to be triggered for, while Failure Point nodes have a `failure` attribute that denotes the name of failure that is triggered. After a failure is triggered, the triggering scenario at the Failure Point terminates, and a number of scenarios start at Failure Start Point nodes with a matching failure name in their failure list.

On the fourth step of the translation algorithm, each failure name is translated into a CPN model fragment – a transition and a place named after the failure name. The place has `UNIT` type with empty initial marking. An arc leading from the place to the transition is added with a `()` inscription. Arcs are also added from the transition to each place matching Failure Start Points with a corresponding failure name in their failure list. Each of these arcs has a `()` inscription as well. This translation procedure closely resembles And-Fork path nodes translation [15].

Translation of Failure Start Points is similar to ordinary Start Points on child maps when they are bound to stub inputs [16]. Transition corresponding to a Failure Start Point is linked with a place of type `UNIT`, with empty initial marking. Arc from the place to the transition has a `()` inscription.

Translation of a Failure Point is similar to an Or-Fork [15], which has two output paths – one that continues the normal execution of the scenario and one that leads to a placed named after the failure name. The conditions on the output paths are based on the failure condition – one of them is the failure condition and the other one is its negation. Therefore, there is no need for additional `*_OrForkWarnings` place which normally tracks that conditions on the output paths of an Or-Fork path node are mutually exclusive.

Note that a Failure Point and Failure Start Points it triggers may be on different maps. This case is automatically resolved on the fifth step of the algorithm by converting some of the places adjacent to the transition that corresponds to the triggered failure name into fusion places when joining CPN model fragments.

6. Verification of CPN Models

A CPN model translated from UCM model may be analyzed using CPN Tools [5, 12] facilities. In fact, it is especially useful for simulation. It also provides some limited state space analysis tools. However, we find that certain model properties may also be formally verified in an automated and more efficient way. We use our own verification system for CPN that uses well-known SPIN model checker [9]. In order to employ SPIN, CPN models are translated into its input language Promela [13].

Properties for verification are expressed either as simple predicates that are expected to be true at the end state (any state without enabled transitions) or as linear temporal logic formulas. In the former case, the property check is represented as an assertion at the end states of the Promela model. In a number of cases, properties for verification may be derived from the UCM model itself. A common choice is to verify that End Points post-conditions hold and the UCM model is correct with respect to branching conditions. Since UCM models are translated into CPN in a way that provides auxiliary warning places to track the branching errors, it is possible to define such kind of property for verification. On the CPN model level, the property holds if all warning places are empty and all post-condition places contain only `true` tokens in the end state. This is translated to Promela model level as a conjunction of several simple state conditions, which is asserted for the end states.

Several restrictions on the input CPN models are imposed to translate them into Promela language. CPN models produced from UCM models by our translation algorithm conform to all of these restrictions but the finiteness restriction required to verify a model efficiently. CPN models are expected to be finite – places and data types' capacities should be limited. The finiteness restriction may be viewed as a reflection of real computer memory finiteness. It is possible to set all finiteness limits manually before the verification.

The finiteness restriction may be conformed to in various ways – by either constructing a UCM model in a certain way or applying additional restrictions on the Promela model level for state space exploration. In case a given finiteness limit is reached during a verification the system advises the user to either increase the limit value or modify the UCM model by adding protected components to it. Protected components are used as a means to limit places capacity. At the same time, protected components usually identify an important limitation on the UCM model level, such as a limited network bandwidth or a limited amount of memory available to the system.

Verification may be successful or not. If the given property does not hold, a counterexample is generated. A counterexample is a sequence of states (places with their markings) and binding elements (transitions and their variable bindings) that lead to the found invalid state or does not satisfy linear temporal logic formula if the property was specified as one. For user convenience, counterexamples may then be mapped back to the UCM model or analyzed with CPN Tools. After correcting issues in either the UCM model or the property to verify, the verification process is repeated.

7. Case Study

We demonstrate algorithms and tools presented in this paper in a case study. A UCM model describes a simple communication protocol designed to transfer reliably a decimal number over a network capable of transmitting only one digit per packet. Data to transfer is integer due to URN Data Language limitations. The UCM model is translated to CPN and then to Promela model, which is then executed to verify a post-condition from the UCM model. The case study demonstrates usage of protected components to ensure the translated model is finite.

Figure 1 shows a top-level map of the UCM model of the protocol. All UCM elements except fork and join elements are labeled. URN Data Language expressions (branch conditions and post-conditions) are depicted as labels with code in square brackets. URN Data Language actions (associated with Responsibilities depicted as crosses) and some expressions are not shown. The model includes four components: Sender, Receiver, TransmitNetwork, and AckNetwork. The latter two components are protected and were added by the user after the initial verification attempt failed. These protected components limit the amount of concurrent scenarios to 2 and reflect a limited network bandwidth. Sender splits `SendData` value into digits and sends them over the network, retransmitting as necessary. Each of the two network components contains a Static Stub that represents an unreliable network environment for transmitting packets from Sender to Receiver and vice versa. Both stubs contain the same Connection plug-in map. Receiver processes packets as soon as they arrive and assembles transmitted data from them. Receiver acknowledges each arriving packet with a sequence number of the next expected packet. Sender receives acknowledgement packets and updates the sequence number of the next packet to send. Sender assumes that sequence numbers can only increase.

After sending a packet, Sender waits on a Timer element. If the current packet sequence number equals to the sequence number of the next packet to send, then the same packet is resent. Otherwise, Sender fetches the next digit to send and sends a new packet with the next sequence number. A packet with the payload `-1` signals the end of data. If Sender receives an acknowledgement that such packet was received, data is considered transmitted and the `EndTransmit` End Point post-condition `[Received && ReceiveData = SendData_pre]` is checked, where `SendData_pre` is the initial value of the `SendData` variable. The post-condition is satisfied if Receiver considers the data received (an appropriate flag is `true`) and the data received equals to the data sent.

The UCM model post-condition for the `EndTransmit` End Point is verified, together with the absence of warnings during model execution. According to this property, the protocol always finishes in the expected correct state and the source UCM model is consistent with respect to branching conditions. The property is simply asserted in the resulting Promela model at end states. The UCM model, CPN and Promela intermediate models with verification results and a detailed description can be found in [14]. During verification, no additional restrictions were imposed on the Promela model. Verification was successful.

8. Conclusion

The Use Case Maps graphical notation provides an expressive means of describing functional requirements for software systems and protocols. In this article, we have presented a method for translation of UCM models to CPN and its application for verification of UCM models. This method enables users to analyze and verify more expressive UCM models as compared with the method in [15, 16] by supporting failure handling and protected components.

Protected components with the extended semantics are especially useful for verification. Protected components limit the number of concurrent scenarios thus limiting places capacity in the translated CPN model. This ensures that the model is finite and can be efficiently verified using SPIN.

A current version of our tool supports translation of jUCMNav editor [11] files to CPN Tools [5] files. UCM models translated to CPN can be analyzed using either built-in CPN Tools facilities or the CPN models verifier based on SPIN [13]. A verification result shows if a model is correct with respect to a given property. If not, an error must be located. While it is possible to map the counterexample generated by SPIN to the UCM model, we find that it is often more convenient and productive to perform the required analysis using CPN Tools.

The algorithm for UCM models translation into CPN is efficient. The translation method described in [15, 16] has polynomial complexity for the size of the resulting CPN models [16]. This estimate holds for the translation algorithm described in this paper as well.

It is important to justify that UCM to CPN translation is correct. However, this requires a formal semantics for the UCM, which is not provided by the standard [10].

We plan to evaluate our tools using other UCM models of communication protocols as well as other systems. We also plan to explore timing extensions [7] for the UCM notation.

9. References

1. Anureev I., Baranov S., Beloglazov D., Bodin E., Drobintsev P., Kolchin A., Kotlyarov V., Letichevsky A., Letichevsky A. Jr., Nepomniaschy V., Nikiforov I., Potienko S., Pryima L., Tyutin B.

Tools for Supporting Integrated Technology of Analysis and Verification of Specifications for Telecommunication Applications // SPIIRAN №1.- St.Petersburg, 2013 (in Russian).

2. Baranov S., Kotlyarov V., Weigert T. Verifiable Coverage Criteria for Automated Testing. // SDL 2011, LNCS 7083.- 2011.- P. 79-89.
3. Baranov S.N., Drobintsev P.D., Kotlyarov V.P., Letichevsky A.A. The Technology of Automated Verification and Testing in Industrial Projects. // Proc. IEEE Russia Northwest Section, 110 Anniversary of Radio Invention Conference.- IEEE Press, St.Petersburg, 2005.- P. 81-89.
4. Boulet P., Amyot D., Stepien B. Towards the Generation of Tests in the Test Description Language from Use Case Map Models. // SDL 2015, LNCS 9369.- Springer.- 2015.- P. 193-201.
5. CPN Tools Homepage, `http://cpntools.org/`
6. Hassine J., Rilling J., Dssouli R. Abstract Operational Semantics for Use Case Maps. // FORTE 2005, LNCS 3731.- Springer.- 2005.- P. 366-380.
7. Hassine J. Early modeling and validation of timed system requirements using Timed Use Case Maps. // Requirements Engineering v. 20 №2.- 2015.- P. 181-211.
8. Hassine J., Rilling J., Dssouli R. Use Case Maps as a Property Specification Language. // Software and Systems Modeling 8(2).- 2009.- P. 205-220.
9. Holzmann, G.J.: The SPIN model checker. Primer and Reference Manual.- Addison-Wesley, 2004.
10. ITU-T, Recommendation Z.151 (10/12), User Requirements Notation (URN) – Language definition. `http://www.itu.int/rec/T-REC-Z.151/en`
11. jUCMNav – Eclipse plugin for the User Requirements Notation, `http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome`
12. Jensen K., Kristensen L.M. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. // Springer 2009.
13. Stenenko A.A., Nepomniaschy V.A. Model checking approach to verification of coloured Petri nets // Preprint 178.- Institute of Informatics Systems SD RAN.- Novosibirsk.- 2015 (in Russian). `http://www.iis.nsk.su/files/preprints/stenenko_nepomniaschy_178.pdf`
14. Vizovitin N.V. Verification of UCM Models of Distributed Systems with Protected Components Using Coloured Petri Nets. Appendix. `http://bitbucket.org/vizovitin/ucm-verification-examples-2`
15. Vizovitin N.V., Nepomniaschy V.A. UCM-specifications to coloured Petri nets translation algorithms // Preprint 168.- Institute of Informatics Systems SD RAN.- Novosibirsk.- 2012 (in Russian). `http://www.iis.nsk.su/files/preprints/168.pdf`
16. Vizovitin N.V., Nepomniaschy V.A., Stenenko A.A. Verifying UCM Specifications of Distributed Systems Using Colored Petri Nets // Cybernetics and Sys. Anal. 51, 2.- Springer.- 2015.- P. 213-222.

UDC 004.415.53 - 519.7

Increasing the fault coverage of tests derived against Extended Finite State Machines

Anton Ermakov (Faculty of Radiophysics, Tomsk State University)
Nina Yevtushenko(Faculty of Radiophysics , Tomsk State University)

Abstract. Extended Finite State Machines (EFSMs) are widely used when deriving tests for checking whether a software implementation meets functional requirements. These tests usually are derived keeping in mind appropriate test purposes such as covering paths, variables, etc. of the specification EFSM. However, it is well known that such tests do not detect many functional faults in an EFSM implementation. In this paper, we propose an approach for increasing the fault coverage of test suites initially derived against the specification EFSM. For this reason, the behavior of the specification EFSM is implemented in Java using a template that is very close to the EFSM description. At the next step, the fault coverage of an initial test suite derived against the specification EFSM is calculated with respect to faults generated by μJava tool. Since the EFSM software implementation is template based, each undetected fault can be easily mapped into a mutant EFSM of the specification machine. Thus, a distinguishing sequence can be derived not for two programs that is very complex but for two machines and there are efficient methods for deriving such a distinguishing sequence for Finite State Machine (FSM) abstractions of EFSMs. As an FSM abstraction, an *l*-equivalent of an EFSM can be considered that in fact, is a subtree of the successor tree of height *l* that describes the EFSM behavior under input sequences of length up to *l*. Such *l*-equivalents are classical FSMs and if *l* is not large then a distinguishing sequence can be derived simply enough. The initial test suite augmented with such distinguishing sequences detects much more functional faults in software implementations of a system described by the specification EFSM.

Key words: *Extended Finite State Machine (EFSM), test derivation, fault coverage, mutation testing, μJava.*

1. Introduction

Model based test derivation is now widely used for deriving functional (conformance) tests for software implementations [1, 2] which nowadays are used everywhere including various critical systems. When deriving tests with the guaranteed fault coverage finite state models such as Finite State Machines (FSMs) and their extensions are widely used [2], since these models have the natural reactivity and there are no races between inputs and outputs. However, traditional FSMs are too big for real-life software and extracting such a model from informal description of functional software requirements is rather difficult. Despite the big number of publications about automatic derivation of an FSM from informal behavioral restrictions, most authors consider small examples and very often such model is manually derived by a test engineer. The Extended Finite State Machine (EFSM) [3] extends the classical FSM with input and output parameters, context variables, update functions and predicates defined over context variables and input parameters. For test derivation for telecommunication protocol implementations the EFSM model is often extracted from the protocol RFC specification [2]. As a result, it is nearly to impossible to find the correlation between model and software faults with respect which we are going to guarantee the fault coverage. Often enough faults are injected into constructed software and then corresponding mutants are distinguished [4, 5]. In this paper, we propose to derive distinguishing sequences not for code but for two EFSMs injecting faults into the template Java implementation of the specification EFSM. We note that the same approach can be applied when using other programming languages for which an automatic mutation tool exists.

There exist a number of EFSM based test derivation methods. Tests usually are derived keeping in mind appropriate test purposes such as covering paths, variables, etc. of the specification EFSM. The derived tests have a good quality but it is well known [6, 7] that such tests do not detect many functional faults in a software implementation of a system described by the EFSM. Correspondingly we propose to increase the fault coverage of EFSM based test suites constructing a Java template implementation of the specification EFSM. Using the tool μJava [8] a number of mutants is generated for the template EFSM implementation which are tested using the initial test suite derived by covering appropriate paths in the specification EFSM. If a mutant is not detected by the initial test suite then a corresponding fault is easily mapped into an EFSM fault and a distinguishing sequence is derived not for two software programs that is known to be a very complex task [5] but for two finite state models that is known to be much simpler [9, 10]. First results have been published in [11]; in this paper, we extend a proposed approach to arbitrary EFSMs.

The rest of the paper is structured as follows. Section 2 contains preliminaries. A proposed approach is described in Section 3. In Section 4, we apply a proposed approach to a Simple Connection Protocol SCP that being a ‘toy example’ has many features that are presented in real protocol descriptions. Section 5 concludes the paper.

1. Preliminaries

A *Finite State Machine* (*FSM*) [12] is a 5-tuple $S = (S, I, O, h_S, s_0)$, where S is a nonempty finite set of states with the designated initial state s_0, I and O are nonempty finite *input* and *output* alphabets, $h_S \subseteq I \times S \times S \times O$ is a *behavior* or a *transition* relation. Extended FSM (EFSM) extends the classical FSM by *context* (internal) variables, *input* and output parameters and conditions when a transition can be fired. Formally [3], an EFSM M is a 5-tuple $M = (S, s_0, X, Y, T, V)$, where S is a nonempty finite set of states of the EFSM, X and Y are nonempty finite *input* and *output* alphabets, V is a finite possibly empty set of context variables, T is a set of transitions between states of S. Every transition of the EFSM is a 7-tuple $(s, x, P, o_M, y, u_M, s')$, where s and s' are the initial and final states of the transition; $x \in X$ is an input, along with $D_{inp\text{-}x}$ denoting the set of input vectors, i.e., vectors with all possible values of input parameters which correspond to x (*input parameters*); $y \in Y$ is an output, along with $D_{out\text{-}y}$ denoting the set of output vectors, i.e., vectors with all possible values of output parameters which correspond to y (*output parameters*); P, o_M and u_M are functions over input parameters and context variables from V. The predicate P: $D_{inp\text{-}x} \times D_V \rightarrow \{0, 1\}$ where D_V is the set of context vectors, describes the conditions when a corresponding transition can be fired; the function $o_M : D_{inp\text{-}x} \times D_V \rightarrow D_{out\text{-}y}$ updates the values of output parameters after firing the transition, while the function $u_M : D_{inp\text{-}x} \times D_V \rightarrow D_V$ updates the context variables.

The *configuration* is a pair «state, context vector»; a *parameterized input* (*parameterized output*) is a pair «input, vector of input parameter values» («output, vector of output parameter values». The *initial* configuration is usually denoted $(s_0, \mathbf{v_0})$. A transition of an EFSM can be *fired* if the corresponding predicate is 'True' for the current parameterized input and configuration. Thus, differently from classical FSMs not each transition at a current state can be fired and this is the well known problem of transition execution [3]. It is possible that in order to fire a given transition we have to execute a number of other transitions first, for example, in order to reach a predefined value of a counter.

Well known test for an EFSM is a transition tour which is widely used for detecting functional faults in various systems' implementations [7]. A *transition tour* is a parameterized input sequence that traverses each transition of the EFSM. As mentioned above, it is not simple to construct such a sequence for an EFSM; however, there exist methods [13] for the transition tour construction. Other methods construct the set of input sequences that cover critical paths, conditions, variables but as it is shown in PhD thesis of S. Nika [6], the fault coverage of such tests with respect to functional software faults is very low, around 70 %; when considering functional faults such as transition and/or predicate faults, variable and/or parameters updating, etc. On the other hand, in [7], it is shown that the transition tour also is not very efficient with respect to such functional faults, since a transition tour covers only appropriate paths. Accordingly in [7] it is shown that the fault coverage of random tests of appropriate length is almost the same as that of a transition tour.

As an example, we consider an EFSM describing the behavior of Simple Connection Protocol (SCP) [14, 15], that has three states; every state describes an appropriate operating mode. State S_1 describes a mode when a protocol implementation is waiting for connection, S_2 corresponds to establishing the connection while state S_3 is related to data transmission. Inputs correspond to standard protocol commands: *Req* (request), *Conn* (connection), *Data* (data transmission) and *Reset*. We also use an input parameter *Support* that equals 1 when the connection of the predefined quality QoS can be established and 0, otherwise. Input parameter *SysAvail* equals 1 if the system is available and 0, otherwise. Output parameters are also very natural: *No support*, *Error*, *Abort*, *Support*, *Refuse*, *Accept*, *Ack_1*. Context variable *TryCount* corresponds to the number of attempts when the connection has failed. Despite the fact that this protocol is somehow a "toy protocol", it illustrates many aspects of protocol implementations.

As mentioned above, differently from deriving a distinguishing sequence against software mutants the derivation of such sequences for finite state machines is much simpler. Distinguishing sequences for two FSMs are constructed based on the product of these machines [11]; for EFSMs it is a bit more complex, since appropriate FSM abstractions are derived first [16, 17, 18]. Such abstractions can be derived in various ways, for example, we can simply delete all predicates, context variables, input and output parameters and updating functions. As shown in [19], in this case, a distinguishing sequence will be constructed for two nondeterministic FSMs. It is also the case when predicate abstractions are considered when deriving a distinguishing sequence [20]. One of simple ways is to use *l*-equivalents of an EFSM which describe the EFSM behavior under critical (parameterized) input sequences of length up to *l*. In the paper [10], it is experimentally shown that when two EFSMs differ in a small number of transitions (the specification EFSM and a mutant EFSM with one or two mutation transitions) usually it is enough to consider $l = 2, 3$ when deriving a distinguishing sequence.

2. Test derivation when using μJava

An initial test suite is derived against the specification EFSM using one of known approaches. It can be a transition tour or a set of randomly derived test cases of appropriate length. This initial test suite will be then augmented with distinguishing sequences for mutants derived by μJava for a template Java implementation of the specification EFSM. The last version of this tool (μJava, v.4) appeared in June, 2013 [8]. The μJava has good functional abilities and according to the documents can generate 34 types of code mutations. There are traditional faults such as the operator or variable replacement and object-oriented faults for inheritance, polymorphism, etc.; there is the high correlation between these mutants and functional software faults. For this reason, we have selected this approach in order to increase the fault coverage of EFSM based test suites. It is known that single faults are most hard detecting faults [6], while test suites complete with respect to single faults detect a big number of other (multi) faults. Correspondingly, μJava injects exactly various types of single faults into software implementations. For correct use of μJava it is necessary to perform appropriate pre-settings which can be found in the official developer site [6]. For mutant generation, the μJava graphic shell should be installed and the code project and mutation types have to be selected. As a result, in the *Results* folder all the generated mutants will appear; subfolders will have titles corresponded to a mutation type. By the use of the JUnit library for module testing [21] a current test suite can be applied to all mutants simultaneously in order to determine mutants which are not detected by the test suite, i.e., have the behavior that cannot be distinguished with the template Java implementation. For those mutants, corresponding faults will be injected into the specification EFSM and a distinguishing sequence will be derived for two machines, a mutant and the specification. Therefore, EFSM based test derivation strengthened with μJava includes the following steps.

Step 1. An initial EFSM based test suite *TS* is derived using one of well known methods. This test suite can be a transition tour of the specification EFSM *M*, or a test suite can cover some critical transitions, conditions, paths, etc., or a test suite can be a random test of appropriate length.

Step 2. A Java template implementation of the specification EFSM is derived. The template is very close to the EFSM notion and thus, there is the strong correlation between faults in the specification EFSM and template implementation. In particular, EFSM states in the template implementation are the values of a corresponding variable (for describing an appropriate mode, for example). Context variables and input and output variables correspond to those in the template implementation; predicates describe the conditions for instruction execution.

Step 3. The fault coverage of the test suite *TS* is checked with respect to faults injected by μJava generator into the template implementation and the set *Mut* of the specification EFSM mutants corresponded to undetected faults is constructed.

Step 4. For each EFSM *Imp* of the set *Mut*, an appropriate FSM abstraction is derived keeping in mind mutated transitions. At the next step, a distinguishing sequence for the specification and mutant FSM abstractions is constructed if such a sequence exists. In this case, a derived distinguishing sequence is added to *TS*. If such a sequence is not found then the conclusion is drawn that the mutant is *indistinguishable* with the specification EFSM.

Remark. We note that when a distinguishing sequence is not found, we cannot guarantee the equivalence of the mutant and specification but our experiments with protocol implementation show that such situations occur very rare. In other words, according to performed experiments, the specification and indistinguishable mutant EFSMs always were equivalent. Nevertheless, we underline that for EFSMs there are no necessary and sufficient conditions of the two EFSMs' equivalence and in our experiments when checking the equivalence, we used only some sufficient conditions. One easily checked condition is that the fault injection creates instructions which do not influence the specification EFSM behavior.

3. Analyzing performed experiments for simple connection protocol

We performed experiments with EFSMs which are used for describing protocols Simple Connection Protocol, Time, SMTP, POP3, TFTP, Audio CD player [11]. Almost in all cases, save for the simplest protocols, a transition tour needed to be augmented with distinguishing sequences obtained after using the μJava tool. The augmentation process in more details is illustrated for the Simple Connection Protocol. A template Java implementation has been obtained for this protocol and a transition tour was used as an initial test suite. After applying μJava, 245 traditional (arithmetic) mutants have been generated, along with seven object oriented mutants (Table 1).

Table 1. Generated mutants

Name	Mutant description	Number of mutants
AOIS	Variable increment/decrement	96
AOIU	Inserting a unary operator (arithmetic "-") before a variable	5
LOI	Operand bit based inversion	24
ROR	Logic operator replacement >,<,=,<=,>=,==	91
COR	Logic operand replacement ^,||,&&,&,|	4
COI	Injecting logic inversion into conditions	17
ASRS	Arithmetic operator modification: +=, /=, -=, %=	8
JSI	Adding the "*Static*" modifier to instance variables	7
Overall		**252 mutants**

When applying the initial test *TS* to all generated mutants, it occurred that 62 mutants (24,6%) produced expected outputs to all test cases. Using FSM abstractions we determined that 9 (3,6%) are distinguishable with the specification EFSM. Other 53 (21%) mutants were indistinguishable with the EFSM specification. Based on the EFSM specification the paths were determined where nonequivalent mutations occurred and a test suite has been augmented with three additional parameterized distinguishing sequences of the total length 11; correspondingly, the test suite length was increased from 18 up to 29 parameterized inputs. We also checked 53 indistinguishable mutants using simple sufficient conditions and all of them were found to have injected faults which do not influence the specification behavior. Therefore, the fault coverage of the initial test suite has been increased from 75,4% up to 100 % (with respect to mutants generated by μJava tool).

4. Conclusion

In this paper, we proposed how to increase the fault coverage of EFSM based test suites when testing software implementations, since tests based on covering appropriate paths, variables, etc. are known to be incomplete with respect to functional software faults. We develop a template Java implementation of the EFSM specification such that implementation faults can be easily mapped into the EFSM faults. The μJava tool is used to inject faults into the template implementation and the set of corresponded EFSM faults undetected with the initial test suite is constructed. Thus, a distinguishing sequence is derived not for two Java programs that is very complex but for two machines and there are efficient methods for deriving such a distinguishing sequence for FSM abstractions of EFSMs. As the performed experiments show this approach allows to eliminate mutants which are equivalent to the EFSM specification and to augment the initial test suite with appropriate distinguishing sequences for non-equivalent mutants. We plan more experiments with real protocol software implementations in order to reveal which functional faults still are not detected with constructed test suites. Another direction of our future work includes the study how to use the obtained results for fault localization in software implementations.

References

1. Kaner C., Falk J., Nguyen H.Q. Testing Computer Software, 2nd Ed. New York, et al: John Wiley and Sons, Inc, 1999. 480 p.
2. Proceedings of the International Conference on testing software and systems (former workshop on protocol testing). Kluwer, Springer, 1991 – 2015.

3. Petrenko A., Boroday S., Groz R. Confirming Configurations in EFSM Testing. IEEE Trans. Software Eng. 30(1), 2004.
4. Nica M., Nica S., Wotawa F. On the use of mutations and testing for debugging. Software: practice & experience. 43(9), 2013. 1121-1142.
5. Nica S., Wotawa F. Using Constraints for Equivalent Mutant Detection. Proceedings of WS-FMDS, 2012. pp. 1-8.
6. Nica S. On the Use of Constraints in Program Mutations and its Applicability to Testing, PhD thesis, Graz Technical University, 2013.
7. El-Fakih K., Salameh T., Yevtushenko N. On Code Coverage of Extended FSM Based Test Suites: An Initial Assessment. Lecture Notes in Computer Science, LNCS 8763, 2014. pp. 198-204.
8. μJava documentation // μJava Home Page. 2014. [WEB]. URL: http://cs.gmu.edu/~offutt/mujava/ (accessed: 10.04.2016).
9. Gill A. Introduction to automata theory, N.Y., 1964.
10. Kushik N., Forostyanova M., Prokopenko S., Yevtushenko N. Studying the optimal height of the EFSM equivalent for testing telecommunication protocols. Proceedings of CCIT, 2014.
11. Ermakov A. Deriving Conformance Tests for Telecommunication Protocols Using μJava Tool // 15th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices (EDM 2014). Novosibirsk: Nobosibirsk State Technical University, 2014. pp. 150-153.
12. Villa T., Yevtushenko N., Brayton R., Mishchenko A., Petrenko A., Sangiovanni-Vincentelli. The Unknown Component Problem. Springer, 2012.
13. Burdonov I.B., Kossachev A.S. Studying transition graph by interacting automata. Vestnik of Tomsk State University: series on control systems and informatics. 2014, № 3 (28), pp. 67-75 (in Russian).
14. Alcalde B., Cavalli A., Chen D., Khuu D., Lee D. Network Protocol System Passive Testing for Fault Management: A Backward Checking Approach. Proceedings of FORTE, 2004, pp. 150-166.
15. Kushik N., Lopez J., Cavalli A., Yevtushenko N. Optimizing Protocol Passive Testing through 'Gedanken' Experiments with Finite State Machines. Proccedings of QSR, 2016, (accepted for publication).
16. Kolomeez A.V. Algorithms for Extended Finite State Machine based test derivation for control systems. PhD thesis, Tomsk State University, 2010 (in Russian).
17. Kushik N., Kolomeez A., Cavalli A., Yevtushenko N. Extended Finite State Machine based Test Derivation Strategies for Telecommunication Protocols. Proceedings of SYRCOSE, 2014. pp. 108-113.
18. Kushik N., Yenigün H., Heuristics for Deriving Adaptive Homing and Distinguishing Sequences for Nondeterministic Finite State Machines. Lecture Notes in Computer Science, LNCS 9447, 2015. pp. 243-248.
19. Kushik N., Yevtushenko N., Cavalli A. On Testing against Partial Non-observable Specifications. Proceedings of QUATIC, 2014. pp. 230-233.

20. El-Fakih K., Yevtushenko N., Bozga M., Bensalem S. Distinguishing extended finite state machine configurations using predicate abstractions. Journal on Software Research and Development, published online March, 31, 2016.
21. JUnit 4, documentation. [web] // JUnit Home Page. – URL: http://junit.org/junit4/ (accessed: 10.04.2016).

УДК 519.717

On the minimization and equivalence checking of sequential reactive systems

Temerbekova G.G. (Lomonosov Moscow State University)

Zakharov V.A.

(National Research University Higher School of Economics (HSE))

Finite state transducers over semigroups can be regarded as a formal model of sequential reactive programs. In some cases verification of such programs can be reduced to minimization and equivalence checking problems for this model of computation. To solve efficiently these problems certain requirements are imposed on a semigroup these transducers operate on. Minimization of a transducer over a semigroup is performed in three stages: at first the greatest common left-divisors are computed for all states of a transducer, next a transducer is brought to a reduced form by pulling all such divisors "upstream", and finally a minimization algorithm for finite state automata is applied to the reduced transducer. As a byproduct of this minimization technique we obtain an equivalence checking procedure for transducers operating on certain classes of semigroups.

Keywords: reactive system, transducer, semigroup, minimization, equivalence checking

1. Introduction

Finite state transducers extend finite state automata to model functions on strings or lists. That is why they are used in fields as diverse as computational linguistics [9] and model-based testing [1, 18]. In software engineering transducers provide a suitable formal model for various device drivers for manipulating with strings, transforming images, filtering dataflows. Transducers also found a usage in regular model checking of parameterized distributed systems. In some formal models of these systems configurations are modeled as words over finite alphabet and a transition relation is specified by a finite state transducers [21]. The more succinct is the presentation of these transducers, the more efficient are regular model checking algorithms. The authors of [17] proposed models of communication protocols as regular transducers operating on bit strings and set up the verification problem as equivalence checking between the protocol transducer and the specification transducer. These considerations show that algorithms for building compositions of transducers, checking equivalence, reducing their state space considerably enhance the effectiveness of designing, verification and maintenance of software routines.

Transducers can take on the role of simple models of sequential reactive programs. These programs operate in the interaction with the environment permanently receiving data (requests) from it. At receiving a piece of data such program performs a sequence of actions. When certain control points are achieved a program outputs the current results of computation as a response. Since different sequences of actions may yield the same result we need a more sensitive interpretation of the outputs than just words in some alphabet. Basic actions of a program are interpreted as generating elements of an appropriate semigroup, and the result of computation is a composition of actions performed by the program.

Imagine, for example, that a radio-controlled robot moves on the earth surface. It can make one step moves in any of 4 directions N, E, S, W. When such robot receives a control signal syg in a state q it must choose and carry out a sequence of steps (say, N, N, W, S), and enter to the next state q'. At some distinguished state q_{fin} robot reports its current location. The most simple model of computation which is suitable for designing such a robot and analyzing its behaviour is non-deterministic finite state transducer operating on free Abelian group of rank 2. These considerations give rise to the concept of a transducer which has some finitely generated semigroup S for the set of outputs.

In this paper we study minimization and equivalence checking problems for finite state transducers operating on certain semigroups. The study of these problems for classical transducers over words began in the early 60s. First, it was shown that the equivalence checking problem is undecidable for non-deterministic transducers [8]. But the undecidability displays itself only in the case of unbounded transduction when an input word may have arbitrary many images. At the next stage bound-valued transducers were studied. The equivalence checking problem was shown to be decidable for deterministic [3], functional transducers [2, 15], and k-valued transducers [5, 20]. In a series of papers [12, 13, 16] a construction to decompose k-valued transducers into a sum of functional and unambiguous ones was developed and used for checking k-valuedness and equivalence of finite state transducers over words. An alternative approach which is applicable to a more wide class of transducers was introduced in [23]. It was shown that the equivalence checking problem is decidable for k-valued transducers operating on any semigroup S which is embeddable in a decidable group.

The minimization problem for finite state transducers over words was considered in [11], but only in [10] an admissible solution to this problem was obtained. Later a minimization algorithm proposed by Mohri was corrected and improved in [4, 14]. In [7] an attempt was made to adapt this result to weighted transducers. An alternative approach to this problem was suggested in [24]: it was shown that minimization of finite state transducers operating on decidable groups can be achieved through the using of equivalence checking algorithms developed in [23].

In this paper a minimization technique proposed by M. Mohri [10] is extended to cover the case of finite state transducers operating on ordered semigroups. Minimization of a transducer π over a semigroup S is performed in three stages. At the first stage for every state q we compute the greatest common left-divisor $GCD(\pi, q)$ of all those elements of S that are the results of the runs of π beginning in q. At the next stage we pull all $GCD(\pi, q)$ "upstream" to obtain such a transducer π' that $GCD(\pi', q)$ for every state q is the neutral element of S. A transducer enjoying this property can be minimized by considering its underlying finite automaton only and by applying any minimization algorithm for finite automata (see, e.g. [19]). As a byproduct of this minimization technique we obtain also an equivalence checking procedure for transducers operating on S.

This work is supported by the Basic Research Program at the National Research University Higher School of Economics in 2016 and by RFBR grants N 16-01-00546.

2. Transducers as models of sequential reactive systems

Let $\mathcal{C}$ and $\mathcal{A}$ be two finite sets. The elements of $\mathcal{C}$ are called *signals*; they may be viewed as abstractions of messages (control instructions, instrument readings, etc.) received by a reactive system from its environment. Finite sequences of signals (words over alphabet $\mathcal{C}$) are called *signal flows*. As usual, the set of signal flows is denoted by $\mathcal{C}^*$. We write uv for concatenation of signal flows u and v.

The elements of $\mathcal{A}$ are called *basic actions*; they may be viewed as abstractions of operations (data processings, movements, etc.) performed by a reactive system in response to received messages. Finite sequences of basic actions (words over $\mathcal{A}$) are called *compound actions*.

Actions are interpreted over semigroups. Consider a semigroup $(S, e, \circ)$ generated by the set $\mathcal{A}$, where e is the neutral element, and $\circ$ is a composition operation. The elements of S may be regarded as *data states*. Every basic action $a, a \in \mathcal{A}$, always terminates and when been applied to a data state $s, s \in S$, yields the result $s \circ a$. Every compound action $g = a_1 a_2 \dots a_k$ is interpreted as the composition $a_1 \circ a_2 \circ \cdots \circ a_k$. In order to distinguish a compound action g from its interpretation we denote the latter by $[g]_S$ and skip the index S when a semigroup is clearly assumed from the context.

A deterministic *finite state transducer* over a set of signals $\mathcal{C}$ and a set of basic actions $\mathcal{A}$ is a labeled transition system $\pi = (\mathcal{C}, \mathcal{A}, Q, q_0, F, T, g_0)$, where Q is a finite set of *control states*, $q_0, q_0 \in Q$, is an *initial state*, $F, F \subseteq Q$ is a subset of *output states*, $T, T : Q \times \mathcal{C} \to Q \times \mathcal{A}^*$, is a *transition function*, and $g_0, g_0 \in \mathcal{A}^*$, is an *initializing action*. Every quadruple (q, c, q', g) such that $T(q, c) = (q', g)$ is called a *transition* which is depicted as $q \xrightarrow{c,g} q'$. By the size $|\pi|$ of a transducer π we mean the number $|Q|$ of its control states.

A *run* of π on a signal flow $w = c_1c_2 \dots c_n$ is a sequence of transitions

$$q \xrightarrow{c_1,g_1} q_1 \xrightarrow{c_2,g_2} q_2 \xrightarrow{c_3,g_3} \cdots \xrightarrow{c_n,g_n} q' \ . \qquad (1)$$

We denote this run by $q \xrightarrow{w,h}_* q'$, where $h = g_1g_2 \dots g_n$. If q is the initial state then the run is called *initial*, and if $q' \in F$ then the run is called *output*. If a run is both initial and output then it is called *complete*. When $q \xrightarrow{w,h}_* q$ is a complete run of π and the actions are interpreted over a semigroup S the element $[g_0h]$ is called the *result* of the run.

Finite state transducers can be used as formal models of sequential reactive systems. At the beginning of the computation a reactive system executes an initializing action g_0. At each step of its computation it receives a signal c from the environment and performs a transition $q \xrightarrow{c,g} q'$ by passing its control to a state q' and executing an action g. When a system turns out to be in an output state it displays an achieved result of its computation to an outside observer and continues its interaction with the environment. A behaviour of such a reactive system is completely specified by a partial function $\pi : \mathcal{C}^* \to S$ such that

$$\pi(w) \quad = \quad \begin{cases} [g_0h], \text{ if there exists a complete run } q \xrightarrow{w,h}_* q' \text{ of } \pi, \\ \text{undefined, otherwise,} \end{cases}$$

for every signal flow w.

