УДК 004.8

Conceptual transition systems*

Anureev I.S. (Institute of Informatics Systems)

A new formalism for description of ontologies of systems and their changes – conceptual transition systems – is presented. The basic definitions of the theory of conceptual transition systems are given. These systems were demonstrated to allow to specify both typical and new kinds of ontological elements constituting ontologies. The classification of ontological elements based on such systems is described.

Keywords: transition systems, conceptual structures, ontologies, ontological elements, conceptual transition systems, conceptuals

1. Introduction

Development of formalisms, languages and tools for describing the conceptual structure of various systems is an important problem of the modern knowledge industry. Description of changes of the conceptual structure of the system when it functions is an another important problem.

Conceptual transition systems (CTSs) are a formalism of description (specification) of systems that solves these problems. This formalism is based on the following requirements:

1. It describes the conceptual structure of the specified system.
2. It describes the content of the conceptual structure of the specified system, i. e. it describes the specified system in the context of the conceptual structure.
3. It describes the change of the conceptual structure of the specified system.
4. It describes the change of the content of the conceptual structure of the specified system, i. e. it describes the change of the specified system in the context of the conceptual structure.
5. It is quite universal to specify typical ontological elements (concepts, attributes, concept instances, relations, relation instances, individuals, types, domains, and so on.).
6. It provides a quite complete classification of ontological elements, including the determination of their new kinds and subkinds.
7. It is based on the conception 'state – transition' of the usual transition systems, keeping their simplicity and universality and adding a conceptual 'filling'. This requirement is

*Partially supported by RFBR under grants 15-01-05974 and 15-07-04144.

important since the simplicity of determination of transition systems makes them an universal formalism to describe the behavior of various systems (algorithms, programs, software models, computer systems, and so on.).

8. It supports reflection of any order, i. e. allows to specify: the system (reflection of the order 0), the specification of the system (reflection of the order 1), the specification of the specification of the system (reflection of the order 2) and so on. Specifications of the higher order (with reflection of the higher order) impose restrictions on the specifications of the lower order (with reflection of the lower order).

To our knowledge, there is no formalism that meets all the above requirements. Comparison of CTSs with the formalisms which partially meet these requirements is given in section 9.

The paper has the following structure. The preliminary concepts and notation are given in section 2. The basic definitions of the theory of CTSs are given in section 3. The classification of elements of conceptual states of CTSs such that concepts, attributes and individuals is considered in section 4. The classification of conceptuals (which can be considered as 'atoms' of conceptual states) and their associated ontological elements is presented in section 5. The ontological elements that are not directly represented in terms of elements and conceptuals of states are modelled in these terms in section 6. A generic conceptual describing sets of conceptuals matching a pattern is defined in section 7. We establish that CTSs meet the above requirements in section 8. CTSs are compared with the related formalisms in section 9.

2. Preliminaries

Let $bool = \{true, false\}$; int, nat and $nat0$ denote the sets of integers, natural numbers and natural numbers with zero, respectively; obj, fun, set, lab, arg, and val denote sets of objects, functions, sets, labels, function arguments and function values, respectively.

The names of the variables which take the values from the set with the name aw, where a is a symbol, and w is a word, are denoted by $\dot{a}w$, $\dot{a}w_1$, $\dot{a}w'$ and so forth. For example, $\dot{s}et$, $\dot{s}et_1$, $\dot{s}et'$ are the names of the variables which take the values from the set set. Depending on the context, the name of a variable is interpreted as either the variable, or the value of the variable.

Let $sup(\dot{f}un)$ and ω denote the support of $\dot{f}un$ and the indeterminate value of $\dot{f}un$, respectively.

Let $\dot{f}un(\dot{a}rg_1 \leftarrow \dot{v}al_1, \ldots, \dot{a}rg_{\dot{n}at} \leftarrow \dot{v}al_{\dot{n}at})$ denote the function $\dot{f}un'$ such that $\dot{f}un'(\dot{a}rg) = \dot{f}un(\dot{a}rg)$, if $\dot{a}rg$ is distinct from $\dot{a}rg_1$, ..., $\dot{a}rg_{\dot{n}at}$, and $\dot{f}un'(\dot{a}rg_{\dot{n}at'}) = \dot{v}al_{\dot{n}at'}$, if $1 \leq \dot{n}at' \leq \dot{n}at$.