Transducers π_1 and π_2 are *S-equivalent* ($\pi_1 \sim_S \pi_2$ in symbols) iff $\pi_1(w) = \pi_2(w)$ holds for every signal flow w. A transducer π' is called *S-minimal* if $|\pi'| \le |\pi|$ holds for any S-equivalent transducer π. The minimization problem for transducer over a semigroup S is to build, given an arbitrary transducer π, a S-minimal transducer π' such that $\pi' \sim_S \pi$.

With every transducer $\pi = (\mathcal{C}, \mathcal{A}, Q, q_0, F, T, g_0)$ operating on a semigroup S one can associate a deterministic finite state automaton $A_\pi = (\mathcal{C} \times S, Q, q_0, F, \varphi)$ over a (possibly infinite) alphabet of pairs $\mathcal{C} \times S$; its transition function $\varphi : Q \times (\mathcal{C} \times S) \to Q$ is specified as follows: $\varphi(q, (c, s)) = q' \iff T(q, c) = (q', g) \wedge s = [g]$. Such an automaton takes at its input a finite sequence of pairs $\alpha = (c_1, s_1), (c_2, s_2), \dots, (c_n, s_n)$ and accepts it at reaching an output state q'. Clearly, A_π accepts α iff the transducer π has a complete run (1) such that $[g_i] = c_i$ for every $i, 1 \le i \le n$. Let $L(A_\pi)$ be the set of all sequences $\alpha, \alpha \in (\mathcal{C} \times S)^*$, accepted by A_π. Transducers π' and π'' are *strongly equivalent* on a semigroup S (in symbols $\pi' \approx_S \pi''$) iff $L(A'_\pi) = L(A''_\pi)$.

It is easy to see that if transducers π' and π'' have the same initializing action (i.e. $[g'_0] = [g''_0]$) and $\pi' \approx_S \pi''$ then $\pi_1 \sim_S \pi_2$. In general case the converse is not true. The key idea of our minimization technique is that of finding, given a certain semigroup S, a subclass of reduced transducers such that

1. for every transducer π one can effectively construct a reduced S-equivalent transducer π' such that $|\pi| = |\pi'|$,
2. for any pair of reduced transducers π' and π'' it is true that $\pi' \sim_S \pi''$ iff $\pi' \approx_S \pi''$ and $[g'_0] = [g''_0]$.

Then to minimize a transducer π one needs only to build an equivalent reduced transducer π' and then apply any of the well-known techniques [19] for minimization of a deterministic finite state automaton $A_{\pi'}$. This approach can be used also for equivalence checking of finite state transducers operating on certain semigroups: to check whether $\pi_1 \sim_S \pi_2$ it is sufficient to build S-equivalent reduced transducers π'_1 and π''_2 and then check the equivalence of deterministic finite state automata $A_{\pi'_1}$ and $A_{\pi'_2}$.

3. Ordered semigroups

In this section we will impose certain requirements on a semigroup S to solve efficiently the minimization problem for transducers operating on such a semigroup.

Let a binary relation $\preceq_S$ on S be defined as follows: $s_1 \preceq_S s_2 \iff \exists s : s_1 \circ s = s_2$. A semigroup S is called *ordered* iff $(S, \preceq_S)$ is a partially ordered set. Sometimes we will skip the underscore symbol S if it is clear from the context. Our first requirement is

Req1: $(S, \preceq)$ is a well-founded lattice such that the greatest lower bound is effectively computable for every pair of elements $[h]$ and $[g]$, where $g, h \in \mathcal{A}^*$.

Denote by $s_1 \vee s_2$ and $s_1 \wedge s_2$ the greatest lower bound and the least upper bound of elements s_1 and s_2 respectively. Actually, $s_1 \vee s_2$ is the greatest common left-divisor of s_1 and s_2, and $s_1 \wedge s_2$ is the lowest common multiple of s_1 and s_2. From the definition of $\preceq$ it follows that $s \circ s_1 \vee s \circ s_2 = s \circ (s_1 \vee s_2)$. The neutral element e of S is the least element in $(S, \preceq)$ but this lattice may have no maximal elements. We add to S a new virtual element τ such that

$s \circ \tau = \tau \circ s = \tau$ holds for any element s in S. Clearly, $s \preceq \tau$ holds for every $s, s \in S$. Let $S_\tau = S \cup \{\tau\}$ Thus, if S meets the requirement **Req1** then $(S_\tau, \preceq)$ is a complete lattice. For any subset S' of S_τ we write $\bigvee S'$ for the greatest lower bound of S'.

Req2: There exists an algorithm for solving equations of the form $[g] \circ X = [h]$ for every pair of actions $g, h \in \mathcal{A}^*$.

It is easy to see that if a semigroup S satisfies the requirements **Req1** and **Req2** then the word problem "$[g] \overset{?}{=} [h]$" is decidable in S.

A semigroup S is called *left cancellative* iff $s \circ s' = s \circ s'' \;\Rightarrow\; s' = s''$ holds for every triple of elements s, s', s''. Our final requirement is

Req3: S is a left cancellative semigroup.

Many semigroups widely used in computer science, including free monoids, partially commutative monoids (traces) [6], a semigroup of conservative substitutions [22], etc. meet the requirements **Req1–Req3** listed above.

4. Greatest common divisors

Our minimization algorithm comprises three stages. At the first stage it figures out for every control state q the greatest common divisor of all results of all output runs that start in q.

A control state q of transducer π is *useful* if it is traversed by at least one complete run. It easy to see that useless states do not affect the function $\pi(\cdot)$ and by deleting all useless states with the incoming and outcoming transition we obtain an equivalent transducer π'. We will assume without loss of generality that all control states of transducers are useful.

Let π be a finite state transducer such that $Q = \{q_1, q_2, \dots, q_n\}$. Consider an arbitrary control state q_i of a transducer π and a set

$$S(\pi, q_i) = \{[h] : q_i \xrightarrow{w,h}_* q_j, q_j \in F\} \ .$$

of results computed by the output runs started in the state q_i. We say that the element $gcd(\pi, q_i) = \bigvee S(\pi, q_i)$ is the *greatest common divisor* of the state q_i and use a notation $GCD(\pi)$ for the tuple $\langle gcd(\pi, q_1), gcd(\pi, q_2), \dots, gcd(\pi, q_n)\rangle$.

To compute the greatest common divisors of all control states of π we introduce an operator $\Psi_\pi \;:\; S_\tau^n \;\to\; S_\tau^n$ as follows. For every tuple $\langle s_1, s_2, \dots, s_n\rangle$ in S_τ^n we assume that

$\Psi_\pi(s_1, s_2, \dots, s_n) = \langle s'_1, s'_2, \dots, s'_n \rangle$, where

$$s'_i \;=\; \begin{cases} e, \text{ if } q_i \text{ is an output state,} \\ \bigvee\{[g] \circ s_j : T(q_i, c) = (q_j, g), c \in \mathcal{C}\}, \text{ otherwise,} \end{cases}$$

for every $i, 1 \le i \le n$.

The partial order $\preceq$ can be extended on the set of tuples S^n_τ in the usual way:

$$\langle s_1, s_2, \dots, s_n \rangle \preceq \langle s'_1, s'_2, \dots, s'_n \rangle \iff \forall i : s_i \preceq s'_i \;.$$

Proposition 1. If a semigroup S meets the requirement **Req1** then Ψ_π is monotone operator.

This proposition follows immediately from the definition of Ψ. Since S_τ is a complete lattice, the operator Ψ_π by Knaster-Tarsky theorem has the greatest fixed point $gfp(\Psi_\pi)$. By Kleene theorem the greatest fixed point of Ψ_π is the limit of the descending sequence $\top \succeq_S \Psi_\pi(\top) \succeq_S \Psi_\pi(\Psi_\pi(\top)) \succeq_S \dots$, where $\top = \langle \tau, \tau, \dots, \tau \rangle$. Since by **Req1** the partially ordered set $(S, \preceq)$ is well-founded, $\Psi^k_\pi(\top) = \Psi^{k+1}_\pi(\top) = gfp(\Psi_\pi)$ holds eventually for some k.

Proposition 2. If a semigroup S meets the requirement **Req1** then $gfp(\Psi_\pi) = GCD(\pi)$.

Proof. 1). If $q_i \in F$ then $e \in S(\pi, q_i)$ and, hence, $gcd(\pi, q_i) = e$. If q_i is not an output control state then $S(\pi, q_i) = \bigcup\limits_{c \in \mathcal{C}} \{[g] \circ [h] : T(q_i, c) = (q_j, g), h \in S(\pi, q_j)\}$. Therefore,

$$gcd(\pi, q_i) = \bigvee\{[g] \circ GCD(\pi, q_j) : T(q_i, c) = (q_j, g), c \in \mathcal{C}\}$$

by left-distributivity of $\circ$ over $\vee$. Hence, $GCD(\pi)$ is a fixed point of the operator Ψ_π.

2). Suppose that $gfp(\Psi_\pi) = \langle s'_1, s'_2, \dots, s'_n \rangle$ and $q_i \xrightarrow{w,h}_* q_j$ is an arbitrary output run of π. It could be shown by induction on the length of this run that $s'_i \preceq [h]$. If $s_i = s_j$ then by definition of Ψ_π we have $s'_i = e \preceq [h]$ for any action h. Consider a case of a run $q_i \xrightarrow{w,h}_* q_j = q_i \xrightarrow{c,g} q_k \xrightarrow{w',h'}_* q_j$. Then by induction hypothesis and by definition of Ψ_π we have $s'_i \preceq [g] \circ s'_k \preceq [g] \circ [h'] = [h]$. Therefore, $s'_i \preceq [h]$ holds for every $[h]$ in $S(\pi, q_i)$. This implies that $s'_i \preceq gcd(\pi, q_i)$ for every $i, 1 \le i \le n$. Thus, $gfp(\Psi_\pi) \le GCD(\pi)$. □

5. Reduced transducers

At the next stage our minimization algorithm brings a finite state transducer to a reduced form. We say that a transducer π operating on a semigroup S which satisfies **Req1** is *reduced* iff $GCD(\pi) = \langle e, e, \dots, e \rangle$.

Theorem 1. If a semigroup S meets requirements **Req1–Req2** then every transducer π can be effectively transformed into a reduced transducer π' such that $\pi \sim_S \pi'$ and $|\pi| = |\pi'|$.

Proof. Suppose that q_1 is the initial control state of π. For an arbitrary transition $q_i \xrightarrow{c,g} q_j$ in π consider an equation $gcd(\pi, q_i) \circ X = [g] \circ gcd(q_j)$. Since $S(\pi, q_i) \supseteq \{[g] \circ s : s \in S(\pi, q_j)\}$, by definition of $GCD(\pi)$ we have $gcd(\pi, q_i) \preceq [g] \circ gcd(\pi, q_j)$. Hence, the equation above always has a solution. By **Req2** this solution $X = g'$ can be computed effectively. A transducer π' is obtained from π by the replacement of every transition $q_i \xrightarrow{c,g} q_j$ with a transition $q_i \xrightarrow{c,g'} q_j$ and by the replacement of the initializing action g_0 of π with an initializing action g_0' such that $[g_0'] = [g_0] \circ gcd(\pi, q_1)$.

The relationship $gcd(\pi, q_i) \circ g' = g \circ gcd(\pi, q_j)$ between transitions $q_i \xrightarrow{c,g} q_j$ and $q_i \xrightarrow{c,g'} q_j$ in transducers π and π' can be extended to the runs of these transducers. Consider an arbitrary pair of corresponding runs of π and π' on some signal flow $w = c_1c_2 \dots c_{m-1}c_m$:

$$q_1 \xrightarrow{c_1,g_1} q_2 \xrightarrow{c_2,g_2} \cdots q_{m-1} \xrightarrow{c_{m-1},g_{m-1}} q_m \xrightarrow{c_m,g_m} q_{m+1},$$
$$q_1 \xrightarrow{c_1,g_1'} q_2 \xrightarrow{c_2,g_2'} \cdots q_{m-1} \xrightarrow{c_{m-1},g_{m-1}'} q_m \xrightarrow{c_m,g_m'} q_{m+1}.$$

Then by definition of π' we have the following chain of equalities:

$$\begin{aligned}&[g_1 \dots g_{m-1}g_m] \circ gcd(\pi, q_m) = [g_1g_2 \dots g_{m-1}] \circ [g_m] \circ [gcd(\pi, q_{m+1})] = \\ &= [g_1g_2 \dots g_{m-1}] \circ [gcd(\pi, q_m)] \circ [g_m'] = [g_1g_2 \dots g_{m-2}] \circ gcd(\pi, q_{m-1}) \circ [g_{m-1}'g_m'] = \cdots \\ &\cdots = [g_1] \circ [gcd(\pi, q_2)] \circ [g_2' \dots g_{m-1}'g_m] = gcd(\pi, q_1) \circ [g_1' \dots g_{m-1}'g_m'] \ .\end{aligned}$$

To make sure that $\pi \sim_S \pi'$ it should be noticed first that both functions $\pi(\cdot)$ and $\pi'(\cdot)$ have the same domain. Consider then an arbitrary signal flow w such that $\pi(w)$ is defined. Let $q_1 \xrightarrow{w,h}_* q_m$ and $q_1 \xrightarrow{w,h'}_* q_m$ be complete runs of π and π' on w. Since $gcd(\pi, q_m) = e$ due to $q_m \in F$, the following chain of equalities holds:

$$\pi(w) = [g_0] \circ [h] = [g_0] \circ [h] \circ [gcd(\pi, q_m)] = [g_0] \circ [gcd(\pi, q_1)] \circ [h'] = [g_0'] \circ [h'] = \pi'(w) \ .$$

Hence, $\pi(w) = \pi'(w)$ for every signal flow w.

To make certain that π' is a reduced transducer consider an arbitrary control state q_i in π' (which is also a control state in π) and $gcd(\pi', q) = \bigvee\{[h'] : q_i \xrightarrow{w,h'}_* q, q \in F\}$. Relying on the relationship between the corresponding runs of transducers π and π' and on the fact that $gcd(\pi, q) = e$ holds for any final state q it is easy to notice that

$$\begin{aligned}&gcd(\pi, q_i) \circ gcd(\pi', q_i) = \bigvee\{gcd(\pi, q_i) \circ [h'] : q_i \xrightarrow{w,h'}_* q, q \in F\} = \\ &= \bigvee\{[h] \circ gcd(\pi, q) : q_i \xrightarrow{w,h}_* q, q \in F\} = gcd(\pi, q) \ .\end{aligned}$$

Since S is an ordered semigroup, $gcd(\pi, q_i) \circ gcd(\pi', q_i) = gcd(\pi, q)$ implies $gcd(\pi', q_i) = e$. □

6. Minimization of reduced transducers

At the final stage to minimize reduced transducers we apply any of minimization techniques for deterministic finite state automata. This consideration is based on the close relationships between reduced transducers and finite state automata revealed in the propositions below.

Proposition 3. Suppose that a semigroup S meets the requirements **Req1, Req3**. Let π' and π'' be a pair of reduced S-equivalent transducers such that g_0' and g_0'' are their initializing actions. Then $[g_0'] = [g_0'']$.

Proof. Consider an arbitrary pair $q_0' \xrightarrow{w,h'}_* q'$ and $q_0'' \xrightarrow{w,h''}_* q''$ of complete runs of π' and π'' on some signal flow w. Since $\pi' \sim_S \pi''$, it is true that $[g_0'h'] = [g_0''h'']$. The latter means that $[g_0'] \wedge [g_0''] \neq \tau$. Hence, by definition of the least upper bound in the lattice $(S, \preceq)$ there exists a triple of elements s, s', s'' such that $[g_0'] \wedge [g_0''] = [g_0'] \circ s' = [g_0''] \circ s''$ and $[g_0'h'] = [g_0''h''] = ([g_0'] \wedge [g_0'']) \circ s$. Thus, $[g_0'rc[h'] = [g_0'] \circ s' \circ s$ and $[g_0''h''] = [g_0''] \circ s'' \circ s$. Since S satisfies the requirement **Req3**, these equalities imply $[h'] = s' \circ s$ and $[h''] = s'' \circ s$. It should be noticed that the elements s' and s'' depend on g_0' and g_0'' only. Therefore, the conclusion can be made that $s' \preceq gcd(\pi', q_0')$ and $s'' \preceq gcd(\pi'', q_0'')$. But once π' and π'' are reduced transducers, we have $gcd(\pi', q_0') = gcd(\pi'', q_0'') = e$. Hence, $s' = s'' = e$. This means that $g_0' = g_0''$ □

Proposition 4. Suppose that a semigroup S meets the requirements **Req1, Req3** and let π' and π'' be a pair of reduced S-equivalent transducers. Suppose also that $q_0' \xrightarrow{w,h'}_* q_1' \xrightarrow{c,g'} q_2'$ and $q_0'' \xrightarrow{w,h''}_* q_1'' \xrightarrow{c,g''} q_2''$ are initial runs of π' and π'' on some signal flow wc, where $c \in \mathcal{C}$. Then $[g'] = [g'']$.

The proof of Proposition 4 follows the same line of reasoning as that of Proposition 3. These propositions bring us to

Theorem 2. If a semigroup S satisfies the requirements **Req1, Req3** then for any pair of reduced transducers π' and π'' it is true that

$$\pi' \sim_S \pi'' \iff \pi' \approx_S \pi'' \wedge [g_0'] = [g_0''],$$

where g_0' and g_0'' are initializing actions of π' and π''.

Theorems 1 and 2 provide a solution to both minimization problem and equivalence checking problem for deterministic finite state transducers operating on a semigroup S which satisfies the requirements **Req1–Req3**. To verify the S-equivalence of transducers π_1 and π_2 it is sufficient to minimize both transducers and then check that these S-minimal transducers are isomorphic.

7. Conclusions

Complexity issues of the minimization problem for finite state transducers over semigroups that fall into the scope of requirements **Req1–Req3** is a topic for further research since the complexity depends greatly on the individual algebraic properties of a lattice $(S, \preceq)$.

One may also wonder how much important for minimization problem are the requirements **Req1–Req3**. Some ordered semigroups of actions arising in program modeling are not left-cancellative, and their lattices $(S, \preceq)$ are not well-founded. It would be interesting to study to build effectively S-minimal transducers for such semigroups.

References

1. Alur R., Cerny P. Streaming transducers for algorithmic verification of single-pass list-processing programs // Proc. of 38-th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. 2011. p. 599-610.
2. Blattner M, Head T. Single-valued transducers // Journal of Computer and System Sciences. 1977. Vol. 15. p. 310-327.
3. Blattner M, Head T. The decidability of equivalence for deterministic finite transducers // Journal of Computer and System Sciences. 1979. Vol. 19. p. 45-49.
4. Beal M.-P., Carton O. Computing the prefix of an automaton // Theoretical Informatics and Applications. 2000. Vol. 34. p. 503-514.
5. Culik K., Karhumaki J. The equivalence of finite-valued transducers (on HDTOL languages) is decidable // Theoretical Computer Science. 1986. Vol. 47. p. 71-84.
6. Diekert V., Metivier Y. Partial commutation and traces // Handbook of formal languages. 1997. Vol. 3. p. 457-533.
7. Eisner J. Simpler and more general minimization for weighted finite-state automata // Proc. of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology. 2003. Vol. 1. p. 64-71.
8. Griffiths T. The unsolvability of the equivalence problem for ε-free nondeterministic generalized machines // Journal of the ACM. 1968. Vol. 15. p. 409-413.
9. Mohri M. Finite-state transducers in language and speech processing // Computational Linguistics. 1997. Vol. 23. p. 269-311.
10. Mohri M. Minimization algorithms for sequential transducers // Theoretical Computer Science. 2000. Vol. 234. p. 177-201.

11. Reutenauer C., Schuzenberger M.P. Minimization of rational word functions // SIAM Journal of Computing. 1991. Vol. 30. p. 669-685.
12. Sakarovitch J., de Souza R. On the decomposition of k-valued rational relations // Proc. of 25th International Symposium on Theoretical Aspects of Computer Science. 2008. p.621-632.
13. Sakarovitch J., de Souza R. On the decidability of bounded valuedness for transducers // Proc. of the 33rd International Symposium on MFCS. 2008. p. 588-600.
14. Shofrutt C. Minimizing subsequential transducers: a survey // Theoretical Computer Science. 2003. Vol. 292. p. 131-143.
15. Schutzenberger M. P. Sur les relations rationnelles // Proc. of Conference on Automata Theory and Formal Languages. 1975. p. 209-213.
16. de Souza R. On the decidability of the equivalence for k-valued transducers // Proc. of 12th International Conference on Developments in Language Theory.(2008. p. 252-263.
17. Thakkar J., Kanade A., Alur R. A transducer-based algorithmic verification of retransmission protocols over noisy channels // Proc. of IFIP Joint International Conference on Formal Techniques for Distributed Systems. Lecture Notes in Computer Science. 2013. Vol. 7892. p. 209-224
18. Veanes M., Hooimeijer P., Livshits B., et al. Symbolic finite state transducers: algorithms and applications // Proc. of the 39th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. ACM SIGPLAN Notices. 2012. Vol. 147. p. 137-150.
19. Watson B.W. A taxonomy of finite automata minimization algorithm // Computing Science Report. Eindhoven University of Technology. 2005. Vol. 93/44. 32 p.
20. Weber A. Decomposing finite-valued transducers and deciding their equivalence // SIAM Journal on Computing. 1993. Vol. 22. p. 175-202.
21. Wolper P., Boigelot B. Verifying systems with infinite but regular state spaces// Proc. 10th Int. Conf. on Computer Aided Verification (CAV-1998). Lecture Notes in Computer Science. 1998. Vol. 1427. p. 88-97.
22. Zakharov V.A. On the decidability of the equivalence problem for orthogonal sequential programs // Grammars. 1999. Vol. 2. p. 271-281.
23. Zakharov V.A.: Equivalence checking problem for finite state transducers over semigroups // Proc. of the 6-th International Conference on Algebraic Informatics (CAI-2015). Lecture Notes in Computer Science. 2015. Vol. 9270. p. 208-221.

24. Zakharov V.A., Podymov V.V. On the application of equivalence checking algorithms for program minimization // Proc. of the Institute for System Programming. 2015. Vol. 27. p. 145-174.

УДК 004.912

Преодоление деградации результатов классификации текстов по тональности в коллекциях, разнесенных во времени

Рубцова Ю.В. (Институт систем информатики СО РАН)

В данной работе представлены подходы для решения задачи улучшения классификации текстов по тональности в динамически обновляемых текстовых коллекциях. Предлагается три метода решении обозначенной задачи, принципиально различающихся между собой. В данном случае для классификации текстов по тональности используются методы машинного обучения с учителем и методы машинного обучения без учителя. Приведены сравнения методов и показано в каких случаях какой метод наиболее применим. Описываются экспериментальные сравнения методов на достаточно представительных текстовых коллекциях.

Ключевые слова: *корпусная лингвистика, классификация текстов, анализ тональности текстов, машинное обучение, анализ данных социальных сетей*

1. Введение

Большая часть информации, содержащейся в сети, представлена в текстовом виде на естественном языке. Это усложняет ее обработку и требует привлечения методов компьютерной лингвистики. Поэтому в настоящее время возрастает актуальность лингвистических исследований, разработок новых эффективных программных систем извлечения фактов из неструктурированных массивов текстовой информации и классификации и кластеризации информации, нацеленных как на анализ самих сообщений в сети, так и на выявление источников распространяемой информации. На протяжении последних десяти лет, задачей автоматического извлечения и анализа отзывов и мнений из социальных медиа занимается много ученых и исследователей по всему миру. При этом в качестве одной из главных задач рассматривается задача классификации текстов по тональности.

Тема автоматической классификации текстов по тональности актуальна в России и за рубежом. Одна из первых задач классификации текстов по тональности, которой занимались исследователи, была задача классификации всего документа целиком [25, 30]. Подобный уровень классификации предполагает, что весь документ выражает всего одну тональную оценку или мнение по поводу некоторого объекта или сущности. Если документ содержит описание нескольких объектов или сравнение нескольких объектов, то классификация текстов на уровне документов не даст корректного представления о тональности документа. Чуть позже, классификацию на уровне коротких фраз и выражений, а не на уровне абзацев или целых документов, проводили Wilson, Wiebe и Hoffmann [31]. В своей работе авторы показали, что важно определить окраску (положительная или отрицательная) отдельно взятого предложения, а не всего текста целиком. В длинном документе мнение автора об объекте может меняться с положительного на отрицательное и наоборот; также автор может отрицательно высказываться о мелких недочетах, но в целом оставаться положительно настроенным по отношению к описываемому в тексте объекту. Другими словами, не всегда длинный документ или отзыв однозначно можно классифицировать как положительно или отрицательно окрашенный.

Сообщения микроблогов не превосходят 140 символов, что дает нам возможность отнести классификацию постов микроблогов к классификации на уровне фраз или предложений. Несмотря на то, что микроблоги достаточно молодое явление, исследователи активно занимаются анализом тональности сообщений блогов в целом и Твиттера в частности [7, 10, 16, 24].

Сообщения микроблогов достаточно короткие, чтобы описывать все различные аспекты продукта или услуги и в то же время насыщены мнениями и эмоциональными оценками, поэтому задачу тоновой классификации коротких сообщений решают не только на уровне фраз и предложений, но в том числе и относительно заданного объекта [17, 19].

Большой научный и практический интерес к задаче автоматического извлечения и анализа текстов связан с тем, что пользователи ежедневно публикуют сотни тысяч мнений в социальных сетях, блогах, форумах, специализированных площадках, которые необходимо обрабатывать в полном объеме. Поэтому системы, автоматически распознающие тональность сообщений и умеющие вычленять мнение в текстах, востребованы специалистами, разрабатывающими рекомендательные системы, экспертные системы; маркетологами и аналитиками, проводящими маркетинговые исследования; политологами, которые оценивают тональность новостей и настроение населения и др.

Одной из сложных проблем в разработке и использовании систем, определяющих тональность сообщений, является то, что с течением времени качество их работы постоянно ухудшается. Это происходит, главным образом, из-за того, что со временем меняется словарный состав сообщений. В статье предлагаются подходы к решению данной проблемы.

Статья организована следующим образом, во второй главе обозначается и обосновывается проблема ухудшения качества классификации текстов по тональности на коллекциях одинаковых по составу и характеристикам, но разнесенных во времени, для этого описываются коллекции на которых проводились эксперименты, описываются метрики оценки качества результатов классификатора и приводятся результаты экспериментов работы классификатора на текстовых коллекциях собранных с разницей в полгода-полтора года. В третьей главе предлагаются подходы к решению этой задачи. Последний раздел состоит из выводов и заключения.

2. Снижение качества классификации текстов по тональности из-за изменения тональной лексики

Пользователи социальных сетей одни из первых начинают использовать новые термины в повседневном общении. Среди 40 новых слов, включенных в словарь Оксфорда в 2013 году были термины, пришедшие из социальных сетей, например: Srsly (сокращение от англ. seriously – серьезно), selfie (фотографирование самого себя, русский аналог – себяшка, селфи). Таким образом, активный лексикон постоянно дополняется, следовательно, автоматические классификаторы должны учитывать это в своих моделях. Если мы говорим о машинном обучении, то обучающие коллекции текстов должны пополняться. Если речь идет об использовании правил и словарей, то для улучшения качества классификаторов необходимо учитывать сленг, которым насыщены социальные сети. Так как активный словарный запас регулярно пополняется новыми терминами, в том числе и терминами, выражающими эмоции, следовательно, и словари тональной лексики также должны регулярно обновляться.

2.1. Коллекции коротких сообщений

Работы и эксперименты по автоматической классификации текстов показывают, что результаты классификации, как правило, зависят от обучающей текстовой выборки и предметной области, к которой относится обучающая коллекция. На сегодняшний день, многие работы сводятся к построению вектора признаков (англ. feature engineering) и подключению дополнительных данных, таких как внешние текстовые коллекции (не пересекающиеся с обучающей коллекцией) или тональные словари. Дополнительные данные позволяют снизить зависимость от обучающей коллекции и улучшить результаты классификации.

Для качественного решения задачи классификации текстов по тональности необходимо иметь размеченные коллекции текстов. Более того, для решения задачи улучшения классификации по тональности в динамически обновляемых коллекциях, необходимо иметь несколько текстовых коллекций, которые были собраны в разные временные промежутки.

Сбор первого корпуса текстов проходил в декабре 2013 года – феврале 2014 года, для краткости будем называть ее коллекцией 2013 года. В соответствии с письменным обозначением эмоций был произведен поиск позитивно и негативно окрашенных сообщений. Таким образом, из коллекции 2013 года сформировано две коллекции: коллекция положительных твитов и коллекция негативных твитов. Нейтральная коллекция была сформирована из сообщений новостных и официальных аккаунтов twitter. С помощью метода [13] и предложенной автором фильтрации [8] из текстов 2013 года была сформирована обучающая коллекция.

Далее, необходимо собрать и подготовить тестовые коллекции текстов. Сбор второго корпуса, который состоит из около 10 миллионов коротких сообщений, проходил в июле-августе 2014 года. Третий корпус, состоящий из около 20 млн. сообщений, был собран в июле и ноябре 2015 года.

Из текстов 2014 и 2015 гг., сформированы две тестовые коллекции. Тексты 2014 и 2015 годов подверглись идентичной фильтрации, что и обучающая коллекция 2013 года. Формирование тестовых коллекций по классам тональности происходило аналогично обучающей коллекции, с помощью метода [13]. Распределение количества сообщений по классам тональности в коллекциях представлено в Таблица 1. Все три коллекции являются предметно независимыми, то есть не относятся ни к какой заранее определенной предметной области.

Таблица 1. Распределение сообщений в коллекциях по классам тональности

	Положительные сообщения	Отрицательные сообщения	Нейтральные сообщения
2013 год	114 911	111 922	107 990
2014 год	5 000	5 000	4 293
2015 год	10 000	10 000	9 595

Собранные коллекции текстов послужили основой для создания обучающей и тестовой коллекций твиттер-сообщений для оценки твита по тональности относительно заданного объекта на соревновании классификаторов SentiRuEval [3, 19, 20] в 2015 и 2016 годах. В 2015 году описанные коллекции использовались в бакалаврской работе студента МГУ на тему «Анализ тональности предложений на материале сообщений из Твиттера». В 2016 году, коллекции были использованы в качестве дополнительных коллекций для извлечения лексикона для алгоритма автоматической классификации текстов по тональности [9]. Более того, так как текстовые коллекции выложены в открытый доступ, на основе этих коллекций были разработаны веб приложения «Настроение России online» [6] и «Мониторинг тональности твитов о ВУЗ'ах в режиме реального времени» [5].

Ранее в работе [7] автором было показано, что собранные коллекции являются полными и достаточно представительными.

2.2. Метрики оценки качества классификатора

Оценка качества системы классификации текстов по тональности происходит путем сравнения результатов, полученных от автоматической системы классификации и эталонных размеченных результатов.

Основываясь на разнице значений эталонной коллекции и коллекции, автоматически размеченной оцениваемым алгоритмом, вычисляют следующие общепринятые метрики: accuracy, формула 1; точность (англ. precision), формула 2; полнота (англ. recall), формула 3; и F–мера, формула 4 [21].

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}, \qquad (1)$$

где,

- TP – истинно положительное решение, количество текстов, правильно отнесенных к классу P;
- FP – ложно положительное решение, количество текстов, не правильно отнесенных к классу P;
- FN – ложно отрицательное решение, количество текстов, не правильно отнесенных к классу N;
- TN – истинно отрицательное решение, количество текстов, правильно отнесенных к классу N.

для получения стабильных результатов при оценке качества систем классификации используют более устойчивые метрики, такие как полноту, точность и гармоническое среднее между полнотой и точностью – F-меру.

Precision (точность) – это доля объектов классифицированных как X, которые действительно принадлежат классу X или вероятность того, что случайно выбранный твит попал в тот класс, которому он на самом деле принадлежит Формула 2.

Recall (полнота) – это доля всех объектов класса X, которые согласно алгоритму классифицируются как принадлежащие классу X или вероятность того, что случайно выбранный твит из класса при классификации в него и попадёт, вычисляется по формуле 3.

$$\text{Precision} = \frac{TP}{TP+FP}, \tag{2}$$

$$\text{Recall} = \frac{TP}{TP+FN}, \tag{3}$$

F-мера это гармоническое среднее между точностью и полнотой:

$$\text{F-мера} = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \tag{4}$$

В данной работе F-мера считается как среднее значение между F-мерой по каждому из классов тональности. Аналогично, Precision и Recall – среднее значение Precision и Recall по каждому из классов тональности в отдельности:

$$F\text{ - }measure = \frac{F_{positive} + F_{negative} + F_{neutral}}{3},$$

$$F_{positive} = 2\frac{\Pr ecision_{positive} \times \operatorname{Re} call_{positive}}{\Pr ecision_{positive} + \operatorname{Re} call_{positive}},$$

$$\Pr ecision = \frac{P_{positive} + P_{negative} + P_{neutral}}{3},$$

$$\operatorname{Re} call = \frac{R_{positive} + R_{negative} + R_{neutral}}{3}.$$

2.3. Описание проблемы снижения качества классификации текстов по тональности из-за изменения тональной лексики

Для моделирования реальной ситуации, когда со временем может видоизменяться язык или обсуждаемые в социальных сетях темы, подготовлены вторая и третья коллекции коротких сообщений. Разница во времени между сбором первой и второй коллекций около полугода, первой и третьей – полтора года. Несмотря на то, что на первый взгляд лексика не может так быстро измениться, тем не менее темы твитов, влияющие на общее настроение в целом и репутацию в частности, значительно зависят от происходящих позитивных или негативных событий с участием целевого объекта и, как правило, такие события невозможно предсказать заранее. Например, в январе-феврале 2014 года около 12% всех сообщений twitter были про олимпиаду, в августе 2014 года упоминания олимпиады не превосходило 0,5% от числа всех сообщений.

Прежде необходимо показать снижение качества классификации на коллекциях, разнесенных во времени. Для это обучаем модель классификатора на коллекции 2013 года и применяем ее к коллекциям 2014 и 2015 годов. В качестве словарей для построения признакового пространства выбраны словари men_3, men_5 и BOW. Префикс men_N означает, что термин встречается не меньше чем N раз в одной из коллекции, соответствующей одному из классов тональности (положительной, отрицательной или нейтральной). Общее значение количества терминов в обучающей коллекции обозначено как BOW (англ. Bag of words).

Результаты эксперимента, показывающие снижения качества классификации текстов, представлены в таблице 2. Из таблицы 2 видно, что за полтора года качество классификации текстов микроблогов согласно F-меры может упасть до 15-20% в зависимости от выбранного набора признаков.

Таблица 2 Метрики качества классификации текстов микроблогов по тональности на коллекциях, разнесенных во времени

BOW				**Men_3_tfidf**				**Men_5_tfidf**			
Acc	**P**	**R**	**F-мера**	**Acc**	**P**	**R**	**F-мера**	**Acc**	**P**	**R**	**F-мера**
Коллекция 2013 года											
0,7459	0,7595	0,7471	0,7505	0,6457	0,6591	0,6471	0,6506	0,6189	0,6542	0,6184	0,6223
Коллекция 2014 года											
0,6964	0,6984	0,7062	0,6933	0,5086	0,5829	0,5040	0,5026	0,5745	0,5823	0,5795	0,5808
Коллекция 2015 года											
0,6118	0,6317	0,6156	0,5996	0,4651	0,5218	0,4638	0,4549	0,5343	0,5337	0,5360	0,5344

3. Способы уменьшения деградации результатов классификации на текстовых коллекциях, разнесенных во времени

В качестве классификатора был использован метод SVM (support vector machine) и библиотека LIBLINEAR [12]. Библиотека LIBLINEAR – это реализация алгоритма SVM с линейным ядром. Как показывают эксперименты, обучение модели с помощью библиотеки LIBLENIAR существенно превосходит по скорости аналоги, поэтому в данной работе использовалась библиотека LIBLINEAR.

3.1. Использование весовой схемы с линейной вычислительной сложностью

В активный словарный запас постоянно входят новые слова и выражения. Первый вариант уменьшения устаревания словаря – это его постоянное обновление. Таким образом, можно будет следить за появлением новых терминов в языке и своевременно учитывать их при классификации. Постоянно обновлять словарь и пересчитывать веса терминов достаточно затратное действие с точки зрения вычислительной мощности. Следовательно, для постоянного обновления словаря надо подобрать вычислительно не затратную весовую схему. Так, например, для использования метода, основанного на мере TF-IDF:

$$tfidf = tf \times log\frac{T}{T(t_i)} \tag{5}$$

необходимо знать частоту встречаемости термина в коллекциях, следовательно, набор данных не должен меняться во время расчета весов. Это существенно усложняет вычисления при обновлении словаря, если требуется провести обсчет данных в реальном времени. При добавлении нового текста в коллекцию требуется пересчитать веса для всех терминов в коллекции. Вычислительная сложность перерасчета весов всех терминов в коллекции равна $O(N^2)$.

Для того, чтобы решить проблему поиска и расчёта весов терминов в режиме реального времени была использована мера Term Frequency – Inverse Corpus Frequency (TF-ICF) – формула 6. [14]

$$tf.icf = tf \times \log(1 + \frac{|C|}{cf(t_i)}) \tag{6}$$

Где C – это число категорий, cf – число категорий, в которых встречается взвешиваемый термин.

Для расчета TF-ICF не требуется информация о частоте использования термина в других документах коллекции, только принадлежность к категории, таким образом, вычислительная сложность меры TF-ICF равна *O(N)*.

Проверим применимость меры TF-ICF для взвешивания признаков для классификации текстов по тональности. Для этого поставим эксперимент на ранее описанных текстовых коллекциях коротких сообщений. Начнем с того, что получим базовые значения результатов классификатора, которые будем улучшать. Для этого из коллекции 2013 года создается словарь, на основе которого строятся вектора признаков. Для векторной модели признаки взвешиваются схемой TF-ICF, также используем булевскую модель (признак может принимать только два значения 0 – признак отсутствует или 1 – признак присутствует). Используем коллекцию 2013 года в качестве обучающей, на ее основе создается модель классификатора, коллекции 2014 и 2015 годов выступают в качестве тестовых коллекций. С целью выбора признаков, был проведен эксперимент для словарей, в котором термины взвешены мерой TF-ICF. На Рис. 1 представлены результаты работы классификатора согласно F-меры. Видно, что наилучшие результаты показывают словари из которых удалили термины, которые в одной из тональных коллекций встречаются менее трех раз (men_3) и менее пяти раз (men_5). В словарях с названиями 1_0_0, 3_0_0, 5_0_0 удалены термины, которые встречаются во всей обучающей коллекции менее 1, 3 или 5 раз соответственно.

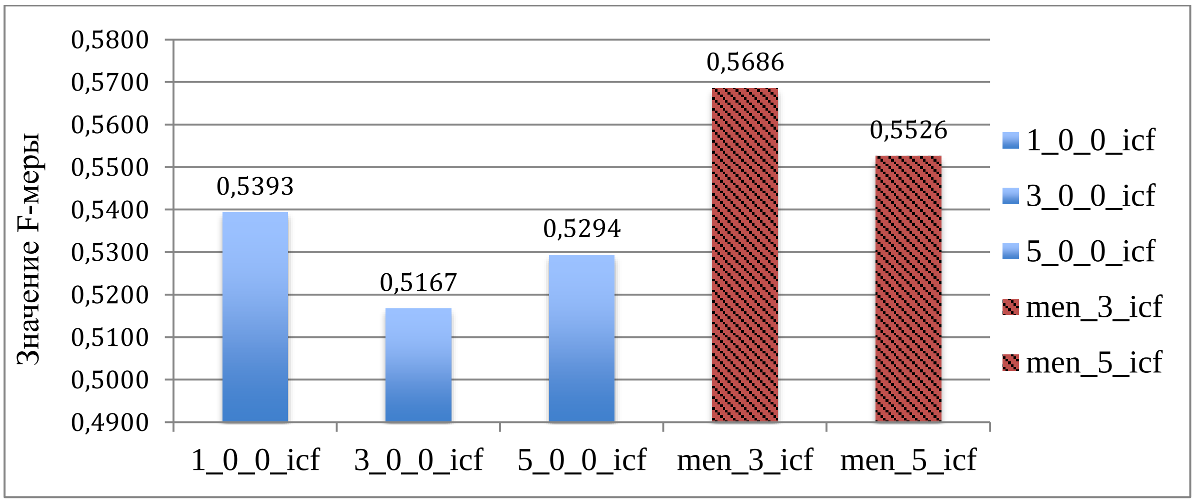


Рис. 1. Усредненные значения F-меры при перекрестной проверке на обучающей коллекции для каждого из словарей признаков взвешенных мерой TF-ICF.

Используя словари men_3_icf и men_5_icf, показавшие согласно F-мере наилучшие результаты классификатора при перекрестной проверке на коллекции 2013 года, протестируем полученную модель классификатора на коллекциях 2014 и 2015 годов. В Таблица 3 приведены результаты F-меры при переносе модели 2013 года на коллекции 2014 и 2015 годов, для наглядности оставлены значения F-меры при перекрестной проверке на коллекции 2013 года.

Таблица 3 Значение F-меры и точности при классификации по тональности с использованием двух лексиконов взвешенных мерой TF-ICF

	Лексикон men_3_TF_ICF		**Лексикон men_5_TF_ICF**	
	F-мера	Accuracy	F-мера	Accuracy
2013	0,5686	0,5648	0,5526	0,5541
2014	0,4645	0,4833	0,4564	0,4971
2015	0,4109	0,4278	0,4143	0,4516

Наблюдается снижение качества классификации при тестировании классификатора на разнесенных во времени коллекциях. Согласно F-мере качество классификации снижается до 15%.

Несмотря на недостатки весовой схемы TF-ICF в виде относительно низкого значения F-меры, у этой схемы есть существенное преимущество в виде линейной вычислительной сложности, что особенно актуально, если речь идет о пространстве, состоящем из нескольких сотен тысяч признаков.

Следующий этап состоит в том, чтобы объединить словарь 2013 года со словарем 2014 года в один, пересчитать веса в полученном словаре и увеличенный словарь использовать для построения классификатора на объединенной коллекции 2013+2014 годов и тестировании на коллекции 2015 года. Для объединённых коллекций был применен метод перекрестной проверки с шагом 5, далее обученный на коллекции 2013+2014 классификатор тестировался на коллекции 2015 года. Ожидается, что значение F-меры для перекрестной проверки классификатора, построенного с помощью словаря признаков men_3_tficf будет в окрестности значения 0,5686 (см Таблица 3), а значение F-меры для коллекции 2015 года будет превосходить 0,4109. Также были проведены эксперименты на словаре признаков BOW. Результаты работы классификатора, согласно F-мере представлены в Таблица 4.

Таблица 4 Результаты классификатора при добавлении коллекций 2014 года к обучающей коллекции

	BOW				**men_3_TF-ICF**			
	Acc	P	R	F	Acc	P	R	F
2013+2014 (перекрестная проверка)	0,7205	0,7339	0,7215	0,7250	0,5539	0,5806	0,5550	0,5565
2015	0,6848	0,6889	0,6862	0,6872	0,5348	0,5571	0,5361	0,5334

Действительно, классификатор показал улучшение значений F-меры для коллекции 2015 года как с помощью использования словаря men_3_tf_icf, так и с помощью метода мешка слов, сохранив порядок результирующих значений при перекрёстной проверке классификатора на обучающей коллекции на уровне 0,55-0,57 для словаря men_3_tf_icf (Таблица 3) и на уровне 0,72-0,75 для мешка слов (Таблица 2).

На третьем этапе все три коллекции были объединены в одну. На основе объединенной коллекции был извлечен словарь для формирования вектора признаков. Термины словаря были взвешены мерой TF-ICF. Аналогично предыдущему эксперименту, задача классификатора состоит в том, чтобы сохранить результирующие значения классификатора не ниже 0,55 согласно F-мере при использовании в качестве признаков словаря men_3_tf_icf и не менее 0,72 при использовании мешка слов. Результаты работы классификатора на объединенных коллекциях 2013, 2014 и 2015 годов представлены на рисунке 2.

На Рис. 2 представлены графики, наглядно показывающие, что динамическое обновление лексикона позволяет сократить ухудшение качества классификации по тональности на разнесенных во времени коллекциях. Сплошная линия показывает значения результата F-меры при обновлении словаря и обучающей коллекции, пунктирная линия показывает результаты классификации при использовании коллекции 2013 года в качестве обучающей.

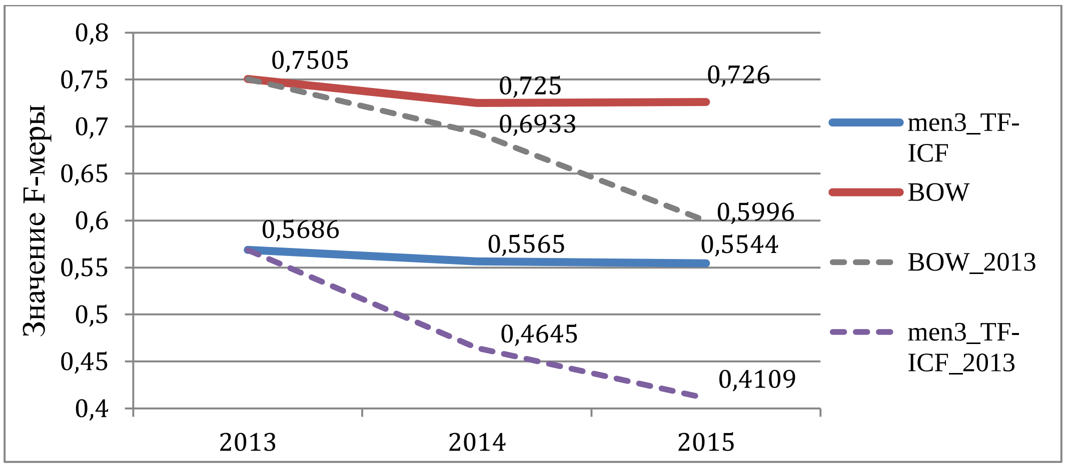


Рис. 2. Результаты F-меры при динамическом обновлении лексикона и тренировочной коллекции (сплошная линия) и без (пунктирная линия)

Классификатор ведет себя единообразно во всех проведенных экспериментах, что позволяет судить о достоверности результатов.

При динамическом обновлении словаря постоянно увеличивается и размерность признакового пространства. Так, при объединении коллекции 2013 года и 2014 года, добавилось более семи с половиной тысяч новых терминов, часть из которых является набором символов, не несущих никакого смысла или встречающихся менее 3-х раз в объединенной коллекции, например, «ш__», «Х_м__с», «х_нд», «болчлоо» или «волчехь». Поэтому, кроме мешка слова, в работе рассматривается отфильтрованный словарь, который показал наилучшие результаты для поставленной задачи – словарь men_3. Это словарь в котором были отфильтрованы все термины, которые встречаются менее трех раз в одной из тональной коллекции. В Таблица 5 представлены значения увеличения размерности исходного словаря при добавлении в него терминов из коллекции 2014 и 2015 годов.

Таблица 5 увеличение размера лексикона, при расширении обучающей коллекции

	BOW	men_3
2013	219 280	41 295
2013+2014	226 964	42 867
2013+2014+2015	245 845	46 312

При накоплении большого количества текстов, становится затруднительно динамически пересчитывать веса терминов в режиме реального времени с использованием меры TF-IDF.

При использовании подхода мешка слов, размерность словаря (а, следовательно, и размерность вектора признаков) будет постоянно расти, потребляя вычислительные ресурсы, но не повышая качество классификации, которое зафиксировалось на уровне 0,72-0,75 согласно F-мере. Использование отфильтрованного словаря и меры TF-ICF сдерживает рост размерности векторов признаков и позволяет пересчитывать веса терминов в режиме реального времени, однако качество классификации согласно F-мере остается на уровне 0,55.

Использование описанного метода оправдано при ограниченных вычислительных ресурсах, а также при отсутствии внешних тональных словарей и дополнительных текстовых коллекций.

3.2. Использование внешних словарей оценочных слов и выражений

Следующая гипотеза состоит в том, что использование внешних словарей эмоционально окрашенной и/или оценочной лексики, повышает качество классификации текстов по тональности, а также сокращается зависимость классификатора от обучающей коллекции. Термины словаря могут быть использованы в качестве признаков в машинном обучении [26] или же использоваться в подходах, основанных на словарях и правилах [27]. Существуют работы, описывающие извлечение или настройку тональных словарей на определенную заранее заданную предметную область [1, 2]. Приводятся примеры терминов, которые могут описывать позитивные характеристики в одной предметной области и нейтральные или даже негативные – в другой. Однако, как показывают [26, 22], объединение обучающих данных из разных предметных областей улучшает качество классификации по тональности в каждой из выбранных областей. Следовательно, существует множество оценочных слов с ярко-выраженной тональной ориентацией, подходящих для разных предметных областей.

В качестве внешних подключаемых словарей в данной работе были использованы два обще тематических словаря тональной лексики, размеченные экспертами: РуСентиЛекс и Linis-crowd.

РуСентиЛекс [4] – лексикон был собран из нескольких источников: оценочные слова из тезауруса русского языка РуТез, сленговые слова из Твиттера и слова с позитивными или негативными ассоциациями (коннотациями) из корпуса новостей. Словарь содержит более десяти тысяч слов и словосочетаний русского языка. Лексикон включает в себя оценочные слова, автоматически извлеченные из текста и проверенные экспертами.

Другой словарь, который использовался в этой работе это **Linis-crowd** [11]. Несмотря на то, что авторы лексикона для формирования словаря использовали тексты социально политической тематики, отмечается, что в словаре присутствует лексика не специфичная для социально-политической тематики, но передающая эмоциональную оценку, поэтому авторами словаря было решено включить ее в прототип Linis-crowd. Словарь содержит 9539 терминов. Каждый термин имеет вес от -2 (сильно-негативный) до +2 (сильно позитивный).

Подключение тональных словарей. Для тонального классификатора, основанного на методах машинного обучения, помимо признаков, порождаемых на основе обучающих данных, были добавлены словарные признаки. Для каждого термина *w* из словаря, обладающего полярностью *p* определено значение *(w, p)*:

$$(w,p)=\begin{cases}>0, w-positive\\<0, w-negative\\=0, w-neutral\end{cases}. \qquad (7)$$

В качестве признаков добавляются:

- общее количество терминов *(w, p)* в тексте твита;
- сумма всех значений полярностей слов лексикона $\sum_{w\in tweet}(w,p)$;
- максимальное значение полярности: $\max_{w\in tweet}(w,p)$.

Каждый из словарей подключался отдельно, сравнение результатов работы словарей можно увидеть в Таблица 6. Как видно из таблицы, оба словаря показывают схожие результаты на обучающей и тестовых коллекциях.

Таблица 6 результаты работы классификатора при подключении словарей РуСентиЛекс и Linis-Crowd

	РуСентиЛекс				**Linis-Crowd**			
	Acc	**P**	**R**	**F**	**Acc**	**P**	**R**	**F**
2013	0,7273	0,74	0,7284	0,7318	0,7272	0,7398	0,7283	0,7316
2014	0,7245	0,7387	0,7259	0,7295	0,7244	0,7386	0,7258	0,7294
2015	0,6724	0,6802	0,6733	0,6759	0,6725	0,6803	0,6733	0,6760

Таким образом, с помощью подключения внешних лексиконов удается приостановить снижение качества классификации на коллекциях, разнесенных во времени. Так как основные признаки порождаются обучающей коллекцией, тенденция к деградации классификатора все же сохраняется, однако, она сокращается с 15% при использовании мешка слов (таблица 2) до 5,6% при подключении словарей эмоциональной лексики.

Таким образом, видно, что при наличии внешних подключаемых тональных словарей имеет смысл использовать этот метод, так как он позволяет сдержать падение качества классификации текстов по тональности на коллекциях, разнесенных во времени.

3.3. Использование распределенных представлений слов в качестве признаков

В двух предыдущих методах, пространство признаков для обучения классификатора строится на основе обучающей коллекции, следовательно, сильно зависит от качества и полноты этой коллекции. Несмотря на хорошие результаты описанных выше моделей, между терминами нет семантических связей, а постоянное добавление новых терминов ведет к увеличению размерности вектора признаков. Еще одним способом преодоления устаревания лексикона является использование пространства распределенных представлений слов в качестве признаков для тренировки классификатора.

3.3.1. Пространство распределённых представлений слов

Распределённое представление слова (англ. distributed word representation, word embedding) – это k-мерный вектор признаков $w=(w_{1,}...,w_k)$, где $w_i \in R$ это компоненты вектора [28]. Если сравнивать с бинарной моделью или моделью взвешенного вектора, то количество координат k такого вектора существенно меньше. Обычно это число не превосходит нескольких сотен, в случае бинарной модели оно измеряется десятками тысяч, в зависимости от размера исходного словаря.

Основная идея векторного распределенного представления слов заключается в нахождении связей между контекстами слов согласно предположению, что находящиеся в похожих контекстах слова, скорее всего означают или описывают похожие предметы или явления, т.е. являются семантически схожими. Для этого каждый термин представляется в виде вектора из k координат в которых закодированы полезные признаки, характеризующие этот термин и позволяющие определять сходство этого термина с похожими терминами в коллекции. Формально это представление терминов является задачей максимизации косинусной близости между векторами слов, которые появляются рядом друг с другом в близких контекстах, и минимизация косинусной близости между векторами слов, которые не появляются в близких контекстах. Косинусная мера близости между векторами, $\cos(\theta)$, может быть представлена следующим образом (формула 8):

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}, \tag{8}$$

где A_i и B_i координаты вектора A и B соответственно.

Помимо сокращения размерности вектора признаков, распределенное представление слов учитывает смысл слова в контексте. То есть позволяет обобщить, например, «быстрый автомобиль» на отсутствующее в обучающей выборке «шустрая машина», тем самым снижается зависимость от обучающей выборки.

Для получения распределенного представления слов используют модели машинного обучения без учителя, напр. CBOW, Skip-Gram, AdaGram [29], Glove. Результаты недавних исследований показывают [18], что нейронная языковая модель Skip-gram превосходит другие модели по качеству получаемых векторных представлений. Поэтому в данной работе используется модель Skip-Gram.

3.3.2. Модель Skip-Gram

Модель Skip-Gram была предложена Томасом Миколовым с соавторами в 2013 году [23]. На вход модели подается неразмеченный корпус текстов, для каждого слова рассчитывается количество встречаемости этого слова в корпусе. Массив слов сортируется по частоте, редкие слова удаляются. Как правило, можно устанавливать порог встречаемости слова при котором слово можно считать редким и до которого все редко встречающиеся слова будут удалены. Для того, чтобы снизить вычислительную сложность алгоритма, строится дерево Хаффмана (англ. Huffman Binary Tree). Далее алгоритм проходит заранее заданным размером окна по выбранному отрезку текста. Размер окна задается как параметр алгоритма. Под окном подразумевается максимальная дистанция между текущим и предсказываемым словом в предложении. То есть если окно равно трем, то для предложения «Я смотрел хороший фильм» применение алгоритма Skip-gram будет проходит внутри блока, состоящего из трах слов: «Я смотрел хороший», «смотрел хороший фильм». Далее применяется нейросеть прямого распространения (англ. Feedforward Neural Network) с много переменной логистической функцией.

Схематически модель Skip-gramm представляется в виде нейронной сети (Рис. 3) [23]:

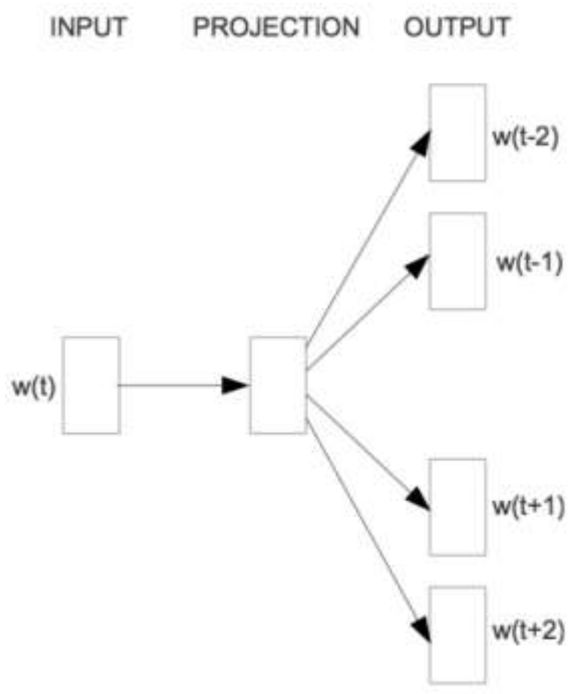


Рис. 3. Архитектура модели Skip-gram

Изображенная на Рис. 3 нейронная сеть состоит из трех слоев: входной (input), выходной (output) и скрытый (projection). Слово, подаваемое на вход, обозначено w(t), в выходном слове w(t-2), w(t-1), w(t+1) и w(t+2) – слова контекста которые пытается предсказать нейронная сеть. То есть модель skip-gram – предсказывает контекст при данном слове.

3.3.3. Использование модели Skip-Gramm для снижения зависимости от обучающей коллекции.

В работе [15] показано, что нейронные сети, с помощью векторных представлений слов, полученных при помощи алгоритма word2vec [29], могут эффективно решать задачи обработки текстов на естественном языке, в общем случае и задачу классификации текстов по тональности в частности. Описанный алгоритм показал лучшие результаты по сравнению с другими алгоритмами на выбранных текстовых коллекциях.

Для обучения модели Skip-Gramm произвольным образом было выбрано 5 миллионов текстов из первоначальной, не разделенной по классам тональности, коллекции 2013 года. Коллекции 2014 и 2015 годов в обучении не участвовали, так как делается предположение, что обученная модель должна быть переносима на более поздние коллекции.

В качестве программной реализации модели Skip-gram был использован Word2Vec [29] со следующими параметрами:

- size 300 – каждое слово представляется в виде вектора заданной размерности;
- window 5 – как много слов из контекста обучающий алгоритм должен принимать во внимание;
- negative 10 – число негативных примеров для негативного сэмплирования;
- sample 1e-4 – суб-сэмплирование, применение суб-сэмплирования улучшает производительность. Рекомендуемый параметр суб-сэмплирования от 1e-3 до 1e-5;
- threads 10 – количество используемых потоков;
- min-count 3 – ограничивает размер словаря для значимых слов. Слова, которые встречаются в тексте менее указанного количества раз, игнорируются. Стандартное значение – 5);
- iter 15 – количество обучающих итераций.

Одной из особенностей Word2Vec является то, что алгоритм не разделяет слово и следующий за ним знак препинания, поэтому, чтобы в файле-модели не было терминов со знаками препинания таких как: «например,», перед началом тренировки знаки препинания были отделены пробелом от идущего перед ними слова. Аналогично, чтобы «не + слово» не было разделено на два различных термина, пробел между частицами не и ни был заменен нижним подчеркиванием (напр. «ни_разу», «не_хотел»).

Из текстов, как и ранее, были отфильтрованы эмотиконы, так как они являются метками принадлежности текста к определенному классу тональности.

Каждый текст был представлен в виде усредненного вектора входящих в него слов (формула 9):

$$d = \frac{\sum w_i}{n}, \qquad (9)$$

где w_i – векторное представление i-го слова, входящего в исследуемый текст, i=(1,..,n), n – число слов из лексикона, входящих в исследуемый текст.

Классификатор был обучен на коллекции 2013 года, далее обученная модель классификатора применялась для тестирования коллекций 2014 и 2015 годов. Результаты классификатора представлены в Таблица 7, результаты метрик качества для коллекции 2013 года оставлены для наглядности.

Таблица 7 Результаты классификации текстов по тональности с использованием векторов слов, полученных при использовании Word2Vec в качестве признаков

	Acc.	**Precision**	**Recall**	**F-мера**
2013	0,7206	0,7250	0,7221	0,7226
2014	0,7756	0,7763	0,7836	0,7787
2015	0,7289	0,7250	0,7317	0,7252

Рис. 4 наглядно показывает, что качество классификации на три класса не только не снижается на коллекциях, собранных с разницей полгода/год, но и держится на уровне лучших значений, полученных при использовании модели мешка слов при перекрестной проверке на коллекции одного года (Таблица 2, Таблица 4). При том, что число координат в векторе слова ровно 300 (задаваемый параметр), а не превосходит 200 тысяч, как в булевской или векторной моделях.

Данный метод хорошо подходит для применения в том случае, если у нас есть внешняя достаточно представительная коллекция текстов, которая схожа по лексике с обучающей и тестовой коллекциями, то есть здесь, как для других нейронных сетей требуется большая обучающая выборка текстов. Метод позволяет получить устойчивые и стабильные результаты.

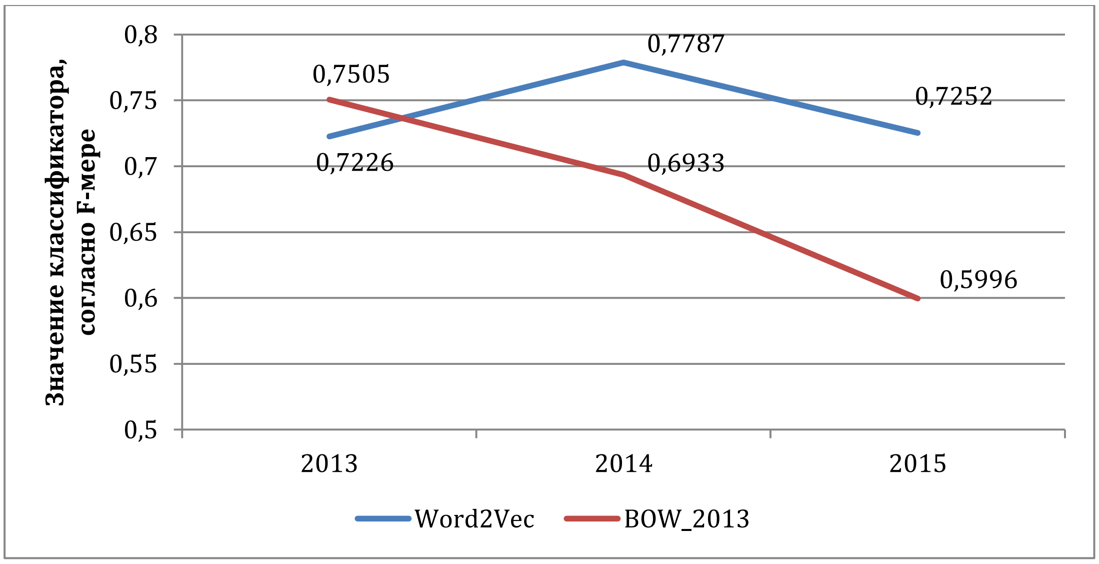


Рис. 4. Сравнение использование векторов слов Word2Vec в качестве признаков и лексикона, основанного на мешке слов коллекции 2013 года.

4. Заключение

В данной статье предложено три принципиально различных модели, позволяющие преодолеть ухудшение результатов классификации по тональности на коллекциях разнесенных во времени. В таблице 2 было показано, что качество классификации текстов по тональности согласно F-мере за полтора года может снизиться до 15%. Таким образом, цель предлагаемых в статье подходов – свести до минимума снижение качества классификации текстов коллекций, разнесенных во времени.

1. В качестве первого подхода, предлагается использовать весовую схему с линейной вычислительной сложностью. Таким образом можно динамически обновлять лексикон и переобучать классификатор. Зависимость от обучающей коллекции снижается потому, что обучающая коллекция постоянно обновляется. В этом случае, разница между работой классификатора на коллекции 2013 года и 2015 года составляет всего 2,4% согласно F-мере при использовании мешка слов и 1,42% при использовании меры TF-ICF. Несмотря на очевидные достоинства этого подхода, у него есть два недостатка:

   1. с обновлением лексикона, увеличивается размерность пространства признаков. Соответственно, с каждым обновлением лексикона, система требует больших ресурсов, вектор текста становится более разреженным.
   2. качество классификации с использованием меры TF-ICF существенно уступает качеству классификации при использовании мешка слов.

2. Второй подход основан на подключении словарей тональной лексики. Лексикон РуСентиЛекс и Linis-Crowd. Использование внешних словарей позволяет сократить разрыв в качестве классификатора между коллекциями 2013 года 2015 до 5,6% согласно F-мере. Разница согласно F-мере между результатами классификации коллекции 2013 и 2014 годов менее 1% и составляет всего 0,2%. При этом качество классификатора держится на уровне 0,68-0,73, что сопоставимо с лучшими результатами. Таким образом, порождение признаков на основе внешних словарей не влечет за собой масштабного увеличения пространства признаков и позволяет показывать хорошие результаты классификации. Несмотря на это, так как пространство признаков по прежнему зависит от обучающей коллекции, наблюдается незначительное снижение качества классификации на более поздних коллекциях.

3. В основе третьего подхода лежит идея пространства распределенных представлений слов и нейронная языковая модель Skip-gram. Как и во втором подходе, здесь были использованы внешние ресурсы. Пространство распределенных векторов слов строилось на

не размеченной коллекции твитов, которая в разы больше автоматически размеченной обучающей коллекции. В качестве признаков были использованы усредненные вектора слов, входящих в один твит. Таким образом, размерность векторного пространства составила всего 300 – это первое преимущество подхода. Вторым преимуществом подхода являются результаты классификации: разница между F-мерой 2013 и 2015 годами составляет 0,26%, при чем результаты классификации на коллекции 2015 года выше. Аналогично с результатами классификации на коллекциях 2013 и 2014 годов, результаты классификации на коллекции 2014 года превосходят на 5,6% результаты классификации на коллекции 2013 года согласно F-мере. Это можно объяснить тем, что для получения результатов на коллекции 2013 года использовался метод перекрестной проверки, то есть коллекция делилась на обучающую и тестовую в отношении 4:5, а при обучении классификатора для тестирования на коллекциях 2014 и 2015 годов использовалась полная коллекция 2013 года.

Таким образом, все три предложенных подхода позволяют снизить ухудшение результатов классификации по тональности на разнесенных во времени коллекциях.

Список литературы

1. Клековкина М. В., Котельников Е. В. Метод автоматической классификации тек-стов по тональности, основанный на словаре эмоциональной лексики //Труды конференции RCDL. – 2012. – С. 118-123.
2. Лукашевич Н. В., Четвёркин И. И. Извлечение и использование оценочных слов в задаче классификации отзывов на три класса //Вычислительные методы и программирование. – 2011. – Т. 12. – №. 4. – С. 73-81.
3. Лукашевич Н., Рубцова Ю. Объектно-ориентированный анализ твитов по тональ-ности: результаты и проблемы // Труды Международной конференции DAMDID/RCDL-2015. — Обнинск, 2015. — С. 499–507.
4. Лукашевич, Н. В., Левчик, А. В., Loukachevitch, N. V., & Levchik, A. V. (2016). Создание лексикона оценочных слов русского языка РуСентиЛекс.
5. Мониторинг тональности твитов о ВУЗ'ах в режиме реального времени [Элек-тронный ресурс]. – Режим доступа: https://tweets-about-universities.herokuapp.com/ (Дата обращения: 10.09.2016).
6. Настроение России online [Электронный ресурс]. – Режим доступа: http://twittermood-ru.appspot.com/ (Дата обращения: 10.09.2016).
7. Рубцова Ю. В. Разработка и исследование предметно независимого классификатора текстов по тональности //Труды СПИИРАН. – 2014. – Т. 5. – №. 36. – С. 59-77.
8. Рубцова Ю.В. Метод построения и анализа корпуса коротких текстов для задачи классификации отзывов // Электронные библиотеки: перспективные методы и технологии,

электронные коллекции: Труды XV Всероссийской научной конфе-ренции RCDL’2013, Ярославль, Россия, 14-17 октября 2013 г. – Ярославль: ЯрГУ, 2013. –С. 269-275.

9. Русначенко Н. Л., NL R. Улучшение качества тональной классификации с ис-пользованием лексиконов.

10. Agarwal A., Xie B., Vovsha I., Rambow O., Passonneau, R. Sentiment analysis of twit-ter data //Proceedings of the Workshop on Languages in Social Media. – Association for Computational Linguistics, 2011. – С. 30-38.

11. Alexeeva, S., Koltsov, S., Koltsova O. (2015), Linis-crowd.org: A lexical resource for Russian sentiment analysis of social media, Computational linguistics and computa-tional ontology, pp 25–34.

12. Fan R.-E. , Chang K.-W., Hsieh C.-J., Wang X.-R., Lin C.-J. LIBLINEAR: a Library for Large Linear Classification. J. of Machine Learning Research. 2008. vol. 9. pp. 1871–1874.

13. J. Read. Using emoticons to reduce dependency in machine learning techniques for sen-timent classification. In Proceedings of ACL-05, 43nd Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2005.

14. Joel W. Reed, Yu Jiao, Thomas E. Potok, Brian A. Klump, Mark T. Elmore, Ali R. Hurson. TF-ICF: A New Term Weighting Scheme for Clustering Dynamic Data Streams. — Proc. Machine Learning and Applications, 2006, ICMLA '06, pp. 258–263.

15. Kim Y. Convolutional neural networks for sentence classification // arXiv preprint arXiv:1408.5882. – 2014.

16. Kouloumpis E., Wilson T., Moore J. Twitter sentiment analysis: The good the bad and the omg! //ICWSM. – 2011. – Т. 11. – С. 538-541.

17. Lek H. H., Poo D. C. C. Aspect-based Twitter sentiment classification //2013 IEEE 25th International Conference on Tools with Artificial Intelligence. – IEEE, 2013. – С. 366-373.

18. Levy, O. Improving Distributional Similarity with Lessons Learned from Word Em-beddings / O. Levy, Y. Goldberg, I. Dagan // Transactions of the Association for Com-putational Linguistics. – 2015. – P. 211–225.

19. Loukachevitch N., Rubtsova Y. Entity-Oriented Sentiment Analysis of Tweets: Results and Problems //Text, Speech, and Dialogue. – Springer International Publishing, 2015. – С. 551-559.

20. Loukachevitch, N., and Rubtsova, Y. SentiRuEval-2016: Overcoming Time Gap and Data Sparsity in Tweet Sentiment Analysis. // In Proceedings of International Confer-ence on Computational Linguistics and Intellectual Technologies Dialog-2016. – 2016. – С. 375-384.

21. Manning C. D., Schutze H. Foundations of Statistical Natural Language Processing // The MIT Press, 1999.

22. Mansour R. et al. Revisiting The Old Kitchen Sink: Do We Need Sentiment Domain Adaptation? //RANLP. – 2013. – С. 420-427.

23. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

24. Pak A., Paroubek P. Twitter as a Corpus for Sentiment Analysis and Opinion Mining //LREC. – 2010. – T. 10. – C. 1320-1326.

25. Pang B., Lee L. Thumbs up? Sentiment classification using machine learning techniques. Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP). Philadelphia: ACL. 2002. pp. 79–86.

26. Saif Mohammad, Svetlana Kiritchenko S. and Xiaodan Zhu. 2013. NRC-Canada: Build-ing the state-of-the-art in sentiment analysis of tweets. In: Proceedings of the Second Joint Conference on Lexical and Computational Semantics (SEMSTAR'13).

27. Taboada, Maite, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede. Lexicon-based methods for sentiment analysis. Computational Linguistics, 2011. 37(2): p. 267-307.

28. Titov, I. Modeling Online Reviews with Multi-grain Topic Models / I. Titov, R. McDonald // Proceedings of the 17th International Conference on World Wide Web (WWW'08). – 2008. – P. 111–120.

29. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Dis-tributed Representations of Words and Phrases and their Compositionality. In Proceed-ings of NIPS, 2013. – Pp. 3111-3119.

30. Turney P. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews // Proceedings of ACL-02, 40th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2002, pp. 417–424.

31. Wilson T., Wiebe J. and Hoffmann P. Recognizing contextual polarity in phrase level sentiment analysis. In Proc: of Human Languages Technologies Conference/ Conference on Emperical Methods in Natural Language Processing (HLT/EMNLP 2005), Vancou-ver, CA, 2005.

УДК 004.8

Formalisms for conceptual design of closed information systems*

Anureev I.S. (Institute of Informatics Systems)

A closed information system is an information system such that its environment does not change it, and there is an information transfer from it to its environment and from its environment to it. In this paper two formalisms (information query systems and conceptual configuration systems) for abstract unified modelling of the artifacts (concept sketches and models) of the conceptual design of closed information systems, early phase of information systems design process, are proposed. Information query systems defines the abstract unified information model for the artifacts, based on such general concepts as state, information query and answer. Conceptual configuration systems are a formalism for conceptual modelling of information query systems. They defines the abstract unified conceptual model for the artifacts. The basic definitions of the theory of conceptual configuration systems are given. These systems were demonstrated to allow to model both typical and new kinds of ontological elements. The classification of ontological elements based on such systems is described. A language of conceptual configuration systems is defined.

Keywords: closed information system, information query system, conceptual structure, ontology, ontological element, conceptual, conceptual state, conceptual configuration, conceptual configuration system, conceptual information query model, CCSL

1. Introduction

The conceptual models play an important role in the overall system development life cycle [1]. Numerous conceptual modelling techniques have been created, but all of them have a limited number of kinds of ontological elements and therefore can only represent ontological elements of fixed conceptual granularity. For example, entity-relationship modelling technique [2] uses two kinds of ontological elements: entities and relationships.

The purpose of the paper is propose formalisms for abstract unified modelling of the artifacts (concept sketches and models) of the conceptual design of closed information systems (IS for short) by ontological elements of arbitrary conceptual granularity. In our two stage approach the informational and conceptual aspects of the system that the conceptual model represents are

*Partially supported by RFBR under grants 15-01-05974 and 15-07-04144 and SB RAS interdisciplinary integration project No.15/10.

described by two separate formalisms. The first formalism describes the informational model of the system, and the second formalism describes the conceptual model of the informational model.

The first formalism called an information query system (IQS for short) is a system characterized by sets of states, state objects, information queries, information query objects, answers, answer objects and an interpretation function. States of an IQS models the information storage in an IS modelled by the IQS, queries of the IQS model the information transferring from an environment to the IS to get the storage content, and answers of the IQS model the information transferring from the IS to the environment initiated by these queries. State objects, query objects and answer objects are objects that can be observed in states, queries and answers, respectively. They describe the observed internal structure of states, queries and answers. The interpretation function models the information transfer from the IS to its environment and from its environment to the IS. It associates queries with functions from states to answers.

A wide variety of information systems is modelled by IQSs in the information aspect, including search services with search results as answers, factual factographic databases with factual information as answers, document databases with documents as answers, content consumption devices with content information as answers, logical systems with truth values as answers, formalisms specifying denotational semantics of programming languages with denotations as answers and so on.

We consider that the second formalism used for for conceptual modelling of IQSs must meet the following general requirements (in relation to modelling of a IQS):

1. It must model the conceptual structure of states and state objects of the IQS.
2. It must model the content of the conceptual structure.
3. It must model information queries, information query objects, answers and answer objects of the IQS.
4. It must model the interpretation function of the IQS.
5. It must be quite universal to model typical ontological elements (concepts, attributes, concept instances, relations, relation instances, types, domains, and so on.).
6. It must provide a quite complete classification of ontological elements, including the determination of their new kinds and subkinds with arbitrary conceptual granularity.
7. The model of the interpretation function must be extensible.
8. It must have language support. The language associated with the formalism must define

syntactic representations of models of states, state objects, queries, query objects, answers and answer objects and includes the set of predefined basic query models.

To our knowledge, there is no formalism that meets all the above requirements. Therefore, we propose a new formalism, conceptual configuration systems (CCS for short), that meets these requirements.

The paper has the following structure. The preliminary concepts and notation are given in section 2. The formal definition of IQSs and the basic definitions of the theory of CCSs are given in section 3. The structure of conceptuals (atomic conceptual structures of CCSs) is described in section 4. The structure of conceptual states is considered in section 5. The classification of elements of conceptual states such that concepts, attributes and individuals is presented in section 6. The structure of concepts is described in section 7. The classification and interpretation of concepts is given in 8. The structure of attributes is described in section 9. The classification and interpretation of attributes is given in 10. The classification of conceptuals and ontological elements modelled by these conceptuals is presented in section 11. Relations, types, domains and inheritance are modelled by conceptual structures of CCSs in section 12. Generic conceptuals describing sets of conceptuals satisfying a pattern are defined in section 13. The language CCSL of CCSs is described in section 14. The semantics of interpretable elements in CCSL is defined in section 15. We establish that CCSs meet the above requirements in section 16. CCSs are compared with the related formalism, abstract state machines [3, 4], in section 17.

2. Preliminaries

2.1. Sets, sequences, multisets

Let O_b be the set of objects considered in this paper. Let S_t be a set of sets. Let I_{nt}, N_t, N_{t0} and B_l be sets of integers, natural numbers, natural numbers with zero and boolean values $true$ and $false$, respectively.

Let the names of sets be represented by capital letters possibly with subscripts and the elements of sets be represented by the corresponding small letters possibly with extended subscripts. For example, i_{nt} and $i_{nt.1}$ are elements of I_{nt}.

Let S_q be a set of sequences. Let $s_{t.(*)}$, $s_{t.\{*\}}$, and $s_{t.*}$ denote sets of sequences of the forms $(o_{b.1}, \ldots, o_{b.n_{t0}})$, $\{o_{b.1}, \ldots, o_{b.n_{t0}}\}$, and $o_{b.1}, \ldots, o_{b.n_{t0}}$ from elements of s_t. For example, $I_{nt.(*)}$ is a set of sequences of the form $(i_{nt.1}, \ldots, i_{nt.n_{t0}})$, and $i_{nt.*}$ is a sequence of the form $i_{nt.1}, \ldots, i_{nt.n_{t0}}$.

Let $o_{b.1}, \ldots, o_{b.n_{t0}}$, denote $o_{b.1}, \ldots, o_{b.n_{t0}}$. Let $s_{t.(*n_{t0})}$, $s_{t.\{*n_{t0}\}}$, and $s_{t.*n_{t0}}$ denote sets of the corresponding sequences of the length n_{t0}.

Let $o_{b.1} \prec_{[\![s_q]\!]} o_{b.2}$ denote the fact that there exist $o_{b.*.1}$, $o_{b.*.2}$ and $o_{b.*.3}$ such that $s_q = o_{b.*.1},\ o_{b.1},\ o_{b.*.2},\ o_{b.2},\ o_{b.*.3}$, or $s_q = (o_{b.*.1},\ o_{b.1},\ o_{b.*.2},\ o_{b.2},\ o_{b.*.3})$.

Let $[o_b\ o_{b.1} \hookleftarrow o_{b.2}]$ denote the result of replacement of all occurrences of $o_{b.1}$ in o_b by $o_{b.2}$. Let $[s_q\ o_b \hookleftarrow_* o_{b.1}]$ denote the result of replacement of each element $o_{b.2}$ in s_q by $[o_{b.1}\ o_b \hookleftarrow o_{b.2}]$. For example, $[(a, b)\ x \hookleftarrow_* (f\ x)]$ denotes $((f\ a), (f\ b))$.

Let $[len\ s_q]$ denote the length of s_q. Let und denote the undefined value. Let $[s_q\ .\ n_t]$ denote the n_t-th element of s_q. If $[len\ s_q] < n_t$, then $[s_q\ .\ n_t] = und$. Let $[s_q\ +\ s_{q.1}]$, $[o_b\ .+\ s_q]$ and $[s_q\ +.\ o_b]$ denote $o_{b.*}, o_{b.*.1}$, $o_b, o_{b.*}$ and $o_{b.*}, o_b$, where $s_q = o_{b.*}$ and $s_{q.1} = o_{b.*.1}$.

Let $[and\ s_q]$ denote $(c_{nd.1}\ and\ \ldots\ and\ c_{nd.n_t})$, where $s_q = c_{nd.1}$, ..., $c_{nd.n_t}$, and $[and]$ denote $true$. In the case of $n_t = 1$, the brackets can be omitted.

Let $o_{b.1}, o_{b.2} \in S_t \cup S_q$. Then $o_{b.1} =_{st} o_{b.2}$ denote that the sets of elements of $o_{b.1}$ and $o_{b.2}$ coincide, and $o_{b.1} =_{ml} o_{b.2}$ denote that the multisets of elements of $o_{b.1}$ and $o_{b.2}$ coincide.

2.2. Contexts

The terms used in the paper are context-dependent.

Let L_b be a set of objects called labels. Contexts have the form $[\![o_{b.*}]\!]$, where the elements of $o_{b.*}$ called embedded contexts have the form: l_b:o_b, l_b: or o_b.

The context in which some embedded contexts are omitted is called a partial context. All omitted embedded contexts are considered bound by the existential quantifier, unless otherwise specified.

Let $o_b[\![o_{b.*}]\!]$ denote the object o_b in the context $[\![o_{b.*}]\!]$.

The object 'in $[\![o_b, o_{b.*}]\!]$' can be reduced to 'in $[\![o_b]\!]$ in $[\![o_{b.*}]\!]$' if this does not lead to ambiguity.

2.3. Functions

Let F_n be a set of functions. Let A_{rg} and V_l be sets of objects called arguments and values. Let $[f_n\ a_{rg.*}]$ denote the application of f_n to $a_{rg.*}$.

Let $[support\ f_n]$ denote the support in $[\![f_n]\!]$, i. e. $[support\ f_n] = \{a_{rg} : [f_n\ a_{rg}] \neq und\}$. Let $[image\ f_n\ s_t]$ denote the image in $[\![f_n, s_t]\!]$, i. e. $[image\ f_n\ s_t] = \{[f_n\ a_{rg}] : a_{rg} \in s_t\}$. Let $[image\ f_n]$ denote the image in $[\![f_n, [support\ f_n]]\!]$. Let $[narrow\ f_n\ s_t]$ denote the function $f_{n.1}$ such that $[support\ f_{n.1}] = [support\ f_{n.1}] \cap s_t$, and $[f_{n.1}\ a_{rg}] = [f_n\ a_{rg}]$ for each $a_{rg} \in [support\ f_{n.1}]$.

The function $f_{n.1}$ is called a narrowing of f_n to s_t. Let $[support\ f_{n.1}] \cap [support\ f_{n.2}] = \emptyset$. Let $f_{n.1} \cup f_{n.2}$ denote the union f_n of $f_{n.1}$ and $f_{n.2}$ such that $[f_n\ a_{rg}] = [f_{n.1}\ a_{rg}]$ for each $a_{rg} \in [support\ f_{n.1}]$, and $[f_n\ a_{rg}] = [f_{n.2}\ a_{rg}]$ for each $a_{rg} \in [support\ f_{n.2}]$. Let $f_{n.1} \subseteq f_{n.2}$ denote the fact that $[support\ f_{n.1}] \subseteq [support\ f_{n.2}]$, and $[f_{n.1}\ a_{rg}] = [f_{n.2}\ a_{rg}]$ for each $a_{rg} \in [support\ f_{n.1}]$.

An object u_p of the form $a_{rg} : v_l$ is called an update. Let U_p be a set of updates. The objects a_{rg} and v_l are called an argument and value in $[\![u_p]\!]$.

Let $[f_n\ u_p]$ denote the function $f_{n.1}$ such that $[f_{n.1}\ a_{rg}] = [f_n\ a_{rg}]$ if $a_{rg} \neq a_{rg}[\![u_p]\!]$, and $[f_{n.1}\ a_{rg}[\![u_p]\!]] = v_l[\![u_p]\!]$. Let $[f_n\ u_p, u_{p.*n_t}]$ be a shortcut for $[[f_n\ u_p]\ u_{p.*n_t}]$. Let $[f_n\ a_{rg}.a_{rg.1}.\ \ldots\ .a_{rg.n_t} : v_l]$ be a shortcut for $[f_n\ a_{rg} : [[f_n\ a_{rg}]\ a_{rg.1}.\ \ldots\ .a_{rg.n_t} : v_l]]$. Let $[u_{p.*}]$ be a shortcut for $[f_n\ u_{p.*}]$, where $[support\ f_n] = \emptyset$.

Let C_{nd} be a set of objects called conditions. Let $[if\ c_{nd}\ then\ o_{b.1}\ else\ o_{b.2}]$ denote the object o_b such that

- if $c_{nd} = true$, then $o_b = o_{b.1}$;
- if $c_{nd} = false$, then $o_b = o_{b.2}$.

2.4. Attributes and multi-attributes

An object $o_{b.ma}$ of the form $(u_{p.*})$ is called a multi-attribute object. Let $O_{b.ma}$ be a set of multi-attribute objects. The elements of $[o_{b.ma}\ w \hookleftarrow_* a_{rg}[\![w]\!]]$ are called multi-attributes in $[\![o_{b.ma}]\!]$. Let $O_{b.ma}$ be a set of multi-attributes. The elements of $[o_{b.ma}\ w \hookleftarrow_* v_l[\![w]\!]]$ are called values in $[\![o_{b.ma}]\!]$. The sequence $u_{p.*}$ is called a sequence in $[\![o_{b.ma}]\!]$ and denoted by $[sequence\ in\ o_{b.ma}]$. An object v_l is a value in $[\![a_{tt.m}, o_{b.ma}]\!]$ if $o_{b.ma} = (u_{p.*.1}, a_{tt.m} : v_l, u_{p.*.2})$ for some $u_{p.*.1}$ and $u_{p.*.2}$.

An object $o_{b.ma}$ is an attribute object if the elements of $[o_{b.ma}\ w \hookleftarrow_* a_{rg}[\![w]\!]]$ are pairwise distinct. Let $O_{b.a}$ be a set of attribute objects. The multi-attributes in $[\![o_{b.a}]\!]$ are called attributes in $[\![o_{b.a}]\!]$. Let A_{tt} be a set of objects called attributes.

Let $[function\ o_{b.a}]$, $[o_{b.a}\ a_{tt}]$, and $[support\ o_{b.a}]$ denote $[[sequence\ in\ o_{b.a}]]$, $[[function\ o_{b.a}]\ a_{tt}]$, and $[support\ [function\ o_{b.a}]]$.

Let $[seq{-}to{-}att{-}obj\ s_q]$ denote $(1 : [s_q\ .\ 1], ..., [len\ s_q] : [s_q\ .\ [len\ s_q]])$. Let $o_{b.a} =_{st} (1 : v_{l.1}, ..., n_t : v_{l.n_t})$. Then $[att{-}obj{-}to{-}seq\ o_{b.a}]$ denote $(v_{l.1}, ..., v_{l.n_t})$.

3. Basic definitions of the theory of conceptual configuration systems

3.1. Information query systems

Let S_{tt} be a state of objects called states. An object $s_{s.q.i}$ of the form $(S_{tt}, O_{b.s}, Q_r, O_{b.q}, A_{ns}, O_{b.a}, value)$ is an information query system if S_{tt}, $O_{b.s}$, Q_r, $O_{b.q}$, A_{ns} and $O_{b.a}$ are nonempty sets, $S_{tt} \subseteq O_{b.s}$, $Q_r \subseteq O_{b.q}$, $und \in A_{ns}$, $A_{ns} \subseteq O_{b.a}$, $value \in Q_r \times S_{tt} \to A_{ns}$, and for all $q_r \in Q_r$ there exists $s_{tt} \in S_{tt}$ such that $[value\ q_r\ s_{tt}] \neq und$. Let $S_{s.q.i}$ be a set of information query systems.

The elements of S_{tt}, $O_{b.s}$, Q_r, $O_{b.q}$, A_{ns} and $O_{b.a}$ are called states, state objects, information queries, information query objects, answers and answer objects in $[\![s_{s.q.i}]\!]$, respectively. The function *value* is called a query interpretation in $[\![s_{s.q.i}]\!]$. An object $o_{b.s}$ is a proper state object if $o_{b.s} \notin S_{tt}$. An object $o_{b.q}$ is a proper query object if $o_{b.q} \notin Q_r$. An object $o_{b.a}$ is a proper answer object if $o_{b.a} \notin A_{ns}$.

As a through illustrative example of the IQS modelled by CCSs we use the geometric system that includes the following proper state objects:

- kinds of geometric spaces (Euclidean, Riemannian, Lobachevskian and so on);
- kinds of geometric figures (triangles, rectangles, cubes and so on);
- numerical characteristics of geometric figures (length, area, volume and so on);
- units of measurement of numerical characteristics (inches, centimeters, metres and so on);
- values of numerical characteristics represented by real numbers;
- numeral systems for representing values of numerical characteristics (binary, octal, decimal and so on);
- dimensions of geometric spaces represented by natural numbers;
- named geometric figures represented by elements of the set F_g).

A state of the geometric system is a set of relations between proper state objects. For example, the relation $\{figure : f_g, kind : triangle, space : Euclidean\}$ in $[\![s_{tt}]\!]$ means that f_g is a triangle in Euclidean space in $[\![s_{tt}]\!]$, the relation $\{figure : f_g, characteristic : perimeter, value : 20\}$ in $[\![s_{tt}]\!]$ means that perimeter of f_g equals 20 in $[\![s_{tt}]\!]$, and the relation $\{kind : cube, space : Euclidean, characteristic : volume, unit : inch)$ in $[\![s_{tt}]\!]$ means volume of cubes in Euclidean space measured in inches in $[\![s_{tt}]\!]$.

The possible queries in the geometric system can be "area of f_g", "f_g is a triangle" and "unit of measurement of perimeter of f_g" returning a number, boolean value and unit of measurement as answers.

3.2. Atoms

A set A_{tm} is a set of atoms if $I_{nt} \cup \{true, und\} \subseteq A_{tm}$. Structures of CCSs are constructed from atoms. Therefore, they are implicitly defined in $[\![A_{tm}]\!]$.

$\bigoplus$ Let $F_g \subseteq A_{tm}$.

3.3. Elements

Elements are basic structures of CCSs. They model query objects, answer objects and some proper state objects of IQSs. Let E_l be a set of objects called elements. An object e_l of the forms a_{tm}, $e_{l.(*)}$, $e_{l.1} : e_{l.2}$, or $e_{l.1} :: e_{l.2}$ is called an element.

An element $e_{l.(*)}$ of the form $(e_{l.*})$ is called a sequence element. The object $e_{l.*}$ is called a sequence in $[\![e_{l.(*)}]\!]$ and denoted by $[sequence\ in\ e_{l.(*)}]$. The element $()$ is called an empty element.

An element $u_{p.e}$ of the form $a_{tt} : v_l$ is called an element update. Let $U_{p.e}$ be a set of element updates. The elements a_{tt} and v_l are called an attribute and value in $[\![u_{p.e}]\!]$.

Let S_{rt} be a set of objects called sorts. An element $e_{l.s}$ of the form $e_l :: s_{rt}$ is called a sorted element. Let $E_{l.s}$ be a set of sorted elements. The elements e_l and s_{rt} are called an element and sort in $[\![e_l]\!]$.

An element e_{xc} of the form $e_l :: exc$ is called an exception. Let E_{xc} be a set of exceptions. The element e_l is called a value in $[\![e_{xc}]\!]$. Thus, the sort exc specifies exceptions. Exceptions in CCSs play the role that is analogous to the role of exceptions in programming languages. An element e_l is abnormal if $e_l \in E_{xc}$, or $e_l = und$. Let $E_{l.ab}$ be a set of abnormal elements. An element e_l is normal if e_l is not abnormal. Let $E_{l.n}$ be a set of normal elements.

An element $e_{l.ma}$ is a multi-attribute element if $e_l \in O_{b.ma}$. Let $E_{l.ma}$ be a set of multi-attribute elements. An element $e_{l.a}$ is an attribute element if $e_l \in O_{b.a}$. Let $E_{l.a}$ be a set of attribute elements.

$\bigoplus$ The element $(f_g,\ is,\ triangle)$ means that f_g is a triangle.

3.4. Conceptuals

Conceptuals are atomic conceptual structures of CCSs. Conceptual structures of CCSs are constructed from conceptuals. Conceptuals model some proper state objects of IQSs. An attribute element c_{ncpl} is a conceptual if $[support\ c_{ncpl}] \subseteq I_{nt}$. Let C_{ncpl} be a set of conceptuals. An element of the form $i_{nt} : e_l$ is called a conceptual update. Let $U_{p.c}$ be a set of conceptual updates.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : \mathit{inch}, -1 : \mathit{area}, 0 : f_g, 1 : \mathit{triangle}, 2 : \mathit{Euclidean}, 3 : 2)$. Then the following properties hold:

– c_{ncpl} is a conceptual;

– $-3 : 10$, $-2 : \mathit{inch}$, $-1 : \mathit{area}$, $0 : f_g$, $1 : \mathit{triangle}$, $2 : \mathit{Euclidean}$ and $3 : 2$ are conceptual updates;

– c_{ncpl} models the area (the attribute -1) of the triangle (the attribute 1) f_g (the attribute 0) in three-dimensional (the attribute 3) Euclidean (the attribute 2) space, measured in inches (the attribute -2) in the decimal system (the attribute -3).

3.5. Conceptual states

Conceptual states are conceptual structures of CCSs specifying values of conceptuals. They model some proper state objects of IQSs. An attribute element s_{tt} is a conceptual state if $[\mathit{support}\ s_{tt}] \subseteq C_{ncpl}$. Thus, s_{tt} can reference to either a state of a IQS or a conceptual state of a QTS depending on the context.

A function $\mathit{value} \in C_{ncpl} \times S_{tt} \to E_l$ is a conceptual interpretation if $[\mathit{value}\ c_{ncpl}\ s_{tt}] = [s_{tt}\ c_{ncpl}]$. The element $[\mathit{value}\ c_{ncpl}\ s_{tt}]$ is called a value in $[\![c_{ncpl}, s_{tt}]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : \mathit{inch}, -1 : \mathit{area}, 0 : f_g, 1 : \mathit{triangle}, 2 : \mathit{Euclidean}, 3 : 2)$ and $s_{tt} = (c_{ncpl} : 3)$. Then the following properties hold:

– $[\mathit{value}\ c_{ncpl}\ s_{tt}] = 3$;

– 3 is the value in $[\![c_{ncpl}, s_{tt}]\!]$;

– area of the triangle f_g in two-dimensional Euclidean space equals 3 inches in the decimal system in $[\![s_{tt}]\!]$.

3.6. Conceptual configurations

Conceptual configurations are conceptual structures of CCSs partitioning states into named substates. They model states of IQSs. Let N_m be a set of objects called names. An attribute element c_{nf} is a conceptual configuration if $[\mathit{image}\ c_{nf}] \subseteq S_{tt}$. Let C_{nf} be a set of configurations. An element n_m is a name in $[\![c_{nf}]\!]$ if $n_m \in [\mathit{support}\ c_{nf}]$. An element n_m is a name in $[\![s_{tt}, c_{nf}]\!]$ if $[c_{nf}\ n_m] = s_{tt}$. An element s_{tt} is a substate in $[\![c_{nf}]\!]$ if $s_{tt} \in [\mathit{image}\ c_{nf}]$. An element s_{tt} is a substate in $[\![n_m, c_{nf}]\!]$ if $[c_{nf}\ n_m] = s_{tt}$. A substate s_{tt} is unnamed in $[\![c_{nf}]\!]$ if $[c_{nf}\ ()] = s_{tt}$. The element () is called an unnamed substate specifier.

A function $\mathit{value} \in C_{ncpl} \times E_l \times C_{nf} \to E_l$ is a conceptual interpretation if $[\mathit{value}\ c_{ncpl}\ n_m$

$c_{nf}] = [value\ c_{ncpl}\ [c_{nf}\ n_m]]$. The element $[value\ c_{ncpl}\ n_m\ c_{nf}]$ is called a value in $[\![c_{ncpl}, n_m, c_{nf}]\!]$.

An element $s_{tt.n}$ of the form $s_{tt} :: state :: n_m$ is called a named state. Let $S_{tt.n}$ be a set of named states. The elements s_{tt} and n_m are called a state and name in $[\![s_{tt.n}]\!]$. The element s_{tt} references to $s_{tt} :: state :: ()$ in the context of named states.

An element $c_{ncpl.n}$ of the form $c_{ncpl} :: state :: n_m$ is called a named conceptual. Let $C_{ncpl.n}$ be a set of named conceptuals. It specifies the conceptual c_{ncpl} in the state with the name n_m. The elements c_{ncpl} and n_m are called a conceptual and name in $[\![c_{ncpl.n}]\!]$. The element c_{ncpl} references to $c_{ncpl} :: state :: ()$ in the context of named conceptuals.

A function $value \in C_{ncpl.n} \times C_{nf} \to E_l$ is a conceptual interpretation if $[value\ c_{ncpl.n}\ c_{nf}] = [value\ c_{ncpl}[\![c_{ncpl.n}]\!]\ n_m[\![c_{ncpl.n}]\!]\ c_{ncpl}]$. The element $[value\ c_{ncpl.n}\ c_{nf}]$ is called a value in $[\![c_{ncpl.n}, c_{nf}]\!]$.

3.7. Substitutions, patterns, pattern specifications, instances

A function $s_b \in E_l \to E_{l.*}$ is called a substitution. Let S_b be a set of substitutions. A function $subst \in S_b \times E_{l.*} \to E_{l.*}$ is a substitution function if it is defined as follows (the first proper rule is applied):

- if $e_l \in [support\ s_b]$, then $[subst\ s_b\ e_l] = [s_b\ e_l]$;
- $[subst\ s_b\ a_{tm}] = a_{tm}$;
- $[subst\ s_b\ l_b : e_l] = [subst\ s_b\ l_b] : [subst\ s_b\ e_l]$;
- $[subst\ s_b\ e_l :: nosubst] = e_l$;
- $[subst\ s_b\ e_l :: (nosubstexcept\ e_{l.*})] = [subst\ [narrow\ s_b\ \{e_{l.*}\}]\ e_l]$;
- $[subst\ s_b\ e_l :: s_{rt}] = [subst\ s_b\ e_l] :: [subst\ s_b\ s_{rt}]$;
- $[subst\ s_b\ (e_{l.*})] = ([e_{l.*}\ w \hookleftarrow_* [subst\ s_b\ w]])$;
- $[subst\ s_b\ e_{l.*}] = [e_{l.*}\ w \hookleftarrow_* [subst\ s_b\ w]]$.

The sort *nosubst* specifies the elements to which the substitution s_b is not applied. The sort $(nosubstexcept\ e_{l.*})$ specifies the elements to which the narrowing of the substitution s_b to the set $e_{l.*}$ is applied. An element p_t is a pattern in $[\![e_l, s_b]\!]$ if $[subst\ s_b\ p_t] = e_l$. Let P_t be a set of patterns. An element i_{nst} is an instance in $[\![p_t, s_b]\!]$ if $[subst\ s_b\ p_t] = i_{nst}$. Let I_{nst} be a set of instances.

Let V_r and $V_{r.s}$ be sets of objects called element variables and sequence variables, respectively. An element $p_{t.s}$ of the form $(p_t, (v_{r.*}), (v_{r.s.*}))$ is a pattern specification if $\{v_{r.s.*}\} \cap \{v_{r.*}\} = \emptyset$, and the elements of $\{v_{r.*}\} \cup \{v_{r.s.*}\}$ are pairwise distinct. Let $P_{t.s}$ be a set of pattern specifications.

The objects p_t, $(v_{r.*})$, and $(v_{r.s.*})$ are called a pattern, element variable specification, and sequence variable specification in $[\![p_{t.s}]\!]$. The elements of $v_{r.*}$ and $v_{r.s.*}$ are called element pattern variables and sequence pattern variables in $[\![p_{t.s}]\!]$, respectively.

An element i_{nst} is an instance in $[\![p_{t.s}, s_b]\!]$ if $[support\ s_b] = \{v_{r.*}\}$, $[s_b\ v_r] \in E_l$ for $v_r \in \{v_{r.*}\} \setminus \{v_{r.s.*}\}$, $[s_b\ v_r] \in E_{l.*}$ for $v_r \in \{v_{r.s.*}\}$, and i_{nst} is an instance in $[\![p_t, s_b]\!]$. An element i_{nst} is an instance in $[\![p_{t.s}]\!]$ if there exists s_b such that i_{nst} is an instance in $[\![p_{t.s}, s_b]\!]$.

A function $m_t \in E_l \times P_{t.s} \to S_b$ is a match if the following property holds:

- if $[m_t\ e_l\ p_{t.s}] = s_b$, then e_l is an instance in $[\![p_{t.s}, s_b]\!]$.

An element i_{nst} is an instance in $[\![p_{t.s}, m_t, s_b]\!]$ if $[m_t\ i_{nst}\ p_{t.s}] = s_b$. An element i_{nst} is an instance in $[\![p_{t.s}, m_t]\!]$ if there exists s_b such that i_{nst} is an instance in $[\![p_{t.s}, m_t, s_b]\!]$.

3.8. The element interpretation

Queries and answers of a IQS is modelled by elements, and the query interpretation of the IQS is modelled by the element interpretation $value \in E_l \times C_{nf} \to E_l$ based on atomic element interpretations, element definitions and the element interpretation order.

The special variable $conf :: in$ references to the current configuration in the definitions below.

An object $i_{ntr.a}$ of the form $(p_t, (v_{r.*}), (v_{r.s.*}), f_n)$ is an atomic element interpretation if $(p_t, (v_{r.*}), (v_{r.s.*}))$ is a pattern specification, $conf :: in \notin \{v_{r.*}\} \cup \{v_{r.s.*}\}$, $f_n \in S_b \to E_l$, $[support\ f_n] = \{s_b : [support\ s_b] = \{v_{r.*}\} \cup \{v_{r.s.*}\} \cup \{conf :: in\}, [s_b\ v_r] \in E_l$ for $v_r \in \{v_{r.*}\}$, and $[s_b\ v_r] \in E_{l.*}$ for $v_r \in \{v_{r.s.*}\}\}$. Let $I_{ntr.a}$ be a set of atomic element interpretations.

The objects p_t, $(v_{r.*})$, $(v_{r.s.*})$, and f_n are called a pattern, element variable specification, sequence variable specification, and value in $[\![i_{ntr.a}]\!]$. The elements of $v_{r.*}$ and $v_{r.s.*}$ are called element pattern variables and sequence pattern variables in $[\![i_{ntr.a}]\!]$, respectively.

A function $i_{ntr.a.s} \in E_l \to I_{ntr.a}$ is called an atomic element interpretation specification if $[support\ i_{ntr.a.s}]$ is finite. An interpretation $i_{ntr.a}$ is an atomic element interpretation in $[\![i_{ntr.a.s}]\!]$ if $[i_{ntr.a.s}\ n_m] = i_{ntr.a}$ for some $n_m \in E_l$. An element n_m is a name in $[\![i_{ntr.a}, i_{ntr.a.s}]\!]$ if $[i_{ntr.a.s}\ n_m] = i_{ntr.a}$. An element n_m a name in $[\![i_{ntr.a.s}]\!]$ if n_m is a name in $[\![i_{ntr.a}, i_{ntr.a.s}]\!]$ for some $i_{ntr.a}$.

An element d_f of the form $(p_t, (v_{r.*}), (v_{r.s.*}), b_d)$ is an element definition if $(p_t, (v_{r.*}), (v_{r.s.*}))$ is a pattern specification, and $conf :: in \notin \{v_{r.*}\} \cup \{v_{r.s.*}\}$. Let D_f be a set of element definitions.

The objects p_t, $(v_{r.*})$, $(v_{r.s.*})$ and b_d are called a pattern, element variable specification, sequence variable specification and body in $[\![d_f]\!]$. The elements of $v_{r.*}$ and $v_{r.s.*}$ are called element pattern variables and sequence pattern variables in $[\![d_f]\!]$, respectively.

An attribute element $d_{f.s}$ is called an element definition specification if $[support\ d_{f.s}] \subseteq E_l$, and $[image\ d_{f.s}] \subseteq D_f$. A definition d_f is an element definition in $[\![d_{f.s}]\!]$ if $[d_{f.s}\ n_m] = d_f$ for some $n_m \in E_l$. An element n_m is a name in $[\![d_f, d_{f.s}]\!]$ if $[d_{f.s}\ n_m] = d_f$. An element n_m a name in $[\![d_{f.s}]\!]$ if n_m is a name in $[\![d_f, d_{f.s}]\!]$ for some d_f.

Let $[support\ i_{ntr.a.s}] \cap [support\ d_{f.s}] = \emptyset$.

An element $o_{rd.intr}$ of the form $(n_{m.*})$ is called an element interpretation order in $[\![i_{ntr.a.s}, d_{f.s}]\!]$ if $\{n_{m.*}\} \subseteq [support\ i_{ntr.a.s}] \cup [support\ d_{f.s}]$, and the elements of $n_{m.*}$ are pairwise distinct. It specifies the order of application of atomic element interpretations and element definitions to the element to be interpreted.

The information about the element definition specification and element interpretation order of configurations is stored in the substate *interpretation* of the configurations. The conceptuals $(0 : definitions) :: state :: interpretation$ and $(0 : order) :: state :: interpretation$ define the element definition specification and element interpretation order of the configurations, respectively.

An element c_{nf} is consistent with $(i_{ntr.a.s}, d_{f.s}, o_{rd.intr})$ if the following properties hold:

- if $[support\ i_{ntr.a.s}] \cap [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] = \emptyset$;
- $d_{f.s} \subseteq [c_{nf}\ (0 : definitions) :: state :: interpretation]$;
- if $n_{m.1} \prec_{[\![o_{rd.intr}]\!]} n_{m.2}$, and $n_{m.1},\ n_{m.2} \in [c_{nf}\ (0 : order) :: state :: interpretation]$, then $n_{m.1} \prec_{[\![[c_{nf}\ (0:order)::state::interpretation]]\!]} n_{m.2}$.

A function $value \in E_l \times C_{nf} \to E_l$ is an element interpretation in $[\![i_{ntr.a.s}, d_{f.s}, o_{rd.intr}, m_t]\!]$ if $[value\ e_l\ c_{nf}] = [value\ e_l\ c_{nf}\ [c_{nf}\ (0 : order) :: state :: interpretation]]$. It specifies interpretation of elements in the context of configurations. The element $[value\ e_l\ c_{nf}]$ is called a value in $[\![e_l, c_{nf}]\!]$.

The auxiliary function $value \in E_l \times C_{nf} \times N_{m.(*)} \to E_l$ is defined by the following rules (the first proper rule is applied):

- if c_{nf} is not consistent with $(i_{ntr.a.s}, d_{f.s}, o_{rd.intr})$, then $[value\ e_l\ c_{nf}\ n_{m.(*)}] = und$;
- if $i_{ntr.a} = [i_{ntr.a.s}\ n_m]$, e_l is an instance in $[\![p_{t.s}[\![i_{ntr.a}]\!], m_t, s_b]\!]$, and $[f_n[\![i_{ntr.a}]\!]\ s_b \cup (conf :: in : c_{nf})] \neq und$, then $[value\ e_l\ c_{nf}\ (n_m\ n_{m.*})] = [f_n[\![i_{ntr.a}]\!]\ [s_b\ conf : c_{nf}]]$;
- if $d_f = [[c_{nf}\ (0 : definitions) :: state :: interpretation]\ n_m]$, e_l is an instance in

$[\![p_{t.s}[\![d_f]\!], m_t, s_b]\!]$, and $[value\ [subst\ s_b \cup (conf\ ::\ in\ :\ c_{nf})\ b_d[\![d_f]\!]]\ c_{nf}] \neq und$, then $[value\ e_l\ c_{nf}\ (n_m\ n_{m.*})] = [value\ [subst\ [s_b\ conf : c_{nf}]\ b_d[\![d_f]\!]]\ c_{nf}]$;

- $[value\ e_l\ c_{nf}\ (n_m\ n_{m.*})] = [value\ e_l\ c_{nf}\ (n_{m.*})]$;
- $[value\ e_l\ c_{nf}\ ()] = und$.

3.9. Satisfiable and valid elements

An element e_l is satisfiable in $[\![(v_{r.*}), c_{nf}]\!]$ if there exists s_b such that $[support\ s_b] = \{v_{r.*}\}$, and $[value\ [subst\ s_b\ e_l]\ c_{nf}] \neq und$.

An element e_l is valid in $[\![(v_{r.*}), c_{nf}]\!]$ if $[value\ [subst\ s_b\ e_l]\ c_{nf}] \neq und$ for each s_b such that $[support\ s_b] = \{v_{r.*}\}$.

3.10. Conceptual configuration systems

An object $s_{s.c.c}$ of the form $(A_{tm}, i_{ntr.a.s}, d_{f.s}, o_{rd.intr}, m_t)$ is called a conceptual configuration system if $i_{ntr.a.s}$, $d_{f.s}$, $o_{rd.intr}$ and m_t are an atomic element interpretation specification, element definition specification element interpretation order and match in $[\![A_{tm}]\!]$, and $[support\ i_{ntr.a.s}] \cap [support\ d_{f.s}] = \emptyset$. Let $S_{s.c.c}$ be a set of conceptual configuration systems.

The elements of A_{tm}, $E_l[\![A_{tm}]\!]$, $C_{ncpl}[\![A_{tm}]\!]$, $S_{tt}[\![A_{tm}]\!]$ and $C_{nf}[\![A_{tm}]\!]$ are called atoms, elements, conceptuals, states and configurations in $[\![s_{s.t.c}]\!]$.

The objects $i_{ntr.a.s}$, $d_{f.s}$, $o_{rd.intr}$ and m_t are called atomic element interpretation specification, element definition specification, element interpretation order and match in $[\![s_{s.c.c}]\!]$.

An element e_l is interpretable in $[\![s_{s.c.c}]\!]$ if there exist n_m such that e_l is an instance in $[\![p_{t.s}[\![[i_{ntr.a.s}\ n_m]]\!], m_t]\!]$, or e_l is an instance in $[\![p_{t.s}[\![[d_{f.s}\ n_m]]\!], m_t]\!]$.

3.11. Conceptual information query models

An object $m_{dl.q.i.c}$ of the form $(s_{s.c.c}, r_{pr.s}, r_{pr.q}, r_{pr.a})$ is a conceptual information query model in $[\![s_{s.q.i}]\!]$ if $r_{pr.s}, r_{pr.q}, r_{pr.a} \in F_n$, $[support\ r_{pr.s}] = O_{b.s}[\![s_{s.q.i}]\!]$, $[image\ r_{pr.s}] \subseteq E_l[\![s_{s.c.c}]\!]$, $[image\ r_{pr.s}\ S_{tt}[\![s_{s.q.i}]\!]] \subseteq C_{nf}[\![s_{s.c.c}]\!]$, $[support\ r_{pr.q}] = O_{b.q}[\![s_{s.q.i}]\!]$, $[image\ r_{pr.q}] \subseteq E_l[\![s_{s.c.c}]\!]$, $[support\ r_{pr.a}] = O_{b.a}[\![s_{s.q.i}]\!]$, $[image\ r_{pr.a}] \subseteq E_l[\![s_{s.c.c}]\!]$, and $[r_{pr.a}\ [value\ q_r\ s_{tt}]] = [value\ [r_{pr.q}\ q_r]\ [r_{pr.s}\ s_{tt}]]$. Let $M_{dl.q.i.c}$ be a set of conceptual information query models.

The system $s_{s.c.c}$ is called a conceptual configuration system in $[\![m_{dl.q.i.c}]\!]$. The functions $r_{pr.s}$, $r_{pr.q}$ and $r_{pr.a}$ are called a state representation, query representation and answer representation in $[\![m_{dl.q.i.c}]\!]$, respectively.

A system $s_{s.q.i}$ is conceptually modelled in $[\![s_{s.c.c}]\!]$ if there exists $m_{dl.q.i.c}$ such that $s_{s.c.c} = s_{s.c.c}[\![m_{dl.q.i.c}]\!]$, and $m_{dl.q.i.c}$ is a conceptual query model in $[\![s_{s.q.i}]\!]$. The set $[image\ r_{pr.s}]$ is called an ontology in $[\![s_{s.q.i}, m_{dl.q.i.c}]\!]$. It includes conceptual structures of $s_{s.c.c}[\![m_{dl.q.i.c}]\!]$ representing the conceptual structure of state objects in $[\![s_{s.q.i}]\!]$.

Let $r^{-}_{pr.s}$, $r^{-}_{pr.q}$ and $r^{-}_{pr.a}$ denote the inverse functions of $r_{pr.s}$, $r_{pr.q}$ and $r_{pr.a}$ in the case of their existence.

3.12. Extensions

A system $s_{s.q.i.1}$ is an extension of $s_{s.q.i.2}$ if $s_t[\![s_{s.q.i.1}]\!] \subseteq s_t[\![s_{s.q.i.2}]\!]$ for each $s_t \in \{S_{tt}, O_{b.s}, Q_r, O_{b.q}, A_{ns}, O_{b.a}, value\}$.

A system $s_{s.c.c.1}$ is an extension of $s_{s.c.c.2}$ if $o_b[\![s_{s.c.c.1}]\!] = o_b[\![s_{s.c.c.2}]\!]$ for each $o_b \in \{A_{tm}, m_t\}$, $s_t[\![s_{s.c.c.1}]\!] \subseteq s_t[\![s_{s.c.c.2}]\!]$ for each $s_t \in \{i_{ntr.a.s}, d_{f.s}\}$, and the following property hold:

- if $n_{m.1} \prec_{[\![o_{rd.intr}[\![s_{s.c.c.1}]\!]]\!]} n_{m.2}$, and $n_{m.1}$, $n_{m.2} \in o_{rd.intr}[\![s_{s.c.c.2}]\!]$, then $n_{m.1} \prec_{[\![o_{rd.intr}[\![s_{s.c.c.2}]\!]]\!]} n_{m.2}$.

A CCS l_n is a language of CCSs if the conceptual structures (atoms, elements, conceptuals and so on) of l_n is syntactically defined.

4. Structure of conceptuals

4.1. Elements of conceptuals

An element e_l is an element in $[\![c_{ncpl}, i_{nt}]\!]$ if $e_l = [c_{ncpl}\ i_{nt}]$ and $e_l \neq und$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then 10, *inch*, *area*, f_g, *triangle*, *Euclidean*, 2 are elements in $[\![c_{ncpl}]\!]$ in $[\![-3]\!]$, $[\![-2]\!]$, $[\![-1]\!]$, $[\![0]\!]$, $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$.

An element e_l is an element in $[\![c_{ncpl}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![c_{ncpl}, i_{nt}]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then 10, *inch*, *area*, f_g, *triangle*, *Euclidean*, 2 are elements in $[\![c_{ncpl}]\!]$.

4.2. Orders of conceptuals in the context of elements

A number i_{nt} is an order in $[\![c_{ncpl}, e_l]\!]$ if $e_l = [c_{ncpl}\ i_{nt}]$ and $e_l \neq und$. Let O_{rd} be a set of objects called orders.

⊕ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then -3, -2, -1, 0, 1, 2, 3 are orders in $[\![c_{ncpl}]\!]$ in $[\![10]\!]$, $[\![inch]\!]$, $[\![area]\!]$, $[\![f_g]\!]$, $[\![triangle]\!]$, $[\![Euclidean]\!]$, $[\![3]\!]$.

A number i_{nt} is an order in $[\![c_{ncpl}, element :]\!]$ if there exists e_l such that i_{nt} is an order in $[\![c_{ncpl}, e_l]\!]$.

⊕ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then -3, -2, -1, 0, 1, 2, 3 are orders in $[\![c_{ncpl}, element :]\!]$.

4.3. Properties of elements of conceptuals

Proposition 1. The element *und* is not an element in $[\![c_{ncpl}]\!]$.

Proof. This follows from the definition of element in $[\![c_{ncpl}]\!]$. □

Proposition 2. The number of elements in $[\![c_{ncpl}]\!]$ is finite.

Proof. This follows from the fact that $[support\ c_{ncpl}]$ is finite and *und* is not an element in $[\![c_{ncpl}]\!]$. □

4.4. Properties of orders of conceptuals in the context of elements

Proposition 3. The number of orders in $[\![c_{ncpl}, e_l[\![c_{ncpl}]\!]]\!]$ is finite.

Proof. This follows from the fact that $[support\ c_{ncpl}]$ is finite and *und* is not an element in $[\![c_{ncpl}]\!]$. □

Proposition 4. The number of orders in $[\![c_{ncpl}, element :]\!]$ is finite.

Proof. This follows from the fact that $[support\ c_{ncpl}]$ is finite. □

4.5. Kinds of orders of conceptuals in the context of elements

An order $o_{rd}[\![c_{ncpl}, e_l]\!]$ is minimal in $[\![c_{ncpl}, e_l]\!]$ if i_{nt} is not an order in $[\![c_{ncpl}, e_l]\!]$ for each i_{nt} such that $i_{nt} < o_{rd}$.

⊕ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then -2 is a minimal order in $[\![c_{ncpl}, inch]\!]$.

An order $o_{rd}[\![c_{ncpl}]\!]$ is minimal in $[\![c_{ncpl}, element :]\!]$ if i_{nt} is not an order in $[\![c_{ncpl}, \hat{e}_l]\!]$ for each i_{nt} such that $i_{nt} < o_{rd}$.

⊕ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then -3 is a minimal order in $[\![c_{ncpl}, element :]\!]$.

An order $o_{rd}[\![c_{ncpl}, e_l]\!]$ is maximal in $[\![c_{ncpl}, e_l]\!]$ if i_{nt} is not an order in $[\![c_{ncpl}, e_l]\!]$ for each i_{nt} such that $o_{rd} < i_{nt}$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 10)$. Then 2 is a maximal order in $[\![c_{ncpl}, Euclidean]\!]$.

An order $o_{rd}[\![c_{ncpl}]\!]$ is maximal in $[\![c_{ncpl}, element :]\!]$ if i_{nt} is not an order in $[\![c_{ncpl}, \hat{e}_l]\!]$ for each i_{nt} such that $o_{rd} < i_{nt}$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then 3 is a maximal order in $[\![c_{ncpl}, element :]\!]$.

4.6. Kinds of elements of conceptuals

An element e_l is minimal in $[\![c_{ncpl}]\!]$ if there exists $o_{rd}[\![c_{ncpl}, e_l]\!]$ such that o_{rd} is minimal in $[\![c_{ncpl}, element :]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then 10 is a minimal element in $[\![c_{ncpl}]\!]$.

An element e_l is maximal in $[\![c_{ncpl}]\!]$ if there exists $o_{rd}[\![c_{ncpl}, e_l]\!]$ such that o_{rd} is a maximal order in $[\![c_{ncpl}, element :]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then 2 is a maximal element in $[\![c_{ncpl}]\!]$.

An element e_l is null in $[\![c_{ncpl}]\!]$ if e_l is an element in $[\![c_{ncpl}, 0]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then f_g is null in $[\![c_{ncpl}]\!]$.

5. Structure of conceptual states

5.1. Conceptuals

A conceptual c_{ncpl} is a conceptual in $[\![s_{tt}]\!]$ if $[value\ c_{ncpl}\ s_{tt}] \neq und$.

A conceptual $c_{ncpl.n}$ is a conceptual in $[\![c_{nf}]\!]$ if $c_{ncpl}[\![c_{ncpl.n}]\!]$ is a conceptual in $[\![[c_{nf}\ n_m\ [\![c_{ncpl.n}]\!]]]\!]$. A conceptual c_{ncpl} is a conceptual in $[\![c_{nf}]\!]$ if there exists n_m such that $c_{ncpl} :: state :: n_m$ is a conceptual in $[\![c_{nf}]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$ and $[support\ s_{tt}] = \{c_{ncpl}\}$. Then c_{ncpl} is a conceptual in $[\![s_{tt}]\!]$.

5.2. Elements, orders, concretizations

An element e_l is an element in $[\![s_{tt}, i_{nt}, c_{ncpl}]\!]$ if c_{ncpl} is a conceptual in $[\![s_{tt}]\!]$ and e_l is an element in $[\![c_{ncpl}, i_{nt}]\!]$. An element e_l is an element in $[\![c_{nf}, i_{nt}, c_{ncpl.n}]\!]$ if e_l is an element in $[\![[c_{nf}\ [name\ in\ c_{ncpl.n}]], i_{nt}, [conceptual\ in\ c_{ncpl.n}]]\!]$.

A number i_{nt} is an order in $[\![e_l, s_{tt}, c_{ncpl}]\!]$ if e_l is an element in $[\![s_{tt}, i_{nt}, c_{ncpl}]\!]$. A number i_{nt} is an order in $[\![e_l, c_{nf}, c_{ncpl.n}]\!]$ if e_l is an element in $[\![c_{nf}, i_{nt}, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![e_l, s_{tt}, i_{nt}]\!]$ if e_l is an element in $[\![s_{tt}, i_{nt}, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, c_{nf}, i_{nt}]\!]$ if e_l is an element in $[\![c_{nf}, i_{nt}, c_{ncpl.n}]\!]$.

5.3. Kinds of elements

An element e_l is an element in $[\![s_{tt}, i_{nt}]\!]$ if there exists c_{ncpl} such that e_l is an element in $[\![s_{tt}, i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![c_{nf}, i_{nt}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is an element in $[\![c_{nf}, i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![s_{tt}, c_{ncpl}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![s_{tt}, i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![c_{nf}, c_{ncpl.n}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![c_{nf}, i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![s_{tt}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![s_{tt}, i_{nt}]\!]$. An element e_l is an element in $[\![c_{nf}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![c_{nf}, i_{nt}]\!]$.

5.4. Kinds of orders

A number i_{nt} is an order in $[\![e_l, s_{tt}]\!]$ if e_l is an element in $[\![s_{tt}, i_{nt}]\!]$. A number i_{nt} is an order in $[\![e_l, c_{nf}]\!]$ if e_l is an element in $[\![c_{nf}, i_{nt}]\!]$.

A number i_{nt} is an order in $[\![s_{tt}, element :]\!]$ if there exists e_l such that i_{nt} is an order in $[\![e_l, s_{tt}]\!]$. A number i_{nt} is an order in $[\![c_{nf}, element :]\!]$ if there exists e_l such that i_{nt} is an order in $[\![e_l, c_{nf}]\!]$.

5.5. Kinds of concretizations

A conceptual c_{ncpl} is a concretization in $[\![e_l, s_{tt}]\!]$ if e_l is an element in $[\![s_{tt}, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, c_{nf}]\!]$ if e_l is an element in $[\![c_{nf}, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![s_{tt}, element :]\!]$ if there exists e_l such that c_{ncpl} is a concretization in $[\![e_l, s_{tt}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![c_{nf}, element :]\!]$ if there exists e_l such that $c_{ncpl.n}$ is a concretization in $[\![e_l, c_{nf}]\!]$.

5.6. Example

⊕ Let $c_{ncl.1} = (-3 : 10, -2 : \textit{inch}, -1 : \textit{area}, 0 : e_{l.g.1}, 1 : \textit{triangle}, 2 : \textit{Euclidean}, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : \textit{cm}, -1 : \textit{volume}, 0 : e_{l.g.2}, 1 : \textit{cube}, 2 : \textit{Lobachevskian}, 3 : 3)$, and $[\textit{support } s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

– 10, 8, *inch*, *cm*, *area*, *volume*, $e_{l.g.1}$, $e_{l.g.2}$, *trianle*, *cube*, *Euclidean*, *Lobachevskian*, 3, 2 are elements in $[\![s_{tt}]\!]$;

– −3, −2, −1, 0, 1, 2, 3 are orders in $[\![s_{tt}, \textit{element} :]\!]$;

– $c_{ncl.1}$, $c_{ncl.2}$ are concretizations in $[\![s_{tt}, \textit{element} :]\!]$.

5.7. Properties of elements

Proposition 5. For all e_l and i_{nt} there exist s_{tt} and c_{ncpl} such that e_l is an element in $[\![s_{tt}, i_{nt}, c_{ncpl}]\!]$.

Proof. We define s_{tt} and c_{ncpl} as follows: $[c_{ncpl}\ i_{nt}] = e_l$ and $[s_{tt}\ c_{ncpl}] \neq \textit{und}$. Then e_l is an element in $[\![s_{tt}, i_{nt}, c_{ncpl}]\!]$. □

6. Classification of elements of states

Elements in $[\![s_{tt}]\!]$ are subclassified into individuals, concepts and attributes.

6.1. Individuals

Individuals in $[\![s_{tt}]\!]$ model elements in $[\![s_{s.q.i}]\!]$.

An element e_l is an individual in $[\![s_{tt}, c_{ncpl}]\!]$ if e_l is an element in $[\![s_{tt}, 0, c_{ncpl}]\!]$. An element e_l is an individual in $[\![c_{nf}, c_{ncpl.n}]\!]$ if e_l is an element in $[\![c_{nf}, 0, c_{ncpl.n}]\!]$.

⊕ Let $c_{ncpl} = (-3 : 10, -2 : \textit{inch}, -1 : \textit{area}, 0 : f_g, 1 : \textit{triangle}, 2 : \textit{Euclidean}, 3 : 2)$ and $s_{tt} = (c_{ncpl} : 3)$. Then f_g is an individual in $[\![s_{tt}, c_{ncpl}]\!]$.

An element e_l is an individual in $[\![s_{tt}]\!]$ if there exists c_{ncpl} such that e_l is an individual in $[\![s_{tt}, c_{ncpl}]\!]$. An element e_l is an individual in $[\![c_{nf}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is an individual in $[\![c_{nf}, c_{ncpl.n}]\!]$.

⊕ Let $c_{ncl.1} = (-3 : 10, -2 : \textit{inch}, -1 : \textit{area}, 0 : e_{l.g.1}, 1 : \textit{triangle}, 2 : \textit{Euclidean}, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : \textit{cm}, -1 : \textit{volume}, 0 : e_{l.g.2}, 1 : \textit{cube}, 2 : \textit{Lobachevskian}, 3 : 3)$, and $s_{tt} = (c_{ncl.1} : 3, c_{ncl.2} : 4)$. Then $e_{l.g.1}$ and $e_{l.g.2}$ are individuals in $[\![s_{tt}]\!]$.

6.2. Concepts

Concepts in $[\![s_{tt}]\!]$ generalize models of the usual concepts in $[\![s_{s.q.i}]\!]$ which are interpreted as sets of elements in $[\![s_{s.q.i}]\!]$.

An element e_l is a concept in $[\![s_{tt}, n_t, c_{ncpl}]\!]$ if e_l is an element in $[\![s_{tt}, n_t, c_{ncpl}]\!]$. A number n_t is an order in $[\![e_l, s_{tt}, c_{ncpl}]\!]$ in $[\![concept : e_l, s_{tt}, c_{ncpl}]\!]$ if e_l is a concept in $[\![s_{tt}, n_t, c_{ncpl}]\!]$. A conceptual c_{ncpl} is a concretization in $[\![concept : e_l, s_{tt}, n_t]\!]$ if e_l is a concept in $[\![s_{tt}, n_t, c_{ncpl}]\!]$.

An element e_l is a concept in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$ if e_l is an element in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$. A number n_t is an order in $[\![e_l, c_{nf}, c_{ncpl.n}]\!]$ in $[\![concept : e_l, c_{nf}, c_{ncpl.n}]\!]$ if e_l is a concept in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![concept : e_l, c_{nf}, n_t]\!]$ if e_l is a concept in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$.

⊕ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$ and $s_{tt} = (c_{ncpl} : 3)$. Then the following properties hold:

– *triangle*, *Euclidean*, 2 are concepts in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$ in $[\![c_{ncpl}]\!]$;

– 1, 2, 3 are orders in $[\![concept : triangle]\!]$, $[\![concept : Euclidean]\!]$, $[\![concept : 2]\!]$ in $[\![s_{tt}]\!]$ in $[\![c_{ncpl}]\!]$;

– c_{ncpl} is a concretization in $[\![concept : triangle]\!]$, $[\![concept : Euclidean]\!]$, $[\![concept : 3]\!]$ in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![2]\!]$.

An element e_l is a concept in $[\![s_{tt}, n_t]\!]$ if there exists c_{ncpl} such that e_l is a concept in $[\![s_{tt}, n_t, c_{ncpl}]\!]$. A number n_t is an order in $[\![e_l, s_{tt}]\!]$ in $[\![concept : e_l, s_{tt}]\!]$ if e_l is a concept in $[\![s_{tt}, n_t]\!]$.

An element e_l is a concept in $[\![c_{nf}, n_t]\!]$ if there exists $c_{ncpl.n}$ such that e_l is a concept in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$. A number n_t is an order in $[\![e_l, c_{nf}]\!]$ in $[\![concept : e_l, c_{nf}]\!]$ if e_l is a concept in $[\![c_{nf}, n_t]\!]$.

⊕ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : cm, -1 : volume, 0 : e_{l.g.2}, 1 : cube, 2 : Lobachevskian, 3 : 3)$, and $s_{tt} = (c_{ncl.1} : 3, c_{ncl.2} : 4)$. Then the following properties hold:

– *triangle*, *Euclidean*, 2 are concepts in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![2]\!]$;

– *cube*, *Lobachevskian*, 3 are concepts in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$;

– 1, 2, 3 are orders in $[\![concept : triangle]\!]$, $[\![concept : Euclidean]\!]$, $[\![concept : 2]\!]$ in $[\![s_{tt}]\!]$;

– 1, 2, 3 are orders in $[\![concept : cube]\!]$, $[\![concept : Lobachevskian]\!]$, $[\![concept : 3]\!]$ in $[\![s_{tt}]\!]$.

An element e_l is a concept in $[\![s_{tt}, c_{ncpl}]\!]$ if there exists n_t such that e_l is a concept in $[\![s_{tt}, n_t, c_{ncpl}]\!]$. A conceptual c_{ncpl} is a concretization in $[\![e_l, s_{tt}]\!]$ in $[\![concept : e_l, s_{tt}]\!]$ if e_l is a concept in $[\![s_{tt}, c_{ncpl}]\!]$.

An element e_l is a concept in $[\![c_{nf}, c_{ncpl.n}]\!]$ if there exists n_t such that e_l is a concept in

$[\![c_{nf}, n_t, c_{ncpl.n}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, c_{nf}]\!]$ in $[\![concept : e_l, c_{nf}]\!]$ if e_l is a concept in $[\![c_{nf}, c_{ncpl.n}]\!]$.

⨁ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : cm, -1 : volume, 0 : e_{l.g.2}, 1 : cube, 2 : Lobachevskian, 3 : 3)$, and $s_{tt} = (c_{ncl.1} : 3, c_{ncl.2} : 4)$. Then the following properties hold:

– *triangle*, *Euclidean*, 2 are concepts in $[\![s_{tt}, c_{ncl.1}]\!]$;

– *cube*, *Lobachevskian*, 3 are concepts in $[\![s_{tt}, c_{ncl.2}]\!]$;

– $c_{ncl.1}$ is a concretization in $[\![concept : triangle]\!]$, $[\![concept : Euclidean]\!]$, $[\![concept : 2]\!]$ in $[\![s_{tt}]\!]$;

– $c_{ncl.2}$ is a concretization in $[\![concept : cube]\!]$, $[\![concept : Lobachevskian]\!]$, $[\![concept : 3]\!]$ in $[\![s_{tt}]\!]$.

An element e_l is a concept in $[\![s_{tt}]\!]$ if there exists n_t such that e_l is a concept in $[\![s_{tt}, n_t]\!]$. A number n_t is an order in $[\![s_{tt}]\!]$ in $[\![s_{tt}, concept :]\!]$ if there exists e_l such that n_t is an order in $[\![concept : e_l, s_{tt}]\!]$. A conceptual c_{ncpl} is a concretization in $[\![s_{tt}]\!]$ in $[\![s_{tt}, concept :]\!]$ if there exists e_l such that c_{ncpl} is a concretization in $[\![concept : e_l, s_{tt}]\!]$.

An element e_l is a concept in $[\![c_{nf}]\!]$ if there exists n_t such that e_l is a concept in $[\![c_{nf}, n_t]\!]$. A number n_t is an order in $[\![c_{nf}]\!]$ in $[\![c_{nf}, concept :]\!]$ if there exists e_l such that n_t is an order in $[\![concept : e_l, c_{nf}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![c_{nf}]\!]$ in $[\![c_{nf}, concept :]\!]$ if there exists e_l such that $c_{ncpl.n}$ is a concretization in $[\![concept : e_l, c_{nf}]\!]$.

⨁ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : cm, -1 : volume, 0 : e_{l.g.2}, 1 : cube, 2 : Lobachevskian, 3 : 3)$, and $s_{tt} = (c_{ncl.1} : 3, c_{ncl.2} : 4)$. Then the following properties hold:

– *triangle*, *Euclidean*, 2, *cube*, *Lobachevskian*, 3 are concepts in $[\![s_{tt}]\!]$;

– 1, 2, 3 are orders in $[\![s_{tt}, concept :]\!]$;

– $c_{ncl.1}$, $c_{ncl.2}$ are concretizations in $[\![s_{tt}, concept :]\!]$.

6.3. Attributes

Attributes in $[\![s_{tt}]\!]$ generalize models of the usual attributes in $[\![s_{s.q.i}]\!]$ which are interpreted as characteristics of elements of $s_{s.q.i}$.

An element e_l is an attribute in $[\![s_{tt}, n_t, c_{ncpl}]\!]$ if e_l is an element in $[\![s_{tt}, -n_t, c_{ncpl}]\!]$. A number n_t is an order in $[\![e_l, s_{tt}, c_{ncpl}]\!]$ in $[\![attribute : e_l, s_{tt}, c_{ncpl}]\!]$ if e_l is an attribute in $[\![s_{tt}, n_t, c_{ncpl}]\!]$. A conceptual c_{ncpl} is a concretization in $[\![attribute : e_l, s_{tt}, n_t]\!]$ if e_l is an attribute in $[\![s_{tt}, n_t, c_{ncpl}]\!]$.

An element e_l is an attribute in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$ if e_l is an element in $[\![c_{nf}, -n_t, c_{ncpl.n}]\!]$. A number n_t is an order in $[\![e_l, c_{nf}, c_{ncpl.n}]\!]$ in $[\![attribute : e_l, c_{nf}, c_{ncpl.n}]\!]$ if e_l is an attribute in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![attribute : e_l, c_{nf}, n_t]\!]$ if e_l is an attribute in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$ and $s_{tt} = (c_{ncpl} : 3)$. Then the following properties hold:

- $area$, $inch$, 10 are attributes in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$ in $[\![c_{ncpl}]\!]$;
- 1, 2, 3 are orders in $[\![attribute : area]\!]$, $[\![attribute : inch]\!]$, $[\![attribute : 10]\!]$ in $[\![s_{tt}]\!]$ in $[\![c_{ncpl}]\!]$;
- c_{ncpl} is a concretization in $[\![attribute : area]\!]$, $[\![attribute : inch]\!]$, $[\![attribute : 10]\!]$ in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$.

An element e_l is an attribute in $[\![s_{tt}, n_t]\!]$ if there exists c_{ncpl} such that e_l is an attribute in $[\![s_{tt}, n_t, c_{ncpl}]\!]$. A number n_t is an order in $[\![e_l, s_{tt}]\!]$ in $[\![attribute : e_l, s_{tt}]\!]$ if e_l is an attribute in $[\![s_{tt}, n_t]\!]$.

An element e_l is an attribute in $[\![c_{nf}, n_t]\!]$ if there exists $c_{ncpl.n}$ such that e_l is an attribute in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$. A number n_t is an order in $[\![e_l, c_{nf}]\!]$ in $[\![attribute : e_l, c_{nf}]\!]$ if e_l is an attribute in $[\![c_{nf}, n_t]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : cm, -1 : volume, 0 : e_{l.g.2}, 1 : cube, 2 : Lobachevskian, 3 : 3)$, and $s_{tt} = (c_{ncl.1} : 3, c_{ncl.2} : 4)$. Then the following properties hold:

- $area$, $inch$, 10 are attributes in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$;
- $volume$, cm, 8 are attributes in $[\![s_{tt}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$;
- 1, 2, 3 are orders in $[\![attribute : area]\!]$, $[\![attribute : inch]\!]$, $[\![attribute : 10]\!]$ in $[\![s_{tt}]\!]$;
- 1, 2, 3 are orders in $[\![attribute : volume]\!]$, $[\![attribute : cm]\!]$, $[\![attribute : 8]\!]$ in $[\![s_{tt}]\!]$.

An element e_l is an attribute in $[\![s_{tt}, c_{ncpl}]\!]$ if there exists n_t such that e_l is an attribute in $[\![s_{tt}, n_t, c_{ncpl}]\!]$. A conceptual c_{ncpl} is a concretization in $[\![e_l, s_{tt}]\!]$ in $[\![attribute : e_l, s_{tt}]\!]$ if e_l is an attribute in $[\![s_{tt}, c_{ncpl}]\!]$.

An element e_l is an attribute in $[\![c_{nf}, c_{ncpl.n}]\!]$ if there exists n_t such that e_l is an attribute in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, c_{nf}]\!]$ in $[\![attribute : e_l, c_{nf}]\!]$ if e_l is an attribute in $[\![c_{nf}, c_{ncpl.n}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : cm, -1 : volume, 0 : e_{l.g.2}, 1 : cube, 2 : Lobachevskian, 3 : 3)$, and

$s_{tt} = (c_{ncl.1} : 3, c_{ncl.2} : 4)$. Then the following properties hold:

– *area*, *inch*, 10 are attributes in $[\![s_{tt}, c_{ncl.1}]\!]$;

– *volume*, *cm*, 8 are attributes in $[\![s_{tt}, c_{ncl.2}]\!]$;

– $c_{ncl.1}$ is a concretization in $[\![attribute : area]\!]$, $[\![attribute : inch]\!]$, $[\![attribute : 10]\!]$ in $[\![s_{tt}]\!]$;

– $c_{ncl.2}$ is a concretization in $[\![attribute : volume]\!]$, $[\![attribute : cm]\!]$, $[\![attribute : 8]\!]$ in $[\![s_{tt}]\!]$.

An element e_l is an attribute in $[\![s_{tt}]\!]$ if there exists n_t such that e_l is an attribute in $[\![s_{tt}, n_t]\!]$. A number n_t is an order in $[\![s_{tt}, attribute :]\!]$ if there exists e_l such that n_t is an order in $[\![attribute : e_l, s_{tt}]\!]$. A conceptual c_{ncpl} is a concretization in $[\![s_{tt}, attribute :]\!]$ if there exists e_l such that c_{ncpl} is a concretization in $[\![attribute : e_l, s_{tt}]\!]$.

An element e_l is an attribute in $[\![c_{nf}]\!]$ if there exists n_t such that e_l is an attribute in $[\![c_{nf}, n_t]\!]$. A number n_t is an order in $[\![c_{nf}, attribute :]\!]$ if there exists e_l such that n_t is an order in $[\![attribute : e_l, c_{nf}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![c_{nf}, attribute :]\!]$ if there exists e_l such that $c_{ncpl.n}$ is a concretization in $[\![attribute : e_l, c_{nf}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 8, -2 : cm, -1 : volume, 0 : e_{l.g.2}, 1 : cube, 2 : Lobachevskian, 3 : 3)$, and $s_{tt} = (c_{ncl.1} : 3, c_{ncl.2} : 4)$. Then the following properties hold:

– *area*, *inch*, 10, *volume*, *cm*, 8 are attributes in $[\![s_{tt}]\!]$;

– 1, 2, 3 are orders in $[\![s_{tt}, attribute :]\!]$;

– $c_{ncl.1}$, $c_{ncl.2}$ are concretizations in $[\![s_{tt}, attribute :]\!]$.

Concepts and attributes are considered in detail below.

7. Structure of concepts

7.1. Direct concepts

The usual concepts in $[\![s_{s.q.i}]\!]$ which are interpreted as sets of elements in $[\![s_{s.q.i}]\!]$ are modelled by the special kind of concepts in $[\![s_{tt}]\!]$, direct concepts in $[\![s_{tt}]\!]$.

7.1.1. Direct concepts

An element e_l is a direct concept in $[\![s_{tt}, c_{ncpl}]\!]$ if e_l is a concept in $[\![s_{tt}, 1, c_{ncpl}]\!]$. An element e_l is a direct concept in $[\![c_{nf}, c_{ncpl.n}]\!]$ if e_l is a concept in $[\![c_{nf}, 1, c_{ncpl.n}]\!]$.

An element e_l is a direct concept in $[\![s_{tt}]\!]$ if there exists c_{ncpl} such that e_l is a direct concept in $[\![s_{tt}, c_{ncpl}]\!]$. An element e_l is a direct concept in $[\![c_{nf}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is a direct concept in $[\![c_{nf}, c_{ncpl.n}]\!]$.

7.1.2. Concretizations

A conceptual c_{ncpl} is a concretization in $[\![direct{-}concept : e_l, s_{tt}]\!]$ if e_l is a concept in $[\![s_{tt}, 1, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![direct{-}concept : e_l, c_{nf}]\!]$ if e_l is a concept in $[\![c_{nf}, 1, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![s_{tt}, direct{-}concept :]\!]$ if there exists e_l such that c_{ncpl} is a concretization in $[\![direct{-}concept : e_l, s_{tt}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![c_{nf}, direct{-}concept :]\!]$ if there exists e_l such that $c_{ncpl.n}$ is a concretization in $[\![direct{-}concept : e_l, c_{nf}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : perimeter, 0 : f_g, 1 : rectangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

- $triangle$ and $rectangle$ are direct concepts in s_{tt};
- $c_{ncl.1}$ is a concretization in $[\![direct{-}concept : triangle, s_{tt}]\!]$;
- $c_{ncl.2}$ is a concretization in $[\![direct{-}concept : rectangle, s_{tt}]\!]$.

7.2. Elements of concepts

7.2.1. Elements, orders, concretizations

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if c_{ncp} is a concept in $[\![s_{tt}, n_t, c_{ncpl}]\!]$, e_l is an element in $[\![c_{ncpl}, i_{nt}]\!]$, and $i_{nt} < n_t$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if c_{ncp} is a concept in $[\![c_{nf}, n_t, c_{ncpl.n}]\!]$, e_l is an element in $[\![c_{ncpl.n}, i_{nt}]\!]$, and $i_{nt} < n_t$.

Thus, elements of c_{ncp} can be concepts of orders which are less than the order of c_{ncp}, individuals and attributes of any orders.

A number n_t is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. It specifies the order in $[\![c_{ncpl}, c_{ncp}]\!]$. A number n_t is an order in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A number i_{nt} is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. It specifies the order in $[\![c_{ncpl}, e_l]\!]$. A number i_{nt} is an order in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. It defines that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

7.2.2. Kinds of elements

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$ if there exists n_t such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if there exists n_t such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t]\!]$ if there exist i_{nt} and c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t]\!]$ if there exist i_{nt} and $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}]\!]$ if there exist n_t and

c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}]\!]$ if there exist n_t and $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, c_{ncpl}]\!]$ if there exist n_t and i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, c_{ncpl.n}]\!]$ if there exist n_t and i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}]\!]$ if there exist n_t, i_{nt}, and c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}]\!]$ if there exist n_t, i_{nt}, and $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

7.2.3. Kinds of orders in the context of concepts

A number n_t is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order :, c_{ncpl}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}]\!]$. A number n_t is an order in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order :, c_{ncpl.n}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}]\!]$.

A number n_t is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$. A number n_t is an order in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$.

A number n_t is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t]\!]$. A number n_t is an order in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t]\!]$.

7.2.4. Kinds of orders in the context of elements

A number i_{nt} is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order :, c_{ncpl}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$. A number i_{nt} is an order in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order :, c_{ncpl.n}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A number i_{nt} is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$. A number i_{nt} is an order in

$[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$.

A number i_{nt} is an order in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}]\!]$. A number i_{nt} is an order in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}]\!]$.

7.2.5. Kinds of concretizations

A conceptual c_{ncpl} is a concretization in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![e_l, concept : c_{ncp}, s_{tt}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, concept : c_{ncp}, c_{nf}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, c_{ncpl.n}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 2, -2 : cm, -1 : perimeter, 0 : e_{l.g.2}, 1 : rectangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

– 10, *inch*, *area*, $e_{l.g.1}$ are elements in $[\![concept : triangle, s_{tt}]\!]$;

– 2, *cm*, *perimeter*, $e_{l.g.2}$ are elements in $[\![concept : rectangle, s_{tt}]\!]$;

– 10, *inch*, *area*, $e_{l.g.1}$, 2, *cm*, *perimeter*, $e_{l.g.2}$, *triangle*, *rectangle* are elements in $[\![concept : Eucludian, s_{tt}]\!]$;

– 10, *inch*, *area*, $e_{l.g.1}$, 2, *cm*, *perimeter*, $e_{l.g.2}$, *triangle*, *rectangle*, *Eucludian* are elements in $[\![concept : 2, s_{tt}]\!]$;

– $c_{ncl.1}$ is a concretization in $[\![concept : triangle]\!]$, $[\![concept : Eucludian]\!]$, $[\![concept : 2]\!]$ in $[\![s_{tt}]\!]$;

– $c_{ncl.2}$ is a concretization in $[\![concept : rectangle]\!]$, $[\![concept : Eucludian]\!]$, $[\![concept : 2]\!]$ in $[\![s_{tt}]\!]$;

– 1 is an order in $[\![e_{l.g.2}, concept : rectangle, s_{tt}, concept{-}order :]\!]$;

– 0 is an order in $[\![e_{l.g.1}, concept : triangle, s_{tt}, element{-}order :]\!]$;

– -1 is an order in $[\![area, concept : triangle, s_{tt}, element{-}order :]\!]$;

– -2 is an order in $[\![cm, concept : Eucludian, s_{tt}, element{-}order :]\!]$.

7.3. The property of direct concepts

Proposition 6. If c_{ncp} is a concept in $[\![s_{tt}]\!]$ and e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : 1]\!]$, then e_l is either an individual in $[\![s_{tt}]\!]$, or e_l is an attribute in $[\![s_{tt}]\!]$.

Proof. This follows from the definition of direct concepts. □

7.4. The content of concepts

The content of a concept describes its semantics.

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if s_t is the set of all elements in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if s_t is the set of all elements in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl}[\![s_{tt}]\!]} s_t[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl.n}[\![c_{nf}]\!]} s_t$ $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t]\!]$ if $s_t = \bigcup_{i_{nt}<n_t} s_t[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t]\!]$ if $s_t = \bigcup_{i_{nt}<n_t} s_t[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$.

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 2, -2 : cm, -1 : perimeter, 0 : e_{l.g.2}, 1 : rectangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

– $\{10, inch, area, e_{l.g.1}\}$ is the content in $[\![concept : triangle, s_{tt}]\!]$;

– $\{2, cm, perimeter, e_{l.g.2}\}$ is the content in $[\![concept : rectangle, s_{tt}]\!]$;

– $\{10, inch, area, e_{l.g.1}, 2, cm, perimeter, e_{l.g.2}, triangle, rectangle\}$ is the content in $[\![concept : Eucludian, s_{tt}]\!]$;

– $\{10, inch, area, e_{l.g.1}, 2, cm, perimeter, e_{l.g.2}, triangle, rectangle, Eucludian\}$ is the content in $[\![concept : 2, s_{tt}]\!]$.

7.5. Mediators

7.5.1. Mediators, elements, degrees

An element $e_{l.1}$ is a mediator in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$, $e_{l.1}$ is an element in $[\![c_{ncpl}, i_{nt.1}]\!]$, and $i_{nt} < i_{nt.1} < n_t$. It is between e_l and c_{ncp} in c_{ncpl} in the position $i_{nt.1}$, thus separating e_l from c_{ncp} in c_{ncpl}. An element $e_{l.1}$ is a mediator in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$, $e_{l.1}$ is an element in $[\![c_{ncpl.n}, i_{nt.1}]\!]$, and $i_{nt} < i_{nt.1} < n_t$.

An element $e_{l.1}$ is a mediator in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if there exists $i_{nt.1}$ such that $e_{l.1}$ is a mediator in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element $e_{l.1}$ is a mediator in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if there exists $i_{nt.1}$ such that $e_{l.1}$ is a mediator in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ and $n_{at.1}$ is the number of orders $i_{nt.1}$ in $[\![c_{ncpl}, \hat{e}_l]\!]$ such that $i_{nt} < i_{nt.1} < n_t$. It is separated from c_{ncp} in c_{ncpl} by $n_{at.1}$ of mediators. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ and $n_{at.1}$ is the number of orders $i_{nt.1}$ in $[\![c_{ncpl.n}, \hat{e}_l]\!]$ such that $i_{nt} < i_{nt.1} < n_t$.

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. It specifies how many mediators separate e_l from c_{ncp} in c_{ncpl}. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

7.5.2. Kinds of elements

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exists c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if there exists n_t such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if there exists n_t such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if there exist i_{nt} and c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if there exist i_{nt} and $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t and c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t and $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if there exist

n_t and i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t and i_{nt} such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![concept : c_{ncp}, s_{tt}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t, i_{nt}, and c_{ncpl} such that e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![concept : c_{ncp}, c_{nf}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t, i_{nt}, and $c_{ncpl.n}$ such that e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

7.5.3. Kinds of degrees

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, concept{-}order : n_t, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, concept{-}order : n_t, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, s_{tt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, concept : c_{ncp}, c_{nf}, mediator{-}degree :]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, mediator{-}degree : n_{at.1}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then f_g is an element in the following contexts:

– $[\![concept : triangle, s_{tt}]\!]$ with the decree 0 and without mediators;

– $[\![concept : Euclidean, s_{tt}]\!]$ with the decree 1 and the mediator *triangle*;

– $[\![concept : 2, s_{tt}]\!]$ with the decree 2 and the mediators *triangle* and *Euclidean*;

– $[\![concept : Euclidean, s_{tt}]\!]$ with the decree 0 and without mediators;

– $[\![concept : 2, s_{tt}]\!]$ with the decree 1 and the mediator *Euclidean*.

7.6. Direct elements

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : 0]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if e_l is an element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : 0]\!]$.

7.6.1. Kinds of direct elements

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists c_{ncpl} such that e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists c_{ncpl} such that e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, c_{ncpl}]\!]$ if there exists i_{nt} such that e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order :$

$i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, c_{ncpl.n}]\!]$ if there exists i_{nt} such that e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$ if there exists n_t such that e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if there exists n_t such that e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t]\!]$ if there exist i_{nt} and c_{ncpl} such that e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t]\!]$ if there exist i_{nt} and $c_{ncpl.n}$ such that e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, element{-}order : i_{nt}]\!]$ if there exist n_t and c_{ncpl} such that e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, element{-}order : i_{nt}]\!]$ if there exist n_t and $c_{ncpl.n}$ such that e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, c_{ncpl}]\!]$ if there exist n_t and i_{nt} such that e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, c_{ncpl.n}]\!]$ if there exist n_t and i_{nt} such that e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}]\!]$ if there exist n_t, i_{nt}, and c_{ncpl} such that e_l is a direct element in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}]\!]$ if there exist n_t, i_{nt}, and $c_{ncpl.n}$ such that e_l is a direct element in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$ and $s_{tt} = (c_{ncpl} : 3)$. Then the following properties hold:

- f_g is a direct element in $[\![concept : triangle, s_{tt}]\!]$ that means that f_g is a triangle in $[\![s_{tt}]\!]$;
- $triangle$ is a direct element in $[\![concept : Eucludian, s_{tt}]\!]$ that means that classification of geometric figures in Eucludian space includes triangles in $[\![s_{tt}]\!]$;

– *Eucludian* is a direct element in $[\![concept : 2, s_{tt}]\!]$ that means that classification of two-dimensional spaces includes Eucludian space in $[\![s_{tt}]\!]$.

7.7. The direct content of concepts

A set s_t is the direct content in $[\![concept : c_{ncp}, s_{tt}, concept\!-\!order : n_t, element\!-\!order : i_{nt}, c_{ncpl}]\!]$ if s_t is the set of all direct elements in $[\![concept : c_{ncp}, s_{tt}, concept\!-\!order : n_t, element\!-\!order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the direct content in $[\![concept : c_{ncp}, c_{nf}, concept\!-\!order : n_t, element\!-\!order : i_{nt}, c_{ncpl.n}]\!]$ if s_t is the set of all direct elements in $[\![concept : c_{ncp}, c_{nf}, concept\!-\!order : n_t, element\!-\!order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the direct content in $[\![concept : c_{ncp}, s_{tt}, concept\!-\!order : n_t, element\!-\!order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl}[\![s_{tt}]\!]} s_t[\![concept : c_{ncp}, s_{tt}, concept\!-\!order : n_t, element\!-\!order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the direct content in $[\![concept : c_{ncp}, c_{nf}, concept\!-\!order : n_t, element\!-\!order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl.n}[\![c_{nf}]\!]} s_t[\![concept : c_{ncp}, c_{nf}, concept\!-\!order : n_t, element\!-\!order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the direct content in $[\![concept : c_{ncp}, s_{tt}, concept\!-\!order : n_t]\!]$ if $s_t = \bigcup_{i_{nt}<n_t} s_t[\![concept : c_{ncp}, s_{tt}, concept\!-\!order : n_t, element\!-\!order : i_{nt}]\!]$. A set s_t is the direct content in $[\![concept : c_{ncp}, c_{nf}, concept\!-\!order : n_t]\!]$ if $s_t = \bigcup_{i_{nt}<n_t} s_t[\![concept : c_{ncp}, c_{nf}, concept\!-\!order : n_t, element\!-\!order : i_{nt}]\!]$.

A set s_t is the direct content in $[\![concept : c_{ncp}, s_{tt}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![concept : c_{ncp}, s_{tt}, concept\!-\!order : n_t]\!]$. A set s_t is the direct content in $[\![concept : c_{ncp}, c_{nf}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![concept : c_{ncp}, c_{nf}, concept\!-\!order : n_t]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.2}, 1 : triangle, 2 : Riemannian, 3 : 2)$, $c_{ncl.3} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}, c_{ncl.3}\}$. Then the following properties hold:

– $\{e_{l.g.1}, e_{l.g.2}\}$ is the direct content in $[\![concept : triangle, s_{tt}]\!]$;

– $\{triangle\}$ is the direct content in $[\![concept : Eucludian, s_{tt}]\!]$;

– $\{triangle\}$ is the direct content in $[\![concept : Riemannian, s_{tt}]\!]$;

– $\{Eucludian, Riemannian\}$ is the direct content in $[\![concept : 2, s_{tt}]\!]$;

– $\{e_{l.g.1}\}$ is the direct content in $[\![concept : 2, s_{tt}]\!]$.

7.8. The content of concepts in the context of mediators

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if s_t is the set of all elements in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if s_t is the set of all elements in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{c_{ncpl}[\![s_{tt}]\!]} s_t[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{c_{ncpl.n}[\![c_{nf}]\!]} s_t [\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{i_{nt}<n_t} s_t[\![concept : c_{ncp}, s_{tt}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{i_{nt}<n_t} s_t[\![concept : c_{ncp}, c_{nf}, concept{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

A set s_t is the content in $[\![concept : c_{ncp}, s_{tt}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![concept : c_{ncp}, s_{tt}, concept{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![concept : c_{ncp}, c_{nf}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![concept : c_{ncp}, c_{nf}, concept{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.2}, 1 : triangle, 2 : Riemannian, 3 : 2)$, $c_{ncl.3} = (-3 : 10, -2 : inch, -1 : perimeter, 0 : e_{l.g.3}, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}, c_{ncl.3}\}$. Then the following properties hold:

– $\{e_{l.g.1}, e_{l.g.2}\}$ is the content in $[\![concept : 2, s_{tt}, mediator{-}degree : 2]\!]$;

– $\{e_{l.g.3}\}$ is the content in $[\![concept : 2, s_{tt}, mediator{-}degree : 1]\!]$;

– $\{area\}$ is the content in $[\![concept : 2, s_{tt}, mediator{-}degree : 3]\!]$;

– $\{perimeter\}$ is the content in $[\![concept : 2, s_{tt}, mediator{-}degree : 2]\!]$.

8. Classification and interpretation of concepts

Concepts are classified according to their orders.

8.1. Concepts of the order 1

A concept c_{ncp} in $[\![s_{tt}, 1]\!]$ models a usual concept in $[\![s_{s.q.i}]\!]$. Elements in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : 1$ are attributes and individuals in $[\![s_{tt}]\!]]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 2, -2 : cm, -1 : perimeter, 0 : e_{l.g.2}, 1 : triangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

- the direct concept *triangle* models triangles in $[\![s_{tt}]\!]$;
- the individuals $e_{l.g.1}$ and $e_{l.g.2}$ are elements of the order 0 of the direct concept *triangle* in $[\![s_{tt}]\!]$ that means that $e_{l.g.1}$ and $e_{l.g.2}$ are triangles in $[\![s_{tt}]\!]$;
- the attributes *area* and *perimeter* are elements of the order -1 of the direct concept *triangle* in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of triangles includes area and perimeter in $[\![s_{tt}]\!]$;
- the attributes *inch* and *cm* are elements of the order -2 of the direct concept *triangle* in $[\![s_{tt}]\!]$ that means that classification of units of measurement of numerical characteristics of triangles includes inches and centimetres in $[\![s_{tt}]\!]$;
- the attributes 10 and 2 are elements of the order -3 of the direct concept *triangle* in $[\![s_{tt}]\!]$ that means that classification of numeral systems for representing values of numerical characteristics of triangles includes decimal and binary systems in $[\![s_{tt}]\!]$.

8.2. Concepts of the order 2

A concept c_{ncp} in $[\![s_{tt}, 2]\!]$ models a concept space in $[\![s_{s.q.i}]\!]$. Elements in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : 2$ are attributes, individuals and direct concepts in $[\![s_{tt}]\!]]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 2, -2 : cm, -1 : perimeter, 0 : e_{l.g.2}, 1 : square, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

- the concept space *Euclidean* models Euclidean space in $[\![s_{tt}]\!]$;
- the direct concepts *triangle* and *square* are elements of the order 1 of the concept space *Euclidean* in $[\![s_{tt}]\!]$ that means that classification of geometric figures in Euclidean space includes triangles and squares in $[\![s_{tt}]\!]$;
- the individuals $e_{l.g.1}$ and $e_{l.g.2}$ are elements of the order 0 of the concept space *Euclidean* in $[\![s_{tt}]\!]$ that means that $e_{l.g.1}$ and $e_{l.g.2}$ are geometric figures in Euclidean space in $[\![s_{tt}]\!]$;
- the attributes *area* and *perimeter* are elements of the order -1 of the concept space

Euclidean in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of geometric figures in Euclidean space includes area and perimeter in $[\![s_{tt}]\!]$;

– the attributes *inch* and *cm* are elements of the order -2 of the concept space *Euclidean* in $[\![s_{tt}]\!]$ that means that classification of units of measurement of numerical characteristics of geometric figures in Euclidean space includes inches and centimetres in $[\![s_{tt}]\!]$;

– the attributes 10 and 2 are elements of the order -3 of the concept space *Euclidean* in $[\![s_{tt}]\!]$ that means that classification of numeral systems for representing values of numerical characteristics of geometric figures in Euclidean space includes decimal and binary systems in $[\![s_{tt}]\!]$.

8.3. Concepts of the order 3

A concept c_{ncp} in $[\![s_{tt}, 3]\!]$ models a space of concept spaces in $[\![s_{s.q.i}]\!]$. Elements in $[\![concept : c_{ncp}, s_{tt}, concept{-}order : 3]\!]$ are attributes, individuals, direct concepts and concept spaces in $[\![s_{tt}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 2, -2 : cm, -1 : perimeter, 0 : e_{l.g.2}, 1 : square, 2 : Riemannian, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

– the concept space space 2 models two-dimensional space in $[\![s_{tt}]\!]$;

– the concept spaces *Euclidean* and *Riemannian* are elements of the order 2 of the concept space space 2 in $[\![s_{tt}]\!]$ that means that classification of two-dimensional spaces includes Euclidean space and Riemannian space in $[\![s_{tt}]\!]$;

– the direct concepts *triangle* and *square* are elements of the order 1 of the concept space space 2 in $[\![s_{tt}]\!]$ that means that classification of geometric figures in two-dimensional space includes triangles and squares in $[\![s_{tt}]\!]$;

– the individuals $e_{l.g.1}$ and $e_{l.g.2}$ are elements of the order 0 of the concept space space 2 in $[\![s_{tt}]\!]$ that means that $e_{l.g.1}$ and $e_{l.g.2}$ are geometric figures in two-dimensional space in $[\![s_{tt}]\!]$;

– the attributes *area* and *perimeter* are elements of the order -1 of the concept space space 2 in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of geometric figures in two-dimensional space includes area and perimeter in $[\![s_{tt}]\!]$;

– the attributes *inch* and *cm* are elements of the order -2 of the concept space space 2 in $[\![s_{tt}]\!]$ that means that classification of units of measurement of numerical characteristics of geometric figures in two-dimensional space includes inches and centimetres in $[\![s_{tt}]\!]$;

– the attributes 10 and 2 are elements of the order -3 of the concept space space 2 in $[\![s_{tt}]\!]$ that means that classification of numeral systems for representing values of numerical characteristics of geometric figures in two-dimensional space includes decimal and binary systems in $[\![s_{tt}]\!]$.

8.4. Concepts of higher orders

A concept c_{ncp} in $[\![s_{tt}, n_t]\!]$, where $n_t > 3$, is classified and interpreted in the similar way (by the introduction of the space of concept space spaces and so on.).

9. Structure of attributes

Attributes use the same terminology as concepts.

9.1. Direct attributes

The usual attributes in $[\![s_{s.q.i}]\!]$ which are interpreted as characteristics of elements in $[\![s_{s.q.i}]\!]$ are modelled by the special kind of attributes in $[\![s_{tt}]\!]$, direct attributes in $[\![s_{tt}]\!]$.

9.1.1. Direct concepts

An element e_l is a direct attribute in $[\![s_{tt}, c_{ncpl}]\!]$ if e_l is a attribute in $[\![s_{tt}, 1, c_{ncpl}]\!]$. An element e_l is a direct attribute in $[\![c_{nf}, c_{ncpl.n}]\!]$ if e_l is a attribute in $[\![c_{nf}, 1, c_{ncpl.n}]\!]$.

An element e_l is a direct attribute in $[\![s_{tt}]\!]$ if there exists c_{ncpl} such that e_l is a direct attribute in $[\![s_{tt}, c_{ncpl}]\!]$. An element e_l is a direct attribute in $[\![c_{nf}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is a direct attribute in $[\![c_{nf}, c_{ncpl.n}]\!]$.

9.1.2. Concretizations

A conceptual c_{ncpl} is a concretization in $[\![direct\text{−}attribute : e_l, s_{tt}]\!]$ if e_l is a attribute in $[\![s_{tt}, 1, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![direct\text{−}attribute : e_l, c_{nf}]\!]$ if e_l is a attribute in $[\![c_{nf}, 1, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![s_{tt}, direct\text{−}attribute :]\!]$ if there exists e_l such that c_{ncpl} is a concretization in $[\![direct\text{−}attribute : e_l, s_{tt}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![c_{nf}, direct\text{−}attribute :]\!]$ if there exists e_l such that $c_{ncpl.n}$ is a concretization in $[\![direct\text{−}$

$attribute : e_l, c_{nf}\rrbracket$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : perimeter, 0 : f_g, 1 : rectangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

– *area* and *perimeter* are direct attributes in s_{tt};

– $c_{ncl.1}$ is a concretization in $\llbracket direct{-}attribute : area, s_{tt}\rrbracket$;

– $c_{ncl.2}$ is a concretization in $\llbracket direct{-}attribute : perimeter, s_{tt}\rrbracket$.

9.2. Elements of attributes

9.2.1. Elements, orders, concretizations

An element e_l is an element in $\llbracket attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}\rrbracket$ if a_{tt} is an attibute in $\llbracket s_{tt}, n_t, c_{ncpl}\rrbracket$, e_l is an element in $\llbracket c_{ncpl}, i_{nt}\rrbracket$, and $-n_t < i_{nt}$. An element e_l is an element in $\llbracket attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}\rrbracket$ if a_{tt} is an attibute in $\llbracket c_{nf}, n_t, c_{ncpl.n}\rrbracket$, e_l is an element in $\llbracket c_{ncpl.n}, i_{nt}\rrbracket$, and $-n_t < i_{nt}$.

Thus, elements of the attribute a_{tt} can be attributes of orders which are less than the order of a_{tt}, individuals and concepts of all orders.

A number n_t is an order in $\llbracket e_l, attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}\rrbracket$ if e_l is an element in $\llbracket attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}\rrbracket$. It specifies the order in $\llbracket c_{ncpl}, a_{tt}\rrbracket$. A number n_t is an order in $\llbracket e_l, attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}\rrbracket$ if e_l is an element in $\llbracket attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}\rrbracket$.

A number i_{nt} is an order in $\llbracket e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}\rrbracket$ if e_l is an element in $\llbracket attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}\rrbracket$. It specifies the order in $\llbracket c_{ncpl}, e_l\rrbracket$. A number i_{nt} is an order in $\llbracket e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}\rrbracket$ if e_l is an element in $\llbracket attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}\rrbracket$.

A conceptual c_{ncpl} is a concretization in $\llbracket e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}\rrbracket$ if e_l is an element in $\llbracket attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}\rrbracket$. It defines that e_l is an element in $\llbracket attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}\rrbracket$. A conceptual $c_{ncpl.n}$ is a concretization in $\llbracket e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}\rrbracket$ if e_l is an element in $\llbracket attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}\rrbracket$.

9.2.2. Kinds of elements

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : i_{nt}, c_{ncpl}]\!]$ if there exists n_t such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : i_{nt}, c_{ncpl.n}]\!]$ if there exists n_t such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$ if there exist i_{nt} and c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$ if there exist i_{nt} and $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}]\!]$ if there exist n_t and c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}]\!]$ if there exist n_t and $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, c_{ncpl}]\!]$ if there exist n_t and i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, c_{ncpl.n}]\!]$ if there exist n_t and i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}]\!]$ if there exist n_t, i_{nt}, and c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}]\!]$ if there exist n_t, i_{nt}, and $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

9.2.3. Kinds of orders in the context of attributes

A number n_t is an order in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order :, c_{ncpl}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}]\!]$. A number n_t is an order in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order :, c_{ncpl.n}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}]\!]$.

A number n_t is an order in $[\![e_l, attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$. A number n_t is an order in $[\![e_l, attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$.

A number n_t is an order in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$. A number n_t is an order in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$.

9.2.4. Kinds of orders in the context of elements

A number i_{nt} is an order in $[\![e_l, attribute : a_{tt}, s_{tt}, element{-}order :, c_{ncpl}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$. A number i_{nt} is an order in $[\![e_l, attribute : a_{tt}, c_{nf}, element{-}order :, c_{ncpl.n}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A number i_{nt} is an order in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$. A number i_{nt} is an order in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$.

A number i_{nt} is an order in $[\![e_l, attribute : a_{tt}, s_{tt}, element{-}order :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}]\!]$. A number i_{nt} is an order in $[\![e_l, attribute : a_{tt}, c_{nf}, element{-}order :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}]\!]$.

9.2.5. Kinds of concretizations

A conceptual c_{ncpl} is a concretization in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![e_l, attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A conceptual c_{ncpl} is a concretization in $[\![e_l, attribute : a_{tt}, s_{tt}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, c_{ncpl}]\!]$. A conceptual $c_{ncpl.n}$ is a concretization in $[\![e_l, attribute : a_{tt}, c_{nf}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, c_{ncpl.n}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : volume, 0 : e_{l.g.2}, 1 : pyramid, 2 : Riemannian, 3 : 3)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

- 2, *Euclidean*, *triangle*, $e_{l.g.1}$ are elements in $[\![attribute : area, s_{tt}]\!]$;
- 3, *Riemannian*, *pyramid*, $e_{l.g.2}$ are elements in $[\![attribute : volume, s_{tt}]\!]$;
- 2, *Euclidean*, *triangle*, $e_{l.g.1}$, 3, *Riemannian*, *pyramid*, $e_{l.g.2}$, *area*, *volume* are elements in $[\![attribute : inch, s_{tt}]\!]$;
- 2, *Euclidean*, *triangle*, $e_{l.g.1}$, 3, *Riemannian*, *pyramid*, $e_{l.g.2}$, *area*, *volume*, *inch* are elements in $[\![attribute : 10, s_{tt}]\!]$;
- $c_{ncl.1}$ is a concretization in $[\![attribute : area]\!]$, $[\![attribute : inch]\!]$, $[\![attribute : 10]\!]$ in $[\![s_{tt}]\!]$;
- $c_{ncl.2}$ is a concretization in $[\![attribute : volume]\!]$, $[\![attribute : inch]\!]$, $[\![attribute : 10]\!]$ in $[\![s_{tt}]\!]$;
- 1 is an order in $[\![e_{l.g.2}, attribute : volume, s_{tt}, attribute{-}order :]\!]$;
- 0 is an order in $[\![e_{l.g.1}, attribute : area, s_{tt}, element{-}order :]\!]$;
- 1 is an order in $[\![triangle, attribute : area, s_{tt}, element{-}order :]\!]$;
- 2 is an order in $[\![Eucludian, attribute : inch, s_{tt}, element{-}order :]\!]$.

9.3. The property of direct attributes

Proposition 7. If a_{tt} is an attribute in $[\![s_{tt}]\!]$ and e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : 1]\!]$, then e_l is either an individual or e_l is a concept in $[\![s_{tt}]\!]$.

Proof. This follows from the definition of direct attributes. □

9.4. The content of attributes

The content of a attributes describes its semantics.

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if s_t is the set of all elements in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if s_t is the set of all elements in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl}[\![s_{tt}]\!]} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl.n}[\![c_{nf}]\!]} s_t$ $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$ if $s_t = \bigcup_{-n_t<i_{nt}} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$ if $s_t = \bigcup_{-n_t<i_{nt}} s_t[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$.

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : volume, 0 : e_{l.g.2}, 1 : pyramid, 2 : Riemannian, 3 : 3)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

- $\{2, Euclidean, triangle, e_{l.g.1}\}$ is the content in $[\![attribute : area, s_{tt}]\!]$;
- $\{3, Riemannian, pyramid, e_{l.g.2}\}$ is the content in $[\![attribute : volume, s_{tt}]\!]$;
- $\{2, Euclidean, triangle, e_{l.g.1}, 3, Riemannian, pyramid, e_{l.g.2}, area, volume\}$ is the content in $[\![attribute : inch, s_{tt}]\!]$;
- $\{2, Euclidean, triangle, e_{l.g.1}, 3, Riemannian, pyramid, e_{l.g.2}, area, volume, inch\}$ is the content in $[\![concept : 10, s_{tt}]\!]$.

9.5. Mediators

9.5.1. Mediators, elements, degrees

An element $e_{l.1}$ is a mediator in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, i_{nt.1}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$, $e_{l.1}$ is an element in $[\![c_{ncpl}, i_{nt.1}]\!]$, and $-n_t < i_{nt.1} < i_{nt}$. It is between a_{tt} and e_l in c_{ncpl} in the position $i_{nt.1}$, thus separating e_l from a_{tt} in c_{ncpl}. An element $e_{l.1}$ is a mediator in

$[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, i_{nt.1}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$, $e_{l.1}$ is an element in $[\![c_{ncpl.n}, i_{nt.1}]\!]$, and $-n_t < i_{nt.1} < i_{nt}$.

An element $e_{l.1}$ is a mediator in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if there exists $i_{nt.1}$ such that $e_{l.1}$ is a mediator in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, i_{nt.1}]\!]$. An element $e_{l.1}$ is a mediator in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if there exists $i_{nt.1}$ such that $e_{l.1}$ is a mediator in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, i_{nt.1}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ and $n_{at.1}$ is the number of orders $i_{nt.1}$ in $[\![c_{ncpl}, \hat{e}_l]\!]$ such that $-n_t < i_{nt.1} < i_{nt}$. It is separated from a_{tt} in c_{ncpl} by $n_{at.1}$ of mediators. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ and $n_{at.1}$ is the number of orders $i_{nt.1}$ in $[\![c_{ncpl.n}, \hat{e}_l]\!]$ such that $-n_t < i_{nt.1} < i_{nt}$.

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. It specifies how many mediators separate e_l from a_{tt} in c_{ncpl}. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

9.5.2. Kinds of elements

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exists c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if there exists i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if there exists n_t such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if there exists n_t such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if there exist i_{nt} and c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if there exist i_{nt} and $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t and c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t and $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t and i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t and i_{nt} such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

An element e_l is an element in $[\![attribute : a_{tt}, s_{tt}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t, i_{nt}, and c_{ncpl} such that e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. An element e_l is an element in $[\![attribute : a_{tt}, c_{nf}, mediator{-}degree : n_{at.1}]\!]$ if there exist n_t, i_{nt}, and $c_{ncpl.n}$ such that e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

9.5.3. Kinds of degrees

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, s_{tt}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, mediator{-}degree : n_{at.1}]\!]$. A number $n_{at.1}$ is a degree in $[\![e_l, attribute : a_{tt}, c_{nf}, mediator{-}degree :]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, mediator{-}degree : n_{at.1}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : cm, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] =$

$\{c_{ncl.1}, c_{ncl.2}\}$. Then f_g is an element in the following contexts:

– $[\![attribute : area, s_{tt}]\!]$ with the decree 0 and without mediators;

– $[\![attribute : inch, s_{tt}]\!]$ with the decree 1 and the mediator $area$;

– $[\![attribute : 10, s_{tt}]\!]$ with the decree 2 and the mediators $area$ and $inch$;

– $[\![attribute : cm, s_{tt}]\!]$ with the decree 0 and without mediators;

– $[\![attribute : 10, s_{tt}]\!]$ with the decree 1 and the mediator cm.

9.6. Direct elements

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : 0]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if e_l is an element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : 0]\!]$.

9.6.1. Kinds of direct elements

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists c_{ncpl} such that e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if there exists $c_{ncpl.n}$ such that e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, c_{ncpl}]\!]$ if there exists i_{nt} such that e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, c_{ncpl.n}]\!]$ if there exists i_{nt} such that e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}, c_{ncpl}]\!]$ if there exists n_t such that e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if there exists n_t such that e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$ if there exist i_{nt} and c_{ncpl} such that e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf},$

$attribute{-}order : n_t]\!]$ if there exist i_{nt} and $c_{ncpl.n}$ such that e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, element{-}order : i_{nt}]\!]$ if there exist n_t and c_{ncpl} such that e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, element{-}order : i_{nt}]\!]$ if there exist n_t and $c_{ncpl.n}$ such that e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, c_{ncpl}]\!]$ if there exist n_t and i_{nt} such that e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, c_{ncpl.n}]\!]$ if there exist n_t and i_{nt} such that e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

An element e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}]\!]$ if there exist n_t, i_{nt}, and c_{ncpl} such that e_l is a direct element in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. An element e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}]\!]$ if there exist n_t, i_{nt}, and $c_{ncpl.n}$ such that e_l is a direct element in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$ and $s_{tt} = (c_{ncpl} : 3)$. Then the following properties hold:

- f_g is a direct element in $[\![attribute : area, s_{tt}]\!]$ that means that classification of numerical characteristics of f_g includes area in $[\![s_{tt}]\!]$;
- $area$ is a direct element in $[\![attribute : inch, s_{tt}]\!]$ that means that classification of units of measurement of numerical characteristics of geometric figures includes inches in $[\![s_{tt}]\!]$;
- $inch$ is a direct element in $[\![attribute : 10, s_{tt}]\!]$ that means that classification of numeral systems for representing values of numerical characteristics of geometric figures includes decimal system in $[\![s_{tt}]\!]$.

9.7. The direct content of attributes

A set s_t is the direct content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$ if s_t is the set of all direct elements in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the direct content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$ if s_t is the set of all direct elements in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the direct content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl}[\![s_{tt}]\!]} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}]\!]$. A set s_t is the direct content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$ if $s_t = \bigcup_{c_{ncpl.n}[\![c_{nf}]\!]} s_t[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}]\!]$.

A set s_t is the direct content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$ if $s_t = \bigcup_{-n_t<i_{nt}} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$. A set s_t is the direct content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$ if $s_t = \bigcup_{-n_t<i_{nt}} s_t[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}]\!]$.

A set s_t is the direct content in $[\![attribute : a_{tt}, s_{tt}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t]\!]$. A set s_t is the direct content in $[\![attribute : a_{tt}, c_{nf}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : cm, -1 : area, 0 : e_{l.g.2}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.3} = (-3 : 10, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}, c_{ncl.3}\}$. Then the following properties hold:

– $\{e_{l.g.1}, e_{l.g.2}\}$ is the direct content in $[\![attribute : area, s_{tt}]\!]$;

– $\{area\}$ is the direct content in $[\![attribute : inch, s_{tt}]\!]$;

– $\{area\}$ is the direct content in $[\![attribute : cm, s_{tt}]\!]$;

– $\{inch, cm\}$ is the direct content in $[\![attribute : 10, s_{tt}]\!]$;

– $\{e_{l.g.1}\}$ is the direct content in $[\![attribute : 10, s_{tt}]\!]$.

9.8. The content of attributes in the context of mediators

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$ if s_t is the set of all elements in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$ if s_t is the set of all elements in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{c_{ncpl}[\![s_{tt}]\!]} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{c_{ncpl.n}[\![c_{nf}]\!]} s_t$

$[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, c_{ncpl.n}, mediator{-}degree : n_{at.1}]\!]$.

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{-n_t<i_{nt}} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{-n_t<i_{nt}} s_t[\![attribute : a_{tt}, c_{nf}, attribute{-}order : n_t, element{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

A set s_t is the content in $[\![attribute : a_{tt}, s_{tt}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![attribute : a_{tt}, s_{tt}, attribute{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$. A set s_t is the content in $[\![attribute : a_{tt}, c_{nf}, mediator{-}degree : n_{at.1}]\!]$ if $s_t = \bigcup_{n_t} s_t[\![attribute : a_{tt}, c_{nf}, attribute{-}order : i_{nt}, mediator{-}degree : n_{at.1}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : perimeter, 0 : e_{l.g.2}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.3} = (-3 : 10, -2 : inch, 0 : e_{l.g.3}, 1 : rectangle, 2 : Euclidean, 3 : 2)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}, c_{ncl.3}\}$. Then the following properties hold:

- $\{e_{l.g.1}, e_{l.g.2}\}$ is the content in $[\![attribute : 10, s_{tt}, mediator{-}degree : 2]\!]$;
- $\{e_{l.g.3}\}$ is the content in $[\![attribute : 10, s_{tt}, mediator{-}degree : 1]\!]$;
- $\{triangle\}$ is the content in $[\![attribute : 10, s_{tt}, mediator{-}degree : 3]\!]$;
- $\{rectangle\}$ is the content in $[\![attribute : 10, s_{tt}, mediator{-}degree : 2]\!]$.

10. Classification and interpretation of attributes

Attributes are classified according to their orders.

10.1. Attributes of the order 1

An attribute a_{tt} in $[\![s_{tt}, 1]\!]$ models a usual attribute in $[\![s_{s.q.i}]\!]$. Elements in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : 1]\!]$ are individuals and concepts in $[\![s_{tt}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.2}, 1 : square, 2 : Riemannian, 3 : 3)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

- the direct attribute *area* classifies geometric figures having area in $[\![s_{tt}]\!]$;
- the individuals $e_{l.g.1}$ and $e_{l.g.2}$ are elements of the order 0 of the direct attribute *area* in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of $e_{l.g.1}$ and $e_{l.g.2}$ includes area in $[\![s_{tt}]\!]$;

– the concepts *triangle* and *square* are elements of the order 1 of the direct attribute *area* in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of triangles and squares includes area in $[\![s_{tt}]\!]$;

– the concept spaces *Euclidean* and *Riemannian* are elements of the order 2 of the direct attribute *area* in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of geometric figures in Euclidean and Riemannian spaces includes area in $[\![s_{tt}]\!]$;

– the concept space spaces 2 and 3 are elements of the order 3 of the direct attribute *area* in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of geometric figures in two-dimensional and three-dimensional spaces includes area in $[\![s_{tt}]\!]$.

10.2. Attributes of the order 2

An attribute a_{tt} in $[\![s_{tt}, 2]\!]$ models an attribute space in $[\![s_{s.q.i}]\!]$. Elements in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : 2]\!]$ are direct attributes, individuals and concepts in $[\![s_{tt}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : inch, -1 : perimeter, 0 : e_{l.g.2}, 1 : square, 2 : Riemannian, 3 : 3)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

– the attribute space *inch* classifies numerical characteristics of geometric figures measured in inches in $[\![s_{tt}]\!]$;

– the direct attributes *area* and *perimeter* are elements of the order -1 of the attribute space *inch* in $[\![s_{tt}]\!]$ that means that classification of numerical characteristics of geometric figures measured in inches includes area and perimeter in $[\![s_{tt}]\!]$;

– the individuals $e_{l.g.1}$ and $e_{l.g.2}$ are elements of the order 0 of the attribute space *inch* in $[\![s_{tt}]\!]$ that means that classifications of geometric figures with numerical characteristics measured in inches includes $e_{l.g.1}$ and $e_{l.g.2}$ in $[\![s_{tt}]\!]$;

– the concepts *triangle* and *square* are elements of the order 1 of the attribute space *inch* in $[\![s_{tt}]\!]$ that means that classifications of geometric figures with numerical characteristics measured in inches includes triangles and squares $[\![s_{tt}]\!]$;

– the concept spaces *Euclidean* and *Riemannian* are elements of the order 2 of the attribute space *inch* in $[\![s_{tt}]\!]$ that means that classifications of spaces containing geometric figures with numerical characteristics measured in inches includes Euclidean and Riemannian spaces in $[\![s_{tt}]\!]$;

– the concept space spaces 2 and 3 are elements of the order 3 of the attribute space *inch* in $[\![s_{tt}]\!]$ that means that classifications of dimensions of spaces containing geometric figures with numerical characteristics measured in inches includes dimensions 2 and 3 in $[\![s_{tt}]\!]$.

10.3. Attributes of the order 3

An attribute a_{tt} in $[\![s_{tt}, 3]\!]$ models a space of attribute spaces in $[\![s_{s.q.i}]\!]$. Elements in $[\![attribute : a_{tt}, s_{tt}, attribute{-}order : 3]\!]$ are attribute spaces, direct attributes, individuals and concepts in $[\![s_{tt}]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-3 : 10, -2 : cm, -1 : perimeter, 0 : e_{l.g.2}, 1 : square, 2 : Riemannian, 3 : 3)$, and $[support\ s_{tt}] = \{c_{ncl.1}, c_{ncl.2}\}$. Then the following properties hold:

- the attribute space space 10 classifies numerical characteristics of geometric figures with values represented in decimal system;
- the attribute spaces *inch* and *cm* are elements of the order -2 of the attribute space space 10 in $[\![s_{tt}]\!]$ that means that classifications of units of measurement of numerical characteristics of geometric figures with values represented in decimal system includes inches and centimeters in $[\![s_{tt}]\!]$;
- the direct attributes *area* and *perimeter* are elements of the order -11 of the attribute space space 10 in $[\![s_{tt}]\!]$ that means that classifications of numerical characteristics of geometric figures with values represented in decimal system includes area and perimeter in $[\![s_{tt}]\!]$;
- the individuals $e_{l.g.1}$ and $e_{l.g.2}$ are elements of the order 0 of the attribute space space 10 in $[\![s_{tt}]\!]$ that means that classifications of geometric figures with numerical characteristics with values represented in decimal system includes $e_{l.g.1}$ and $e_{l.g.2}$ in $[\![s_{tt}]\!]$;
- the concepts *triangle* and *square* are elements of the order 1 of the attribute space space 10 in $[\![s_{tt}]\!]$ that means that classifications of geometric figures with numerical characteristics with values represented in decimal system includes triangles and squares in $[\![s_{tt}]\!]$;
- the concept spaces *Euclidean* and *Riemannian* are elements of the order 2 of the attribute space space 10 in $[\![s_{tt}]\!]$ that means that classifications of spaces containing geometric figures with numerical characteristics with values represented in decimal

system includes Euclidean space and Riemannian space in $[\![s_{tt}]\!]$;

– the concept space spaces 10 and 2 are elements of the order 3 of the attribute space space 10 in $[\![s_{tt}]\!]$ that means that classifications of dimensions of spaces containing geometric figures with numerical characteristics with values represented in decimal system includes dimensions 10 and 2 in $[\![s_{tt}]\!]$.

10.4. Attributes of higher orders

An attribute a_{tt} in $[\![s_{tt}, n_t]\!]$, where $n_t > 3$, is classified and interpreted in the similar way (by the introduction of spaces of attribute space spaces and so on.).

11. Classification of conceptuals

11.1. General principles and definitions

We use the two-level scheme of classification of conceptuals. The upper (first) level is defined by the maximal order of attributes of a conceptual. This level is described by the notion of concretization order of a conceptual. The lower (second) level is defined by the set of all element orders of a conceptual. This level is described by the notion of integral order of a conceptual.

11.1.1. Concretization orders of conceptuals

The number 0 is an order in $[\![c_{ncpl}]\!]$ if the minimal order in $[\![c_{ncpl}, element :]\!]$ is greater than or equal to 0. A number n_t is an order in $[\![c_{ncpl}]\!]$ if $-n_t$ is a minimal order in $[\![c_{ncpl}, element :]\!]$.

$\bigoplus$ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.2} = (-2 : inch, -1 : area, 0 : e_{l.g.2}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.3} = (-1 : area, 0 : e_{l.g.3}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.4} = (0 : e_{l.g.4}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.5} = (1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.6} = (2 : Euclidean, 3 : 2)$, and $c_{ncl.7} = (3 : 2)$. Then the conceptuals $c_{ncl.1}$, $c_{ncl.2}$, $c_{ncl.3}$ have the orders 3, 2, 1 and the conceptuals $c_{ncl.4}$, $c_{ncl.5}$, $c_{ncl.6}$, $c_{ncl.7}$ have the order 0.

Conceptuals of the order n_t concretizes conceptuals of the orders which are less than n_t. They define the special kinds of such conceptuals and are used to classify them. Concretization is performed by attributes of the order n_t and their values. Therefore, the order of a conceptual is also called the concretization order of the conceptual.

11.1.2. Integral orders of conceptuals

11.1.2.1. Integral orders

A set s_t is an integral order in $[\![c_{ncpl}]\!]$ if s_t is a set of all orders in $[\![c_{ncpl}, element :]\!]$.

⊕ Let $c_{ncl.1} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$, $c_{ncl.1} = (-3 : 10, -1 : area, 1 : triangle, 3 : 2)$, $c_{ncl.1} = (-2 : inch, -1 : area, 2 : Euclidean, 3 : 2)$. Then $o_{r.i}[\![c_{ncl.1}]\!] = \{-3, -2, -1, 0, 1, 2, 3\}$, $o_{r.i}[\![c_{ncl.2}]\!] = \{-3, -1, 1, 3\}$, and $o_{r.i}[\![c_{ncl.3}]\!] = \{-2, -1, 2, 3\}$.

11.1.2.2. Refined integral orders

A set s_t is a refined integral order in $[\![c_{ncpl}]\!]$ if s_t is a result of replacement of zero or more orders i_{nt} in $[\![[\![c_{ncpl}, element :]\!]]\!]$ in the set $o_{r.i}[\![c_{ncpl}]\!]$ by objects $i_{nt} : [c_{ncpl}\ i_{nt}]$. A refined integral order in $[\![c_{ncpl}]\!]$ refines an integral order in $[\![c_{ncpl}]\!]$, providing information on some elements of c_{ncpl} with their orders. Let $c_{ncpl} : o_{r.i.r}$ denote a conceptual c_{ncpl} which has the refined integral order $o_{r.i.r}$.

⊕ Let $c_{ncpl} = (-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2)$. Then $\{-3, -2, -1, 0, 1, 2, 3\}$, $\{-3, -2 : inch, -1, 0, 1 : triangle, 2, 3\}$ and $\{-3 : 10, -2 : inch, -1 : area, 0 : e_{l.g.1}, 1 : triangle, 2 : Euclidean, 3 : 2\}$ are refined integral orders in $[\![c_{ncpl}]\!]$.

11.1.2.3. Properties of integral orders

Proposition 8. A conceptual c_{ncpl} has the single integral order.

Proof. This follows from the definition of the integral order of a conceptual. □

Proposition 9. A conceptual c_{ncpl} has a finite set of refined integral orders.

Proof. This follows from the definition of the refined integral order and the finite number of orders of conceptuals in the context of elements. □

Proposition 10. The integral order in $[\![c_{ncpl}]\!]$ is a refined integral order in $[\![c_{ncpl}]\!]$.

Proof. This follows from the definition of the refined integral order of a conceptual. □

11.1.2.4. Notes

Conceptuals of the same concretization order are classified according to their integral orders. Each integral order defines a separate kind of conceptuals.

Conceptuals allow to model ontological elements in detail. Each kind of conceptuals models a separate kind of ontological elements.

11.2. Modelling of ontological elements by conceptuals of the order 0

In this section conceptuals of the order 0 is classified according to their integral orders and the ontological elements modelled by conceptuals of this classification is described.

A conceptual $c_{ncpl} : \{0\}$ models the individual $[c_{ncpl}\ 0]$.

$\bigoplus$ The conceptual $(0 : f_g)$ models the geometric figure f_g.

A conceptual $c_{ncpl} : \{0, 1\}$ models the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$.

$\bigoplus$ The conceptual $(0 : f_g, 1 : triangle)$ models the triangle f_g.

A conceptual $c_{ncpl} : \{1\}$ models the concept $[c_{ncpl}\ 1]$.

$\bigoplus$ A conceptual $(1 : triangle)$ models triangles.

A conceptual $c_{ncpl} : \{1, 2\}$ models the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(1 : triangle, 2 : Euclidean)$ models triangles in Euclidean space.

A conceptual $c_{ncpl} : \{2\}$ models the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(2 : Euclidean)$ models Euclidean space.

A conceptual $c_{ncpl} : \{0, 2\}$ models the individual $[c_{ncpl}\ 0]$ from the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(0 : f_g, 2 : Euclidean)$ models the geometric figure f_g in Euclidean space.

A conceptual $c_{ncpl} : \{0, 1, 2\}$ models the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(0 : f_g, 1 : triangle, 2 : Euclidean)$ models the triangle f_g in Euclidean space.

Classification of other conceptuals of the order 0 and description of the ontological elements modelled by these conceptuals is performed in a similar way (by the introduction of the concept space space and so on.). For example, a conceptual $c_{ncpl} : \{0, 1, 2, 3\}$ models the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$ from the concept space space $[c_{ncpl}\ 3]$.

$\bigoplus$ The conceptual $(0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$ models the triangle f_g in two-dimensional Euclidean space.

11.3. Modelling of ontological elements by conceptuals of the order 1

In this section conceptuals of the order 1 is classified according to their integral orders and the ontological elements modelled by conceptuals of this classification is described.

A conceptual $c_{ncpl} : \{-1\}$ models the attribute $[c_{ncpl}\ -1]$.

$\bigoplus$ The conceptual $(-1 : area)$ models area of geometric figures.

A conceptual $c_{ncpl} : \{-1, 0\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$.

$\bigoplus$ The conceptual $(-1 : area, 0 : f_g)$ models area of the geometric figure f_g.

A conceptual $c_{ncpl} : \{-1, 0, 1\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$.

$\bigoplus$ The conceptual $(-1 : area, 0 : f_g, 1 : triangle)$ models area of the triangle f_g.

A conceptual $c_{ncpl} : \{-1, 1\}$ models the attribute $[c_{ncpl}\ -1]$ of individuals from the concept $[c_{ncpl}\ 1]$.

$\bigoplus$ The conceptual $(-1 : area, 1 : triangle)$ models area of triangles.

A conceptual $c_{ncpl} : \{-1, 0, 1, 2\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(-1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean)$ models area of the triangle f_g in Euclidean space.

A conceptual $c_{ncpl} : \{-1, 1, 2\}$ models the attribute $[c_{ncpl}\ -1]$ of individuals from the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(-1 : area, 1 : triangle, 2 : Euclidean)$ models area of triangles in Euclidean space.

A conceptual $c_{ncpl} : \{-1, 0, 2\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$ from the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(-1 : area, 0 : f_g, 2 : Euclidean)$ models area of the geometric figure f_g in Euclidean space.

A conceptual $c_{ncpl} : \{-1, 2\}$ models the attribute $[c_{ncpl}\ -1]$ of individuals from concepts from the concept space $[c_{ncpl}\ 2]$.

$\bigoplus$ The conceptual $(-1 : area, 2 : Euclidean)$ models area of geometric figures in Euclidean space.

Correlation between other kinds of conceptuals of the order 1 and the corresponding kinds of ontological elements is performed in a similar way.

11.4. Modelling of ontological elements by conceptuals of the order 2

In this section conceptuals of the order 2 is classified according to their integral orders and the ontological elements modelled by conceptuals of this classification is described.

A conceptual $c_{ncpl}:\{-2,-1\}$ models the attribute $[c_{ncpl}\ -1]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area)$ models area measured in inches.

A conceptual $c_{ncpl}:\{-2,-1,0\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area,0:f_g)$ models area of the geometric figure f_g measured in inches.

A conceptual $c_{ncpl}:\{-2,-1,0,1\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area,0:f_g,1:triangle)$ models area of the triangle f_g measured in inches.

A conceptual $c_{ncpl}:\{-2,-1,1\}$ models the attribute $[c_{ncpl}\ -1]$ of individuals from the concept $[c_{ncpl}\ 1]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area,1:triangle)$ models area of triangles measured in inches.

A conceptual $c_{ncpl}:\{-2,-1,0,1,2\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area,0:f_g,1:triangle,2:Euclidean)$ models area of the triangle f_g in Euclidean space measured in inches.

A conceptual $c_{ncpl}:\{-2,-1,1,2\}$ models the attribute $[c_{ncpl}\ -1]$ of individuals from the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area,1:triangle,2:Euclidean)$ models area of triangles in Euclidean space measured in inches.

A conceptual $c_{ncpl}:\{-2,-1,0,2\}$ models the attribute $[c_{ncpl}\ -1]$ of the individual $[c_{ncpl}\ 0]$ from the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area,0:f_g,2:Euclidean)$ models area of the geometric figure f_g in Euclidean space measured in inches.

A conceptual $c_{ncpl}:\{-2,-1,2\}$ models the attribute $[c_{ncpl}\ -1]$ of individuals from concepts from the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

$\bigoplus$ The conceptual $(-2:inch,-1:area,2:Euclidean)$ models area of geometric figures in Euclidean space measured in inches.

A conceptual $c_{ncpl} : \{-2, 0\}$ models the individual $[c_{ncpl}\ 0]$ in the attribute space $[c_{ncpl}\ -2]$.

⊕ The conceptual $(-2 : inch, 0 : f_g)$ models the geometric figure f_g with numerical characteristics measured in inches.

A conceptual $c_{ncpl} : \{-2, 0, 1\}$ models the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$ in the attribute space $[c_{ncpl}\ -2]$.

⊕ The conceptual $(-2 : inch, 0 : f_g, 1 : triangle)$ models the triangle f_g with numerical characteristics measured in inches.

A conceptual $c_{ncpl} : \{-2, 1\}$ models the concept $[c_{ncpl}\ 1]$ in the attribute space $[c_{ncpl}\ -2]$.

⊕ The conceptual $(-2 : inch, 1 : triangle)$ models triangles with numerical characteristics measured in inches.

A conceptual $c_{ncpl} : \{-2, 1, 2\}$ models the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

⊕ The conceptual $(-2 : inch, 1 : triangle, 2 : Euclidean)$ models triangles in Euclidean space with numerical characteristics measured in inches.

A conceptual $c_{ncpl} : \{-2, 2\}$ models the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

⊕ The conceptual $(-2 : inch, 2 : Euclidean)$ models geometric figures in Euclidean space with numerical characteristics measured in inches.

A conceptual $c_{ncpl} : \{-2, 0, 2\}$ models the individual $[c_{ncpl}\ 0]$ from the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

⊕ The conceptual $(-2 : inch, 0 : f_g, 2 : Euclidean)$ models the geometric figure f_g in Euclidean space with numerical characteristics measured in inches.

A conceptual $c_{ncpl} : \{-2, 0, 1, 2\}$ models the individual $[c_{ncpl}\ 0]$ from the concept $[c_{ncpl}\ 1]$ from the concept space $[c_{ncpl}\ 2]$ in the attribute space $[c_{ncpl}\ -2]$.

⊕ The conceptual $(-2 : inch, 0 : f_g, 1 : triangle, 2 : Euclidean)$ models the triangle f_g in Euclidean space with numerical characteristics measured in inches.

Correlation between other kinds of conceptuals of the order 2 and the corresponding kinds of ontological elements is performed in a similar way.

11.5. Modelling of ontological elements by conceptuals of the higher orders

Classification of conceptuals of the order 3 or higher and description of the ontological elements modelled by conceptuals of this classification is performed in a similar way (by the introduction of the attribute space space and so on.).

$\bigoplus$ The conceptual $(-3 : 10, -2 : inch, -1 : area, 0 : f_g, 1 : triangle, 2 : Euclidean, 3 : 2)$ models area of the triangle f_g in two-dimensional Euclidean space measured in inches in decimal system.

12. Modelling of relations, types, domains, inheritance

12.1. Relations and their instances

Finite binary relations are modelled by direct concepts and their instances are modelled by the elements of the order 0 of these concepts, represented by pairs of elements.

Finite relations of the arity n_t are modelled by direct concepts and their instances are modelled by the elements of the order 0 of these concepts, represented by sequence elements of the length n_t.

Finite relations of the variable arity are modelled by direct concepts and their instances are modelled by the elements of the order 0 of these concepts, represented by sequence elements of the variable length.

12.2. Types and domains

Finite types are modelled by direct concepts and their values are modelled by the elements of the order 0 of these concepts. Domains as the special kind of finite types are also modelled by direct concepts and their values are modelled by the elements of the order 0 of these concepts.

Types of attributes of the order n_t are modelled by the special attribute *type* of the order $n_t + 1$. Values of this attribute are types.

$\bigoplus$ Let $c_{ncpl} = (-2 : type, -1 : area, 0 : f_g)$, and $s_{tt} = (c_{ncpl} : real)$. Then the area of the geometric figure f_g is a real number in $[\![s_{tt}]\!]$.

$\bigoplus$ Let $c_{ncpl} = (-2 : type, -1 : area, 0 : *)$, and $s_{tt} = (c_{ncpl} : real)$. Then the area of any geometric figure is a real number in $[\![s_{tt}]\!]$. The semantics of $*$ is defined in section **??**

.

12.3. Inheritance

12.3.1. Inheritance on elements

The usual inheritance relation on concepts is generalized to the inheritance relation on elements of the same order in $[\![s_{tt}]\!]$. It is modelled by the special direct concept *inheritance* and their instances are modelled by the elements of the order 0 of the concept *inheritance*, represented by the triples of elements. Elements of the triple specify the inheriting element, the inherited element and their order. An element e_l inherits from $e_{l.1}$ in $[\![s_{tt}, i_{nt}]\!]$ if $[s_{tt}\ (0 : (e_l, e_{l.1}, i_{nt}), 1 : inheritance)] \neq und$.

Inheritance on elements redefines interpretation *value* of conceptuals as follows:

- if $[s_{tt}\ c_{ncpl}] \neq und$, then $[value\ c_{ncpl}\ s_{tt}] = [s_{tt}\ c_{ncpl}]$;
- if $[s_{tt}\ c_{ncpl}] = und$, i_{nt} is a maximal order in $[\![c_{ncpl}, element :]\!]$, s_t is a set of $e_l[\![s_{tt}]\!]$ such that $[c_{ncpl}\ i_{nt}]$ inherits from e_l in $[\![s_{tt}, i_{nt}]\!]$, $s_t \neq \emptyset$, and $[value\ [c_{ncpl}\ i_{nt} : e_l]\ s_{tt}] = [value\ [c_{ncpl}\ i_{nt} : e_{l.1}]\ s_{tt}]$ for all $e_l, e_{l.1} \in s_t$, then $[value\ c_{ncpl}\ s_{tt}] = [value\ [c_{ncpl}\ i_{nt} : e_l]\ s_{tt}]$, where $e_l \in s_t$;
- otherwise, $[value\ c_{ncpl}\ s_{tt}] = und$.

12.3.2. Inheritance on direct concepts

The inheritance on direct concepts is the special case of the inheritance on elements.

A concept $c_{ncp.d}$ inherits from a concept $c_{o..pt.d.1}$ in $[\![s_{tt}]\!]$ if $c_{ncp.d}$ inherits from $c_{o..pt.d.1}$ in $[\![s_{tt}, 1]\!]$.

12.3.3. Inheritance on element sequences

The inheritance relation on elements is generalized to the inheritance relation on element sequences. This relation is modelled by the special direct concept *inheritance* :: *sq* and their instances are modelled by the elements of the order 0 of this concept, represented by the triples of sequence elements of the same length. The elements of the triple specify inheriting elements, inherited elements and their orders. An element $e_{l.(*)}$ inherits from $e_{l.(*).1}$ in $[\![s_{tt}, i_{nt.(*)}]\!]$ if $i_{nt.(*)} = (i_{nt.1}, ..., i_{nt.n_t})$, $i_{nt.1} < \ldots < i_{nt.n_t}$, $[len\ e_{l.(*)}] = [len\ e_{l.(*).1}] = n_t$, and $[s_{tt}\ (0 : (e_{l.(*)}, e_{l.(*).1}, i_{nt.(*)}), 1 : inheritance :: sq)] \neq und$.

Inheritance on ordered elements redefines interpretation *value* of conceptuals as follows:

- if $[s_{tt}\ c_{ncpl}] \neq und$, then $[value\ c_{ncpl}\ s_{tt}] = [s_{tt}\ c_{ncpl}]$;
- if
  - $[s_{tt}\ c_{ncpl}] = und$,
  - $i_{nt.1} < \ldots < i_{nt.n_t}$ are orders in $[\![c_{ncpl}, element :]\!]$,
  - for all i_{nt} if $i_{nt} \geq i_{nt.1}$ and i_{nt} is an order in $[\![c_{ncpl}, element :]\!]$, then i_{nt} coincides with one of the numbers $i_{nt.1}$, ..., $i_{nt.n_t}$,

– s_t is a set of $e_l[\![s_{tt}]\!]$ such that $([c_{ncpl}\ i_{nt.1}], \ldots, [c_{ncpl}\ i_{nt.n_t}])$ inherits from e_l in $[\![s_{tt}, (i_{nt.1}, \ldots, i_{nt.n_t})]\!]$,

– $s_t \neq \emptyset$,

– $[value\ [c_{ncpl}\ i_{nt.1} : [e_l\ .\ 1], \ldots, i_{nt.n_t} : [e_l\ .\ n_t]]\ s_{tt}] = [value\ [c_{ncpl}\ i_{nt.1} : [e_{l.1}\ .\ 1], \ldots, i_{nt.n_t} : [e_{l.1}\ .\ n_t]]\ s_{tt}]$ for each $e_l, e_{l.1} \in s_t$,

then $[value\ c_{ncpl}\ s_{tt}] = [value\ [c_{ncpl}\ i_{nt.1} : [e_l\ .\ 1], \ldots, i_{nt.n_t} : [e_l\ .\ n_t]]\ s_{tt}]$, where $e_l \in s_t$;

- otherwise, $[value\ c_{ncpl}\ s_{tt}] = und.$

13. Generic conceptuals

A generic conceptual defines a set of conceptuals satisfying a certain template and sets the default value for these conceptuals. Conceptuals from this set are called instances of the generic conceptual. The template of the generic conceptual is defined by its form.

13.1. The main definitions

13.1.1. Generic conceptuals

Let $* \in A_{tm}$. A conceptual $c_{ncpl}[\![s_{tt}]\!]$ is a generic conceptual in $[\![s_{tt}]\!]$ if there exists $o_{rd}[\![c_{ncpl}]\!]$ such that $[c_{ncpl}\ o_{rd}] \in \{*, (*, t_p), (*, t_p, p_{rm}), (*, *, p_{rm})\}$. The element $p_{l.s}$ of the form $[c_{ncpl}\ o_{rd}]$ from this definition is called a substitution place in $[\![c_{ncpl}, s_{tt}, o_{rd}]\!]$. The number o_{rd} is called an order in $[\![p_{l.s}, c_{ncpl}, s_{tt}]\!]$. The elements t_p and p_{rm} are called a type and parameter in $[\![p_{l.s}, c_{ncpl}, s_{tt}, o_{rd}]\!]$.

13.1.2. Kinds of generic conceptuals

A conceptual $c_{ncpl.g}$ is partially typed in $[\![s_{tt}]\!]$ if there exist $p_{l.s}$, t_p and o_{rd} such that $p_{l.s}$ is a substitution place in $[\![c_{ncpl.g}, s_{tt}, o_{rd}]\!]$ and t_p is a type in $[\![p_{l.s}, c_{ncpl.g}, s_{tt}, o_{rd}]\!]$.

A conceptual $c_{ncpl.g}$ is typed in $[\![s_{tt}]\!]$ if for all $p_{l.s}$ and o_{rd} if $p_{l.s}$ is a substitution place in $[\![c_{ncpl.g}, s_{tt}, o_{rd}]\!]$, then there exists t_p such that t_p is a type in $[\![p_{l.s}, c_{ncpl.g}, s_{tt}, o_{rd}]\!]$.

A conceptual $c_{ncpl.g}$ is parametric in $[\![s_{tt}]\!]$ if there exist $p_{l.s}$, p_{rm} and o_{rd} such that $p_{l.s}$ is a substitution place in $[\![c_{ncpl.g}, s_{tt}, o_{rd}]\!]$ and p_{rm} is a parameter in $[\![p_{l.s}, c_{ncpl.g}, s_{tt}, o_{rd}]\!]$.

13.1.3. Instances of generic conceptuals

A conceptual c_{ncpl} is an instance in $[\![c_{ncpl.g}, s_{tt}]\!]$, if the following properties hold:

- if $[c_{ncpl.g}\ i_{nt}]$ is not a substitution place in $[\![c_{ncpl.g}, s_{tt}, i_{nt}]\!]$, then $[c_{ncpl}\ i_{nt}] = [c_{ncpl.g}\ i_{nt}]$;

- if $[c_{ncpl.g}\ i_{nt}]$ is a substitution place in $[\![c_{ncpl.g}, s_{tt}, i_{nt}]\!]$, then $[c_{ncpl}\ i_{nt}]$ is an element in $[\![s_{tt}, i_{nt}]\!]$;
- if $[c_{ncpl.g}\ i_{nt}] \in \{(*, t_p), (*, t_p, p_{rm})\}$, then $[c_{ncpl}\ i_{nt}]$ is an element in $[\![concept : t_p, s_{tt}, concept{-}order : 1, element{-}order : 0]\!]$;
- if p_{rm} is a parameter in $[\![p_{l.s.1}, c_{ncpl.g}, s_{tt}, o_{r.e.1}]\!]$ and $[\![p_{l.s.2}, c_{ncpl.g}, s_{tt}, o_{r.e.2}]\!]$, then $[c_{ncpl}\ o_{r.e.1}] = [c_{ncpl}\ o_{r.e.2}]$.

13.1.4. States with generic conceptuals

A state s_{tt} is a state with generic conceptuals, if the following properties hold:

- *(the consistency property)* if $c_{ncl.g.1} \neq c_{ncl.g.2}$, then there is no c_{ncpl} such that c_{ncpl} is an instance of $c_{ncl.g.1}$ in $[\![s_{tt}]\!]$ and c_{ncpl} is an instance of $c_{ncl.g.2}$ in $[\![s_{tt}]\!]$;
- interpretation *value* of conceptuals is redefined as follows:
  - if $[s_{tt}\ c_{ncpl}] \neq und$, then $[value\ c_{ncpl}\ s_{tt}] = [s_{tt}\ c_{ncpl}]$;
  - if $[s_{tt}\ c_{ncpl}] = und$ and c_{ncpl} is an instance in $[\![c_{ncpl.g}, s_{tt}]\!]$, then $[value\ c_{ncpl}\ s_{tt}] = [s_{tt}\ c_{ncpl.g}]$;
  - otherwise, $[value\ c_{ncpl}\ s_{tt}] = und$.

13.2. Examples of generic conceptuals

A conceptual $c_{ncpl.g} : \{-1, 0 : *, 1\}$ models the property that the value of the attribute $[c_{ncpl.g}\ -1]$ of individuals from the concept $[c_{ncpl.g}\ 1]$ equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined explicitly.

⊕ The conceptual $c_{ncpl.g} = (-1 : area, 0 : *, 1 : triangle)$ models the property that area of triangles equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined explicitly.

A conceptual $c_{ncpl.g} : \{-1, 0 : *\}$ models the property that the value of the attribute $[c_{ncpl.g}\ -1]$ of individuals equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined explicitly.

⊕ The conceptual $c_{ncpl.g} = (-1 : area, 0 : *)$ models the property that area of geometric figures equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined explicitly.

A conceptual $c_{ncpl.g} : \{0 : *, 1\}$ models the property that the value of individuals from the concept $[c_{ncpl.g}\ 1]$ equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined explicitly.

⊕ The conceptual $c_{ncpl.g} = (0 : *, 1 : triangle)$ models the property that the value of triangles equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined explicitly. What is the value of a triangle depends on interpretation.

13.3. Modelling of ontological elements and their properties based on generic conceptuals

Generic conceptuals together with attributes allow to model ontological elements and their properties in more detail.

A conceptual $c_{ncpl.g} : \{-2 : type, -1, 0 : *, 1\}$ models the property that the type of the attribute $[c_{ncpl.g}\ -1]$ of individuals from the concept $[c_{ncpl.g}\ 1]$ equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for individuals explicitly.

$\bigoplus$ The conceptual $c_{ncpl.g} = (-2 : type, -1 : area, 0 : *, 1 : triangle)$ models the property that the type of the attribute *area* of triangles equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for triangles explicitly.

A conceptual $c_{ncpl.g} : \{-2 : type, -1, 0 : *\}$ models the property that the type of the attribute $[c_{ncpl.g}\ -1]$ of individuals equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for individuals explicitly.

$\bigoplus$ The conceptual $c_{ncpl.g} = (-2 : type, -1 : area, 0 : *)$ models the property that the type of the attribute *area* of geometric figures equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for geometric figures explicitly.

A conceptual $c_{ncpl.g} : \{-2 : type, 0 : *\}$ models the property that the type of individuals equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for individuals explicitly.

$\bigoplus$ The conceptual $c_{ncpl.g} = (-2 : type, 0 : *)$ models the property that the type of geometric figures equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for geometric figures explicitly.

A conceptual $c_{ncpl.g} : \{-2 : type, 0 : *, 1\}$ models the property that the type of individuals from the concept $[c_{ncpl.g}\ 1]$ equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for such individuals explicitly.

$\bigoplus$ The conceptual $c_{ncpl.g} = (-2 : type, 0 : *, 1 : triangle)$ models the property that the type of triangles equals $[s_{tt}\ c_{ncpl.g}]$ in $[\![s_{tt}]\!]$ if it is not defined for triangles explicitly.

14. The CCSL language

The CCSL language (Conceptual Configuration System Language) is a basic language of CCSs. Interpretable elements of CCSL are called basic elements of CCSs.

Let $s_b \subseteq (x : x_0,\ y : y_0,\ z : z_0,\ u : u_0,\ v : v_0,\ w : w_0,\ x_1 : x_{1.0},\ ...,\ x_{n_t} : x_{n_t.0},\ conf :: in : c_{nf})$.

14.1. Syntax of CCSL

An object o_b is an atom in CCSL if

- o_b is a sequence of Unicode symbols except for the whitespace symbols and the symbols ", ′, (,), ;, ,, and :, or
- o_b is a special atom, or
- o_b has the form "$o_{b.1}$" called a string, where $o_{b.1}$ is a sequence of Unicode symbols in which each occurrence of the symbol " is preceded by the symbol ′ and each occurrence of the symbol ′ is doubled.

The set $A_{to.s}$ of special atoms includes the object ::= and can be extended.

An object o_b is an element in CCSL if $o_b \in A_{tm}$, $o_b = e_l : e_{l.1}$, $o_b = (e_{l.*})$, or $o_b = e_l :: e_{l.1}$.

The whitespace symbols and the semicolon in CCSL are element delimiters along with comma. For example, $(e_{l.1},\ e_{l.2})$, $(e_{l.1};\ e_{l.2})$ and $(e_{l.1}\ e_{l.2})$ represent the same element.

An element $e_{l.a}$ is a conceptual in CCSL if all its attributes are integers.

An element $e_{l.a}$ is a conceptual state in CCSL if all its attributes are conceptuals.

An element $e_{l.a}$ is a conceptual configuration in CCSL if $[image\ e_{l.a}] \subseteq S_{tt}$.

The element $(pattern\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*}))$ in CCSL represents the pattern specification $(p_t, (v_{r.*}), (v_{r.s.*}))$.

The element $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ b_d) :: name :: n_m$ in CCSL represents the element definition $(p_t, (v_{r.*}), (v_{r.s.*}), b_d)$ with the name n_m.

For simplicity, we omit the names of interpretations and definitions below.

14.2. The special forms for interpretations and definitions

In this section we define the special forms for interpretations and definitions used below.

The form $(interpretation\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ f_n) :: name :: n_m$ denotes the interpretation $(p_t, (v_{r.*}), (v_{r.s.*}), f_n)$ with the name n_m.

The objects $var\ (v_{r.*})$ and $seq\ (v_{r.s.*})$ in the form $(interpretation\ ...)$ can be omitted. The omitted objects correspond to $var\ ()$ and $seq\ ()$, respectively.

Let $\{v_{r.*}\}$, $\{v_{r.s.*}\}$, $\{v_{r.*.1}\}$ and $\{v_{r.*.2}\}$ are pairwise disjoint, and $\{v_{r.*.3}\} \subseteq \{v_{r.*}\} \cup \{v_{r.*.1}\} \cup \{v_{r.*.2}\}$. The form $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.1})\ und\ (v_{r.*.2})\ val\ (v_{r.*.3})\ where\ c_{nd}\ then\ b_d)$ called a definition form is defined as follows:

- $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ where\ c_{nd}\ then\ b_d)$ is a shortcut for $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.1})\ und\ (v_{r.*.2})\ val\ (v_{r.*.3})\ then\ (if\ c_{nd}\ then\ b_d\ else\ und))$;

- $(definition\, p_t\, var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3},\ v_r)\ then\, b_d)$ is a shortcut for $(definition\, p_t\, var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ then\ (let\ w\ be\ v_r\ in\ [subst\ (v_r :: * : w)\ b_d]))$, where w is a new element that does not occur in this definition;
- $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ ()\ then\ b_d)$ is a shortcut for $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ then\ b_d)$;
- $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1},\ v_r)\ abn\ (v_{r.*.2})\ then\ b_d)$ is a shortcut for $(definition\, p_t\, var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ then\ (if\ (v_r\ is\ undefined)\ then\ und\ else\ b_d))$;
- $(definition\, p_t\, var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ ()\ abn\ (v_{r.*.2})\ then\, b_d)$ is a shortcut for $(definition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2})\ then\ b_d)$;
- $(definition\, p_t\, var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2},\ v_r)\ then\, b_d)$ is a shortcut for $(definition\, p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2})\ then\ (if\ (v_r\ is\ abnormal)\ then\ v_r\ else\ b_d))$;
- $(definition\, p_t\, var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ ()\ then\, b_d)$ is a shortcut for $(definition\, p_t\, var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ b_d)$.

The element c_{nd} specifies the restriction on the values of the pattern variables. The undefined value is propagated through the variables of $v_{r.*.1}$. Abnormal values are propagated through the variables of $v_{r.*.2}$. The special element $v_r :: *$ references to the value of element associated with the pattern variable v_r. A pattern variable is evaluated if the element associated with it is evaluated. Thus, the sequence $v_{r.*.3}$ contains evaluated pattern variables. A pattern variable is quoted if the element associated with it is not evaluated. Let $F_{rm.d}$ be a set of definition forms.

The objects $var\ (v_{r.*})$, $seq\ (v_{r.s.*})$, $und\ (v_{r.*.1})$, $abn\ (v_{r.*.2})$, $val\ (v_{r.*.3})$ and $where\ c_{nd}$ in the form $(definition\ ...)$ can be omitted. The omitted objects correspond to $var\ ()$, $seq\ ()$, $und\ ()$, $abn\ ()$, $val\ ()$ and $where\ true$, respectively.

15. Semantics of interpretable elements in CCSL

15.1. Abnormal elements operations

The element und is defined as follows:

$(definition\ \ und\ \ then\ \ und :: q)$.

The element e_{xc} is defined as follows:

$(definition\ \ x\ \ var\ \ (x)\ \ where\ \ (x\ \ is\ \ exception)\ \ then\ \ x :: q) :: name :: (\text{"@"},\ \ exception)$.

The definition satisfies the property: $n_m \prec_{[\![o_{rd.intr}]\!]} ("@", exception)$ for each n_m such that n_m is a name of an atomic element interpretation or element definition with the pattern distinct from v_r, where v_r is a variable of this pattern.

The element $e_l :: q$ is defined as follows:

$(interpretation\ \ x :: q\ \ var\ \ (x)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = x_0$.

15.2. Statements

The element $(if\ x\ then\ y\ else\ z)$ is defined as follows:

$(definition\ \ (if\ \ x\ \ then\ \ y\ \ else\ \ z)\ \ var\ \ (x,\ \ y,\ \ z)\ \ val\ \ (x)$

$\ then\ \ (if\ \ x :: *\ \ then\ \ y\ \ else\ \ z) :: atm)$;

$(interpretation\ \ (if\ \ x\ \ then\ \ y\ \ else\ \ z) :: atm\ \ var\ \ (x,\ \ y,\ \ z)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \neq und]\ then\ [value\ y_0\ [s_b\ conf :: in]]\ else\ [value\ z_0\ [s_b\ conf :: in]]]$.

The element $(if\ x\ then\ y\ elseif\ z\ then\ u\ ...\ else\ v)$ is defined as follows:

$(definition\ \ (if\ \ x\ \ then\ \ y\ \ elseif\ \ z)\ \ var\ \ (x,\ \ y,\ \ z)\ \ seq\ \ (z)$

$\ then\ \ (if\ \ x\ \ then\ \ y\ \ else\ \ (if\ \ z)))$.

The element $(let\ x\ be\ y\ in\ z)$ is defined as follows:

$(interpretation\ \ (let\ \ x\ \ be\ \ y\ \ in\ \ z)\ \ var\ \ (x,\ \ y,\ \ z)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [value\ [subst\ (x_0 : [value\ y_0\ [s_b\ conf :: in]])\ z_0]\ [s_b\ conf :: in]]$.

The element e_l of the form $(let\ ::\ seq\ x\ be\ y\ in\ z)$, where $x \in E_{l.(*)}$, $y \in E_{l.(*)}$, and $[len\ x] = [len\ y]$, is defined by the rule

$(rule\ \ (let :: seq\ \ x,\ \ y\ \ be\ \ z,\ \ u\ \ in\ \ v)\ \ var\ \ (x,\ \ z,\ \ v)\ \ seq\ \ (y,\ \ u)$

$\ then\ \ (let\ \ x\ \ be\ \ z\ \ in\ \ (let :: seq\ \ y\ \ be\ \ u\ \ in\ \ v)))$;

$(rule\ \ (let :: seq\ \ be\ \ in\ \ v)\ \ var\ \ (v)\ \ then\ \ v)$.

The elements x, y and z are called a substitution variables specification, substitution values specification and substitution body in $[\![e_l]\!]$. The elements of x and y are called substitution variables and substitution values in $[\![e_l]\!]$.

15.3. Characteristic functions for defined concepts

An object $d_{f.c}$ is a concept definition if $d_{f.c}$ is an interpretation of the form $(interpretation$ $(e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ f_n) :: name :: n_m$, or $d_{f.c}$ is a definition of the form $(definition\ (e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ b_d) :: name :: n_m$. Concept definitions specify concepts and their instances. Concepts specified by them are called defined concepts. The elements $e_{l.1}$ and $e_{l.2}$ are called an instance pattern and concept pattern in $[\![d_{f.c}]\!]$. The element $(e_{l.1}\ is\ e_{l.2})$ is called a characteristic function in $[\![d_{f.c}]\!]$. Let $D_{f.c}$ be a set of concept definitions.

An element $c_{ncp.d}$ is a defined concept in $[\![d_{f.c}, s_b]\!]$ if c_{ncp} is an instance in $[\![(e_{l.2}, var\ (v_{r.*})\ seq\ (v_{r.s.*})), m_t, s_b]\!]$. An element $c_{ncp.d}$ is a defined concept in $[\![d_{f.c}]\!]$ if there exists s_b such that $c_{ncp.d}$ is a defined concept in $[\![d_{f.c}, s_b]\!]$. An element $c_{ncp.d}$ is a defined concept in $[\![c_{nf}]\!]$ if there exists $d_{f.c}[\![c_{nf}]\!]$ such that $c_{ncp.d}$ is a defined concept in $[\![d_{f.c}]\!]$. Let $C_{ncp.d}$ be a set of defined concepts.

An element i_{nstn} is an instance in $[\![d_{f.c}, s_b]\!]$ if i_{nstn} is an instance in $[\![(e_{l.1}, var\ (v_{r.*})\ seq\ (v_{r.s.*})), m_t, s_b]\!]$. An element i_{nstn} is an instance in $[\![d_{f.c}]\!]$ if there exists s_b such that $c_{ncp.d}$ is an instance in $[\![d_{f.c}, s_b]\!]$.

An element i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}, d_{f.c}]\!]$ if i_{nstn} is an instance in $[\![d_{f.c}, c_{ncp.d}]\!]$ is a defined concept in $[\![d_{f.c}]\!]$, and $[value\ (i_{nstn}\ is\ c_{ncp.d})\ c_{nf}\ (n_m)] \neq und$. An element i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}]\!]$ if there exists $d_{f.c}$ such that i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}, d_{f.c}]\!]$. An element $c_{ncp.d}$ is an instance in $[\![c_{nf}, m_t]\!]$ if there exists $c_{ncp.d}$ such that i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}]\!]$. Let I_{nstn} be a set of instances.

A set s_t is called a content in $[\![c_{ncp.d}, c_{nf}]\!]$ if s_t is a set of all i_{nstn} such that i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}]\!]$. Let $[content\ c_{ncp.d}\ c_{nf}]$ denote the content in $[\![c_{ncp.d}, c_{nf}]\!]$.

The notion of defined concepts is extended to the definitions of the form $(definition\ (e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ val\ (v_{r.*.3})\ where\ c_{nd}\ then\ b_d)$. Let d_f have this form. An element $c_{ncp.d}$ is a defined concept in $[\![d_f, s_b]\!]$ if $c_{ncp.d}$ is a defined concept in $[\![d_{f.1}, s_b]\!]$, where $d_{f.1}$ is a definition of the form $(definition\ (e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ b_{d.1})$ such that d_f is reduced to $d_{f.1}$.

The element $(x\ is\ atom)$ specifying that x is an atom is defined as follows:

$(interpretation\ \ (x\ \ is\ \ atom)\ \ var\ \ (x)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in A_{tm}]\ then\ true\ else\ und]$.

The element $(x\ is\ update)$ specifying that x is an element update is defined as follows:

$(interpretation\ \ (x\ \ is\ \ update)\ \ var\ \ (x)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in U_{p.e}]\ then\ true\ else\ und]$.

The element $(x\ is\ multi{-}attribute)$ specifying that x is a multi-attribute element is defined as follows:

$(interpretation\ \ (x\ \ is\ \ multi{-}attribute)\ \ var\ \ (x)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in E_{l.ma}]\ then\ true\ else\ und]$.

The element $(x\ is\ attribute)$ specifying that x is an attribute element is defined as follows:
$(interpretation\ \ (x\ \ is\ \ attribute)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in E_{l.a}]\ then\ true\ else\ und]$.

The element $(x\ is\ sorted)$ specifying that x is a sorted element is defined as follows:
$(interpretation\ \ (x\ \ is\ \ sorted)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in E_{l.s}]\ then\ true\ else\ und]$.

The element $(x\ is\ undefined)$ specifying that x equals und is defined as follows:
$(interpretation\ \ (x\ \ is\ \ undefined)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 = und]\ then\ true\ else\ und]$.

The element $(x\ is\ defined)$ specifying that x does not equal und is defined as follows:
$(interpretation\ \ (x\ \ is\ \ defined)\ \ var\ \ (x)\ \ then\ \ f_n)$.
where $[f_n\ s_b] = [if\ [x_0 \neq und]\ then\ true\ else\ und]$.

The element $(x\ is\ exception)$ specifying that x is an exception is defined as follows:
$(interpretation\ \ (x\ \ is\ \ exception)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in E_{xc}]\ then\ true\ else\ und]$.

The element $(x\ is\ normal)$ specifying that x is a normal element is defined as follows:
$(interpretation\ \ (x\ \ is\ \ normal)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in E_{l.n}]\ then\ true\ else\ und]$.

The element $(x\ is\ abnormal)$ specifying that x is an abnormal element is defined as follows:
$(interpretation\ \ (x\ \ is\ \ abnormal)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in E_{l.ab}]\ then\ true\ else\ und]$.

The element $(x\ is\ sequence)$ specifying that x is a sequence element is defined as follows:
$(interpretation\ \ (x\ \ is\ \ sequence)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in E_{l.(*)}]\ then\ true\ else\ und]$.

The element $(x\ is\ set)$ specifying that the elements of the sequence element x are pairwise distinct is defined as follows:
$(definition\ \ (x\ \ is\ \ set)\ \ var\ \ (x)\ \ where\ \ (x\ \ is\ \ sequence)\ \ then\ \ (x\ \ is\ \ set) :: atm)$;
$(interpretation\ \ (x\ \ is\ \ set) :: atm\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ \ s_b] = [if\ \ [[x_0\ \ .\ \ n_{t.1}] \neq [x_0\ \ .\ \ n_{t.2}]$ for all $n_{t.1}$ and $n_{t.2}$ such that $n_{t.1} \neq n_{t.2}, n_{t.1} \leq [len\ x_0]$ and $n_{t.2} \leq [len\ x_0]]\ then\ true\ else\ und]$.

The element $(x\ is\ empty)$ specifying that x is an empty element is defined as follows:
$(definition\ \ (x\ \ is\ \ empty)\ \ var\ \ (x)\ \ then\ \ (x :: q\ \ =\ \ ()))$.

The element (x *is nonempty*) specifying that x is not an empty element is defined as follows:
$(definition\ \ (x\ \ is\ \ nonempty)\ \ var\ \ (x)\ \ then\ \ (x :: q\ \ != \ \ ()))$.

The element (x *is conceptual*) specifying that x is a conceptual is defined as follows:
$(interpretation\ \ (x\ \ is\ \ conceptual)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in C_{ncpl}]\ then\ true\ else\ und]$.

The element (x *is* (*conceptual in* y)) specifying that x is a conceptual in the context of the state y is defined as follows:
$(definition\ \ (x\ \ is\ \ (conceptual\ \ in\ \ y))\ \ var\ \ (x,\ \ y)$
$where\ \ ((x\ \ is\ \ conceptual)\ \ and\ \ (y\ \ is\ \ state))\ \ then\ \ (x\ \ is\ \ conceptual\ \ in\ \ y) :: atm)$;
$(interpretation\ \ (x\ \ is\ \ (conceptual\ \ in\ \ y)) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in C_{ncpl}[\![y_0]\!]]\ then\ true\ else\ und]$.

The element (x *is state*) specifying that x is a conceptual state is defined as follows:
$(interpretation\ \ (x\ \ is\ \ state)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in S_{tt}]\ then\ true\ else\ und]$.

The element (x *is configuration*) specifying that x is a conceptual configuration is defined as follows:
$(interpretation\ \ (x\ \ is\ \ configuration)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in C_{nf}]\ then\ true\ else\ und]$.

The element (x *is nat*) specifying that x is a natural number is defined as follows:
$(interpretation\ \ (x\ \ is\ \ nat)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in N_t]\ then\ true\ else\ und]$.

The element (x *is nat*0) specifying that x is either a natural number, or a zero is defined as follows:
$(interpretation\ \ (x\ \ is\ \ nat0)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in N_{t0}]\ then\ true\ else\ und]$.

The element (x *is int*) specifying that x is an integer is defined as follows:
$(interpretation\ \ (x\ \ is\ \ int)\ \ var\ \ (x)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ [x_0 \in I_{nt}]\ then\ true\ else\ und]$.

The element (x *is* (*satisfiable in* y)) specifying that x is satisfiable in the context of variables y is defined as follows:
$(definition\ \ (x\ \ is\ \ (satisfiable\ \ in\ \ y))\ \ var\ \ (x,\ \ y)\ \ where\ \ (y\ \ is\ \ sequence)$
$then\ \ (x\ \ is\ \ (satisfiable\ \ in\ \ y)) :: atm)$;

$(interpretation\ \ (x\ \ is\ \ (satisfiable\ \ in\ \ y))::atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0$ is satisfiable in $[\![(y_0, [s_b\ conf :: in])]\!]]\ then\ true\ else\ und]$.

The element $(x\ is\ (valid\ in\ y))$ specifying that x is valid in the context of variables y is defined as follows:

$(definition\ \ (x\ \ is\ \ (valid\ \ in\ \ y))\ \ var\ \ (x,\ \ y)\ \ where\ \ (y\ \ is\ \ sequence)$
$\ then\ \ (x\ \ is\ \ (valid\ \ in\ \ y))::atm)$;

$(interpretation\ \ (x\ \ is\ \ (valid\ \ in\ \ y))::atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0$ is valid in $[\![(y_0, [s_b\ conf :: in])]\!]]\ then\ true\ else\ und]$.

The element $(x\ is\ (sequence\ y))$ specifying that x is a sequence element such that the value in $[\![(e_l\ is\ y)]\!]$ does not equal und for each element e_l of x is defined as follows:

$(definition\ \ ((x\ \ y)\ \ is\ \ (sequence\ \ z))\ \ var\ \ (x,\ \ z)\ \ seq\ \ (y)$
$\ then\ \ ((x\ \ is\ \ z)\ \ and\ \ ((y)\ \ is\ \ (sequence\ \ z)))$;

$(definition\ \ (()\ \ is\ \ (sequence\ \ x))\ \ var\ \ (x)\ \ then\ \ true)$.

15.4. Elements operations

The element $()$ is defined as follows:

$(definition\ \ ()\ \ then\ \ ()::q)$.

The element $(len\ x)$ specifying the length of the element x is defined as follows:

$(definition\ \ (len\ \ x)\ \ var\ \ (x)\ \ val\ \ (x)\ \ then\ \ (len\ \ x::*)::atm)$;

$(interpretation\ \ (len\ \ x)::atm\ \ var\ \ (x)\ \ then\ \ f_n)$,

where

- if $x_0 \in A_{tm} \cup U_{p.e} \cup E_{l.s}$, then $[f_n\ s_b] = 1$;
- if $x_0 = (e_{l.*})$, then $[f_n\ s_b] = [len\ e_{l.*}]$.

The element $(x\ = \ y)$ specifying the equality of the elements x and y is defined as follows:

$(definition\ \ (x\ \ =\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)$
$\ then\ \ (x::*\ \ =\ \ y::*)::atm)$;

$(interpretation\ \ (x\ \ =\ \ y)::atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where

- if x_0 and y_0 are equal atoms, then $[f_n\ s_b] = true$;
- if $x_0 \in U_{p.e}$, $y_0 \in U_{p.e}$, $a_{rg}[\![x_0]\!] = a_{rg}[\![y_0]\!]$, and $v_l[\![x_0]\!] = v_l[\![y_0]\!]$, then $[f_n\ s_b] = true$;
- if $x_0 \in E_{l.s}$, $y_0 \in E_{l.s}$, $e_l[\![x_0]\!] = e_l[\![y_0]\!]$, and $s_{rt}[\![x_0]\!] = s_{rt}[\![y_0]\!]$, then $[f_n\ s_b] = true$;
- if $x_0 \in E_{l.(*)}$, $y_0 \in E_{l.(*)}$, and x_0 and y_0 are equal sequences, then $[f_n\ s_b] = true$;

• otherwise, $[f_n\ s_b] = und$.

The element $(x\ !=\ y)$ specifying the inequality of the elements x and y is defined in the similar way.

The element $(x\ .\ y)$ specifying the y-th element of the sequence element x is defined as follows:

$(definition\ \ (x\ \ .\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)$

$\ where\ \ ((x :: *\ \ is\ \ sequence)\ \ and\ \ (y :: *\ \ is\ \ nat))\ \ then\ \ (x :: *\ \ .\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ .\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [x_0\ .\ y_0]$.

The element $(x\ ..\ y)$ specifying the value of the attribute element x for the attribute y is defined as follows:

$(definition\ \ (x\ \ ..\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x)\ \ where\ \ (x :: *\ \ is\ \ attribute)$

$\ then\ \ (x :: *\ \ ..\ \ y) :: atm)$;

$(interpretation\ \ (x\ \ ..\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [x_0\ y_0]$.

The element $(x\ +\ y)$ specifying the concatenation of the sequence elements x and y is defined as follows:

$(definition\ \ (x\ \ +\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)$

$\ where\ \ ((x :: *\ \ is\ \ sequence)\ \ and\ \ (y :: *\ \ is\ \ sequence))\ \ then\ \ (x :: *\ \ +\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ +\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = (e_{l.*}\ e_{l.1.*})$ for some $e_{l.*}$ and $e_{l.1.*}$ such that $x_0 = (e_{l.*})$ and $y_0 = e_{l.1.*}$.

The element $(x\ .+\ y)$ specifying the addition of the element x to the head of the sequence element y is defined as follows:

$(definition\ \ (x\ \ .+\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)\ \ where\ \ (y :: *\ \ is\ \ sequence)$

$\ then\ \ (x :: *\ \ .+\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ .+\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [y_0 = (e_{l.*})$ for some $e_{l.*}]\ then\ (x_0\ e_{l.*})\ else\ und]$.

The element $(x\ .+ ::\ set\ y)$ specifying the addition of the element x to the head of the sequence element y representing a set is defined as follows:

$(definition\ \ (x\ \ .+ :: set\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)\ \ where\ \ (y :: *\ \ is\ \ set)$

$\ then\ \ (x :: *\ \ .+ :: set\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ .+ :: set\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$, where $[f_n\ s_b] = [if\ [y_0 = (e_{l.*})$ for some $e_{l.*}]\ then\ [if\ [x_0 \in e_{l.*}]\ then\ (e_{l.*})\ else\ (x_0\ e_{l.*})]\ else\ und]$.

The element $(x\ \ +.\ y)$ specifying the addition of the element y to the tail of the sequence element x is defined as follows:

$(definition\ \ (x\ \ +.\ y)\ \ var\ \ (x,\ y)\ \ val\ \ (x,\ y)\ \ where\ \ (x :: *\ \ is\ \ sequence)$

$\ then\ \ (x :: *\ \ +.\ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ +.\ y) :: atm\ \ var\ \ (x,\ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 = (e_{l.*})$ for some $e_{l.*}]\ then\ (e_{l.*}\ y_0)\ else\ und]$.

The element $(x\ \ +.::set\ y)$ specifying the addition of the element y to the tail of the sequence element x representing a set is defined as follows:

$(definition\ \ (x\ \ +.::set\ \ y)\ \ var\ \ (x,\ y)\ \ val\ \ (x,\ y)\ \ where\ \ (x :: *\ \ is\ \ set)$

$\ then\ \ (x :: *\ \ +.::set\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ +.::set\ \ y) :: atm\ \ var\ \ (x,\ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 = (e_{l.*})$ for some $e_{l.*}]\ then\ [if\ [y_0 \in e_{l.*}]\ then\ (e_{l.*})\ else\ (e_{l.*}\ y_0)]\ else\ und]$.

The element $(x\ \ -.::set\ y)$ specifying the deletion of the element y from the sequence element x representing a set is defined as follows:

$(definition\ \ (x\ \ -.::set\ \ y)\ \ var\ \ (x,\ y)\ \ val\ \ (x,\ y)\ \ where\ \ (x :: *\ \ is\ \ set)$

$\ then\ \ (x :: *\ \ -.::set\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ -.::set\ \ y) :: atm\ \ var\ \ (x,\ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 = (e_{l.*.1}\ y_0\ e_{l.*.2})$ for some $e_{l.*.1}$ and $e_{l.*.2}]\ then\ (e_{l.*.1}\ e_{l.*.2})\ else\ [if\ [x_0 = (e_{l.*})$ for some $e_{l.*}]\ then\ (e_{l.*})\ else\ und]]$.

The element $(upd\ x\ y_1 : z_1, ..., y_{n_t} : z_{n_t})$ specifying the sequential updates of the attribute element x at the points y_1, ..., y_{n_t} by z_1, ..., z_{n_t} is defined as follows:

$(definition\ \ (upd\ \ x\ \ y)\ \ var\ \ (x)\ \ seq\ \ (y)\ \ val\ \ (x)$

$\ where\ \ ((x :: *\ \ is\ \ attribute)\ \ and\ \ ((y)\ \ is\ \ (sequence\ \ update)))\ \ then\ \ (upd :: att\ \ x :: *\ \ y))$;

$(definition\ \ (upd :: att\ \ x\ \ y\ \ z)\ \ var\ \ (y)\ \ seq\ \ (z)\ \ und\ \ (x)$

$\ then\ \ (let\ \ w\ \ be\ \ (upd1 :: att\ \ x\ \ y)\ \ in\ \ (upd :: att\ \ w\ \ z)))$;

$(definition\ \ (upd :: att\ \ x)\ \ var\ \ (x)\ \ then\ \ x)$;

$(definition\ \ (upd1 :: att\ \ x\ \ y : z)\ \ var\ \ (x,\ y,\ z)\ \ val\ \ (z)$

$\ then\ \ (upd1 :: att\ \ x\ \ y : z :: *) :: atm)$;

$(interpretation\ \ (upd1 :: att\ \ x\ \ y : z) :: atm\ \ var\ \ (x,\ y,\ z)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [x_0\ y_0 : z_0]$.

The element $(upd\ x\ y : z)$ specifying the update of the sequence element x at the index y by z is defined as follows:

$(definition\ \ (upd\ \ x\ \ y\ \ z)\ \ var\ \ (x,\ \ y,\ \ z)\ \ val\ \ (x,\ \ y,\ \ z)$
$\ where\ \ ((x :: *\ is\ \ sequence)\ \ and\ \ (y :: *\ is\ \ nat)\ \ and\ \ (y :: *\ \ <=\ \ ((len\ \ x :: * :: q)\ \ +\ \ 1)))$
$\ then\ \ (upd :: seq\ \ x :: *\ \ y :: *\ \ z :: *) :: atm)$;
$(interpretation\ \ (upd :: seq\ \ x\ \ y : z) :: atm\ \ var\ \ (x,\ \ y,\ \ z)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [att{-}obj{-}to{-}seq\ [[seq{-}to{-}att{-}obj\ x_0]\ y_0 : z_0]]$.

The element $(x\ in :: set\ y)$ specifying that x is an element of the sequence element y is defined as follows:

$(definition\ \ (x\ \ in :: set\ \ y)\ \ var\ \ (x,\ \ y)\ \ where\ \ (y\ \ is\ \ sequence)$
$\ then\ \ (x\ \ in :: set\ \ y) :: atm)$;
$(interpretation\ \ (x\ \ in :: set\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [x_0 \in y_0]$.

The element $(x\ includes :: set\ y)$ specifying that the sequence element x includes the elements of the sequence element y is defined as follows:

$(definition\ \ (x\ \ includes :: set\ \ y)\ \ var\ \ (x,\ \ y)$
$\ where\ \ ((x\ \ is\ \ sequence)\ \ and\ \ (y\ \ is\ \ sequence))\ \ then\ \ (x\ \ includes :: set\ \ y) :: atm)$;
$(interpretation\ \ (x\ \ includes :: set\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = [if\ \ [e_l \in x_0$ for each $e_l \in y_0]\ then\ true\ else\ und]$.

The element $(attributes\ in\ x)$ specifying the sequence of attributes of the attribute element x is defined as follows:

$(definition\ \ (attributes\ \ in\ \ x)\ \ var\ \ (x)\ \ where\ \ (x\ \ is\ \ attribute)$
$\ then\ \ (attributes\ \ in\ \ x) :: atm)$;
$(interpretation\ \ (attributes\ \ in\ \ x) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = (a_{rg.1}, ..., a_{rg.n_{t0}})$ for $x_0 = (a_{rg.1} : v_{l.1}, ..., a_{rg.n_{t0}} : v_{l.n_{t0}})$.

The element $(values\ in\ x)$ specifying the sequence of attribute values of the attribute element x is defined as follows:

$(definition\ \ (values\ \ in\ \ x)\ \ var\ \ (x)\ \ where\ \ (x\ \ is\ \ attribute)\ \ then\ \ (values\ \ in\ \ x) :: atm)$;
$(interpretation\ \ (values\ \ in\ \ x) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,
where $[f_n\ s_b] = (v_{l.1}, ..., v_{l.n_{t0}})$ for $x_0 = (a_{rg.1} : v_{l.1}, ..., a_{rg.n_{t0}} : v_{l.n_{t0}})$.

The element $(element\ in\ x)$ specifying the element of the sorted element x is defined as follows:

$(definition\ \ (element\ \ in\ \ x)\ \ var\ \ (x)\ \ then\ \ (if\ \ x\ \ matches\ \ y :: z\ \ var\ \ (y,\ \ z)\ \ then\ \ y :: q))$.

The element $(sort\ in\ x)$ specifying the sort of the sorted element x is defined as follows:

$(definition\ \ (sort\ \ in\ \ x)\ \ var\ \ (x)\ \ then\ \ (if\ \ x\ \ matches\ \ y :: z\ \ var\ \ (y,\ \ z)\ \ then\ \ z :: q))$.

The element $(attribute\ in\ x)$ specifying the attribute of the element update x is defined as follows:

$(definition\ \ (attribute\ \ in\ \ x)\ \ var\ \ (x)\ \ then\ \ (if\ \ x\ \ matches\ \ y : z\ \ var\ \ (y,\ \ z)\ \ then\ \ y :: q))$.

The element $(value\ in\ x)$ specifying the value of the element update x is defined as follows:

$(definition\ \ (value\ \ in\ \ x)\ \ var\ \ (x)\ \ then\ \ (if\ \ x\ \ matches\ \ y : z\ \ var\ \ (y,\ \ z)\ \ then\ \ z :: q))$.

15.5. Boolean operations

The element $true$ is defined as follows:

$(definition\ \ true\ \ then\ \ true :: q)$.

The element $(x\ and\ y)$ specifying the conjunction of x and y is defined as follows:

$(definition\ \ (x\ \ and\ \ y)\ \ var\ \ (x,\ \ y)\ \ then\ \ (if\ \ x\ \ then\ \ y\ \ else\ \ und))$.

The elements $(x\ o_p\ y)$, where $o_p \in \{or, =>, <=>\}$ specifying the disjunction, implication and equivalence of x and y are defined in the similar way.

The element $(x_1\ and\ x_2\ and\ ...\ and\ x_{n_t})$ specifying the conjunction of x_1, x_2, ..., x_{n_t} is defined as follows:

$(definition\ \ (x\ \ and\ \ y\ \ and\ \ z)\ \ var\ \ (x,\ \ y)\ \ seq\ \ (z)\ \ then\ \ ((x\ \ and\ \ y)\ \ and\ \ z)$.

The element $(x_1\ or\ x_2\ or\ ...\ or\ x_{n_t})$ specifying the disjunction of x_1, x_2, ..., x_{n_t} is defined in the similar way.

The element $(not\ x)$ specifying the negation of x is defined as follows:

$(definition\ \ (not\ \ x)\ \ var\ \ (x)\ \ then\ \ (if\ \ x\ \ then\ \ und\ \ else\ \ true))$.

15.6. Integers

The element i_{nt} is defined as follows:

$(definition\ \ x\ \ var\ \ (x)\ \ where\ \ (x\ \ is\ \ int)\ \ then\ \ x :: q) :: name :: ("@",\ \ int)$.

The definition satisfies the property: $("@", exception) \prec_{[\![o_{rd.intr}]\!]} ("@", int)$.

The element $(x\ +\ y)$ specifying the sum of x and y is defined as follows:

$(definition\ \ (x\ \ +\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)$

$where\ \ ((x :: *\ \ is\ \ int)\ \ and\ \ (y :: *\ \ is\ \ int))\ \ then\ \ (x :: *\ \ +\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ +\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$, where $[f_n\ s_b] = [x_0\ +\ y_0]$.

The elements $(x\ o_p\ y)$, where $o_p \in \{-, *, \}$, specifying the integer operations $-$ and $*$ are defined in the similar way.

The element $(x\ div\ y)$ specifying the quotient of x divided by y is defined as follows:

$(definition\ \ (x\ \ div\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)$

$\ where\ \ ((x :: *\ \ is\ \ int)\ \ and\ \ (y :: *\ \ is\ \ int)\ \ and\ \ (y :: *\ \ !=\ \ 0))$

$\ then\ \ (x :: *\ \ div\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ div\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [x_0\ div\ y_0]$.

The element $(x\ mod\ y)$ specifying the integer operation mod is defined in the similar way.

The element $(x\ <\ y)$ specifying that x is less than y is defined as follows:

$(definition\ \ (x\ \ <\ \ y)\ \ var\ \ (x,\ \ y)\ \ val\ \ (x,\ \ y)$

$\ where\ \ ((x :: *\ \ is\ \ int)\ \ and\ \ (y :: *\ \ is\ \ int))\ \ then\ \ (x :: *\ \ <\ \ y :: *) :: atm)$;

$(interpretation\ \ (x\ \ <\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [x_0\ <\ y_0]$.

The elements $(x\ o_p\ y)$, where $o_p \in \{<=, >, >=\}$, specifying the integer relations $\leq$, $>$ and $\geq$, are defined in the similar way.

15.7. Conceptuals operations

The element $(x\ in\ y)$ specifying the value of the conceptual x in the state y is defined as follows:

$(definition\ \ (x\ \ in\ \ y)\ \ var\ \ (x,\ \ y)$

$\ where\ \ ((x\ \ is\ \ conceptual)\ \ and\ \ (z\ \ is\ \ state))\ \ then\ \ (x\ \ in\ \ y) :: atm)$;

$(interpretation\ \ (x\ \ in\ \ y) :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [y_0\ x_0]$.

The element $x :: state :: y$ specifying the value of the conceptual x in the substate with the name y of the current configuration is defined as follows:

$(definition\ \ x :: state :: y\ \ var\ \ (x,\ \ y)\ \ where\ \ (x\ \ is\ \ conceptual)$

$\ then\ \ (x\ \ in\ \ (conf :: q\ \ ..\ \ y))\ \ x :: state :: y :: atm)$;

$(in\ \ x :: state :: y :: atm\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where $(x_0 :: state :: y_0 :: atm, e_{l.*}\ \#\ c_{nf} \to_{f_n,s_b} e_{l.*}\ \#\ [[c_{nf}\ y_0]\ x_0]\ \#\ c_{nf}$.

The element c_{ncpl} is a shortcut for $c_{ncpl} :: ()$.

15.8. Countable concepts operations

A normal element $c_{ncp.c}$ is a countable concept in $[\![c_{nf}]\!]$ if $[[c_{nf}$ $countable{-}concept]$ $(0 : c_{ncp.c})] \in N_t$. Thus, the substate *countable−concept* specifies countable concepts. Let $C_{ncp.c}$ be a set of countable concepts. The element $[[c_{nf}$ $countable{-}concept]$ $(0 : c_{ncp.c})]$ is called an order in $[\![c_{ncp.c}, c_{nf}]\!]$. Let $O_{rd.cncp.c}$ be a set of orders of countable concepts. An element $n_t :: cc :: c_{ncp.c}$ is called an instance in $[\![c_{ncp.c}]\!]$. An element $n_t :: cc :: c_{ncp.c}$ is an instance in $[\![c_{ncp.c}, c_{nf}]\!]$ if $n_t \leq o_{rd.cncp.c}[\![c_{ncp.c}, c_{nf}]\!]$.

The element (x *is countable−concept*) specifying that x is a countable concept is defined as follows:

$(definition\ \ (x\ \ is\ \ countable{-}concept)\ \ var\ \ (x)$

$\ \ then\ \ (let\ \ w\ \ be\ \ ((cnf\ \ ..\ \ countable{-}concept)\ \ ..\ \ (0 : x))\ \ in\ \ (w\ \ is\ \ int)).$

The element $n_t :: cc :: c_{ncp.c}$ is defined by the rule:

$(definition\ \ x :: cc :: y\ \ var\ \ (x,\ \ y)\ \ where\ \ ((x\ \ is\ \ int)\ \ and\ \ (y\ \ is\ \ countable{-}concept))$

$\ \ then\ \ x :: cc :: y :: q).$

15.9. Matching operations

The conditional pattern matching element e_l of the form $(if\ x\ matches\ y\ var\ z\ seq\ u\ then\ v\ else\ w)$, where (y, z, u) is a pattern specification, is defined as follows:

$(definition\ \ (if\ \ x\ \ matches\ \ y\ \ var\ \ z\ \ seq\ \ u\ \ then\ \ v\ \ else\ \ w)\ \ var\ \ (x,\ \ y,\ \ z,\ \ u,\ \ v,\ \ w)$

$\ \ where\ \ ((z\ \ is\ \ sequence)\ \ and\ \ (u\ \ is\ \ sequence)\ \ and\ \ (z\ \ includes :: set\ \ u))$

$\ \ then\ \ (if\ \ x\ \ matches\ \ y\ \ var\ \ z\ \ seq\ \ u\ \ then\ \ v\ \ else\ \ w) :: atm);$

$(interpretation\ \ (if\ \ x\ \ matches\ \ y\ \ var\ \ z\ \ seq\ \ u\ \ then\ \ v\ \ else\ \ w) :: atm$

$\ \ var\ \ (x,\ \ y,\ \ z,\ \ u,\ \ v,\ \ w)\ \ then\ \ f_n),$

where $[value\ (if\ x_0\ matches\ y_0\ var\ z_0\ seq\ u_0\ then\ v_0\ else\ w_0) :: atm\ s_b\ c_{nf}],\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b}$ $[if\ [x_0$ is an instance in $[\![(y_0, z_0, u_0), m_t,\ s_{b.1}]\!]$ for some $s_{b.1}]$ $then\ [subst\ s_{b.1} \cup (conf :: in : c_{nf})$ $v_0]$ $else\ [subst\ (conf :: in : c_{nf})\ w_0],\ e_{l.*}\ \#\ c_{nf}$. The objects x, y, z, u, v and w are called a matched element, pattern, variable specification, sequence variable specification, *then*-branch and *else*-branch in $[\![e_l]\!]$. The elements of z are called pattern variables in $[\![e_l]\!]$. The element e_l executes the instance of the *then*-branch v in $[\![s_{b.1}]\!]$ if x is an instance in $[\![y, s_{b.1}]\!]$. Otherwise, the element e_l executes the *else*-branch w.

Let $\{v_{r.*}\}$, $\{v_{r.s.*}\}$, $\{v_{r.*.1}\}$ and $\{v_{r.*.2}\}$ are pairwise disjoint, and $\{v_{r.*.3}\} \subseteq \{v_{r.*}\} \cup \{v_{r.*.1}\} \cup \{v_{r.*.2}\}$. The form $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.1})\ und\ (v_{r.*.2})\ val\ (v_{r.*.3})\ where$

c_{nd} *then* $e_{l.1}$ *else* $e_{l.2}$) is defined as follows:

- (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *val* $(v_{r.*.3})$ *where* c_{nd} *then* $e_{l.1}$ *else* $e_{l.2}$) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *abn* $(v_{r.*.1})$ *und* $(v_{r.*.2})$ *val* $(v_{r.*.3})$ *then* (*if* c_{nd} *then* $e_{l.1}$ *else* $e_{l.2}$ :: (*nosubstexcept conf* :: *in*)) *else* $e_{l.2}$);
- (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *val* $(v_{r.*.3},\ v_r)$ *then* $e_{l.1}$ *else* $e_{l.2}$) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *val* $(v_{r.*.3})$ *then* (*let* w *be* v_r *in* [*subst* $(v_r :: * : w)$ $e_{l.1}$]) *else* $e_{l.2}$), where w is a new element that does not occur in this definition;
- (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *val* () *then* $e_{l.1}$ *else* $e_{l.2}$) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *then* $e_{l.1}$ *else* $e_{l.2}$);
- (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1},\ v_r)$ *abn* $(v_{r.*.2})$ *then* b_d) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *then* (*if* (v_r *is undefined*) *then und else* $e_{l.1}$) *else* $e_{l.2}$);
- (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* () *abn* $(v_{r.*.2})$ *then* $e_{l.1}$ *else* $e_{l.2}$) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *abn* $(v_{r.*.2})$ *then* $e_{l.1}$ *else* $e_{l.2}$);
- (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *abn* $(v_{r.*.2},\ v_r)$ *then* $e_{l.1}$ *else* $e_{l.2}$) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *abn* $(v_{r.*.2})$ *then* (*if* (v_r *is abnormal*) *then* v_r *else* $e_{l.1}$) *else* $e_{l.2}$);
- (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *abn* () *then* $e_{l.1}$ *else* $e_{l.2}$) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *then* $e_{l.1}$ *else* $e_{l.2}$).

The element c_{nd} specifies the restriction on the values of the pattern variables. The undefined value is propagated through the variables of $v_{r.*.1}$. Abnormal values are propagated through the variables of $v_{r.*.2}$. The special element $v_r :: *$ references to the value of element associated with the pattern variable v_r. A pattern variable is evaluated if the element associated with it is evaluated. Thus, the sequence $v_{r.*.3}$ contains evaluated pattern variables. A pattern variable is quoted if the element associated with it is not evaluated.

The objects *var* $(v_{r.*})$, *seq* $(v_{r.s.*})$, *und* $(v_{r.*.1})$, *abn* $(v_{r.*.2})$, *val* $(v_{r.*.3})$, *where* c_{nd} and *else* $e_{l.2}$ in this form can be omitted. The omitted objects correspond to *var* (), *seq* (), *und* (), *abn* (), *val* (), *where true* and *else skip*, respectively.

The form (e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *val* $(v_{r.*.3})$ *where* c_{nd}) is a shortcut for (*if* e_l *matches* p_t *var* $(v_{r.*})$ *seq* $(v_{r.s.*})$ *und* $(v_{r.*.1})$ *abn* $(v_{r.*.2})$ *val* $(v_{r.*.3})$ *where* c_{nd}

then true else und). The objects *var* ($v_{r.*}$), *seq* ($v_{r.s.*}$), *und* ($v_{r.*.1}$), *abn* ($v_{r.*.2}$), *val* ($v_{r.*.3}$) and *where* c_{nd} in this form can be omitted. The omitted objects correspond to *var* (), *seq* (), *und* (), *abn* (), *val* () and *where true*, respectively.

15.10. Configurations operations

The element $conf :: cur$ specifying the current configuration is defined as follows:
$(definition\ \ conf :: cur\ \ then\ \ conf :: cur :: atm)$;
$(interpretation\ \ conf :: cur :: atm\ \ then\ \ f_n)$,
where $[f_n\ s_b] = c_{nf}$.

16. Justification of requirements for conceptual configuration systems

In this section, we establish that CCSs meet the requirements stated in section 1:

1. *The formalism must model the conceptual structure of states and state objects of the IQS.* The conceptual structure of states of the IQS is modelled by elements (attributes, concepts, individuals) and, in more detail, usual and generic conceptuals of conceptual configurations.
2. *The formalism must model the content of the conceptual structure.* The content of the conceptual structure is modelled by conceptual configurations.
3. *The formalism must model information queries, information query objects, answers and answer objects of the IQS.* Information queries, information query objects, answers and answer objects of the IQS are modelled by elements of the CCS.
4. *The formalism must model the interpretation function of the IQS.* The interpretation function of the IQS is modelled by the interpretation function *value* of the CCS.
5. *The formalism must be quite universal to model typical ontological elements.* Models of typical ontological elements is presented in sections 6-10, 12 and 13.
6. *The formalism must provide a quite complete classification of ontological elements, including the determination of their new kinds and subkinds with arbitrary conceptual granularity.* Classification of ontological elements based on the two-level scheme is presented in section 11. The arbitrary conceptual granularity is provided by conceptuals.
7. *The model of the interpretation function must be extensible.* The model of the interpretation function of the IQS is extended by addition of element definitions.

8. *The formalism must have language support. The language associated with the formalism must define syntactic representations of models of states, state objects, queries, query objects, answers and answer objects and includes the set of predefined basic query models.* The CCSL language associated with CCSs defines syntactic representations of models of states, state objects, queries, query objects, answers and answer objects and includes the set of predefined basic query models.

Thus, the requirements are met for CCSs.

17. Comparison of conceptual configuration systems with abstract state machines

Abstract state machines (ASMs) [3, 4] are the special kind of transition systems in which states are algebraic systems. They are a formalism for abstract unified modelling of computer systems. We compare CCSs with ASMs, based on the requirements stated in section 1:

1. *The formalism models the conceptual structure of states of the IQS.* The conceptual structure of states of the IQS is modelled by the appropriate choice of symbols of the signature of an algebraic system. Thus, both ASMs and CCSs model the conceptual structure of states of the IQS, but CCSs make it by more natural ontological way.
2. *The formalism models the content of the conceptual structure.* The content of the conceptual structure is modelled by the interpretation of signature symbols in a particular state.
3. *The formalism must model information queries, information query objects, answers and answer objects of the IQS.* Information queries and information query objects of the IQS are modelled by terms, and answers and answer objects of the IQS are modelled by values of the terms. The element-based representation in CCSs is reacher than the term-based representation in ASMs.
4. *The formalism must model the interpretation function of the IQS.* The interpretation function of the IQS are modelled by the term interpretation function that is simpler than the element interpretation function in CCSs.
5. *The formalism is quite universal to model typical ontological elements.* In contrast to CCSs, typical ontological elements are not naturally modelled by ASMs.
6. *The formalism provides a quite complete classification of ontological elements, including the determination of their new kinds and subkinds with arbitrary conceptual granularity.*

In contrast to CCSs, ASMs do not allow to classify naturally ontological elements and define their new kinds and subkinds with arbitrary conceptual granularity.

7. *The model of the interpretation function must be extensible.* The model of the interpretation function can not be directly extended in ASMs.
8. *The formalism must have language support.* There are two languages AsmL [5] and XasM [6] for specification of ASMs. The AsmL language is more expressive than CTSL. It is fully integrated into the Microsoft .NET environment and uses XML and Word for literate specifications. XASM realizes a component-based modularization concept based on the notion of external functions as defined in ASMs.

18. Conclusion

In the paper two formalisms (information query systems and conceptual configuration systems) for abstract unified modelling of the artifacts of the conceptual design of closed information systems have been proposed. The basic definitions of the theory of CCSs have been given. The classification and interpretation of elements of such conceptual structures of CCSs as conceptuals, conceptual states, conceptual configurations, concepts and attributes has been presented. The classification of ontological elements based on these conceptual structures has been described. A language of CCSs has been defined.

The feature of conceptual design for closed information systems based on conceptual configuration systems is that they allow us to describe the conceptual structure of states of the information systems in detail. We plan to extend this formalism to describe both states and state transitions in detail and apply it for conceptual design of wider class of information systems.

References

1. Sokolowski J., Banks C. Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains. Wiley, 2010.
2. Chen P. Entity-relationship modeling: historical events, future trends, and lessons learned // Software pioneers. Springer-Verlag New York, 2002. P. 296-310.
3. Gurevich Y. Abstract state machines: An Overview of the Project // Foundations of Information and Knowledge Systems. Lect. Notes Comput. Sci. 2004. Vol. 2942. P. 6-13.
4. Gurevich Y. Evolving Algebras. Lipari Guide // Specification and Validation Methods. Oxford University Press, 1995. P. 9-36.

5. AsmL: Abstract State Machine Language. URL: https://www.microsoft.com/en-us/research/project/asml-abstract-state-machine-language/ (accessed: 27.12.2016).
6. XasM — An Extensible, Component-Based Abstract State Machines Language. URL: http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html (accessed: 27.12.2016).