Let $\{\dot{a}rg_1{:}\dot{v}al_1, \ldots, \dot{a}rg_{\dot{n}at}{:}\dot{v}al_{\dot{n}at}\}$ denote the function $\dot{f}un$ such that $sup(\dot{f}un) = \{\dot{a}rg_1, \ldots, \dot{a}rg_{\dot{n}at}\}$, and $\dot{f}un(\dot{a}rg_1) = \dot{v}al_1$, ..., $\dot{f}un(\dot{a}rg_{\dot{n}at}) = \dot{v}al_{\dot{n}at}$. The arguments $\dot{a}rg_1$, ..., $\dot{a}rg_{\dot{n}at}$ are pairwise distinct.

The terms used in the paper are context-dependent. Contexts have the form $[\![\dot{o}bj_1, \ldots, \dot{o}bj_{\dot{n}at}]\!]$, where the embedded contexts $\dot{o}bj_1$, ..., $\dot{o}bj_{\dot{n}at}$ have the form: $\dot{l}ab{:}\dot{o}bj$, $\dot{l}ab{:}$ or $\dot{o}bj$.

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 $\dot{o}bj[\![\dot{o}bj_1, \ldots, \dot{o}bj_{\dot{n}at}]\!]$ denote the object $\dot{o}bj$ in the context $[\![\dot{o}bj_1, \ldots, \dot{o}bj_{\dot{n}at}]\!]$.

3. Basic definitions of the theory of conceptual transition systems

Let *cts* and *sys* be sets of CTSs and systems specified by these CTSs, respectively. Let $\dot{c}ts$ specifies $\dot{s}ys$.

3.1. The example of the specified system

Let $geoSys \in sys$ be a system which is specified by $\dot{c}ts$ and is a through illustrative example of this paper.

The conceptual structure of *geoSys* includes the following entities:

- the kinds of geometric spaces (Euclidean, Riemannian, Lobachevskian and so on) specified by the labels *Euclidean*, *Riemannian*, *Lobachevskian* and so on;
- the kinds of geometric figures (triangles, rectangles, cubes and so on) specified by the concepts *triangle*, *rectangle*, *cube* and so on;
- geometric elements (certain geometric figures in a certain space) specified by individuals. Let *geoEle* be a set of geometric elements;
- the numerical characteristics of geometric figures (length, area, volume and so on) specified by the attributes *length*, *area*, *volume* and so on;
- the units (of measurement) of the numerical characteristics (inches, centimeters, metres and so on) specified by the labels *inch*, *centimeter*, *metre* and so on;
- the numeral systems for representing the values of the numerical characteristics (binary, octal, decimal and so on) specified by the natural numbers *2*, *8*, *10* and so on;
- the dimensions of geometric spaces specified by the natural numbers *1*, *2*, *3* and so on.

The change of the system *geoSys* can, for example, include various geometric transformations such that parallel a shift, rotation, homothety and so on.

3.2. Conceptual transition systems

A transition system $\dot{cts}$ = *(sta, TraRel)* is called a conceptual transition system in $[\![ato]\!]$, if *ato* is a set of atoms in $[\![\dot{cts}]\!]$, *sta* is a set of conceptual states in $[\![\dot{cts}]\!]$, and *traRel* $\in$ *tra* $\to$ *bool* is a transition relation in $[\![\dot{cts}]\!]$, where *tra* = *sta* $\times$ *sta* is a set of transitions in $[\![\dot{cts}]\!]$. The system $\dot{cts}$ executes a transition $\dot{tra}$, if *traRel(*$\dot{tra}$*)*. The notion of conceptual state is based on notions of state, element and conceptual which are defined below.

A set *ato* is called a set of atoms in $[\![\dot{cts}]\!]$, if $\omega \notin ato$, *int* $\subseteq$ *ato*, and *true, false* $\in$ *ato*. Atoms in $[\![\dot{cts}]\!]$ are elementary 'bricks' of $\dot{cts}$. They are used to define elements, conceptuals and states in $[\![\dot{cts}]\!]$.

Elements in $[\![\dot{cts}]\!]$ are basic structures of $\dot{cts}$. In particular, they specify elements of $\dot{sys}$. Let *ele* be a set of elements in $[\![\dot{cts}]\!]$. An object $\dot{obj}$ is called an element in $[\![\dot{cts}]\!]$, if the following properties hold:

1. $\dot{obj}$ is an atom in $[\![\dot{cts}]\!]$, or
2. $\dot{obj}$ has the form $\{\dot{lab}_1{:}\dot{ele}_1, \ldots, \dot{lab}_{\dot{nat}}{:}\dot{ele}_{\dot{nat}}\}$, where the labels $\dot{lab}_1$, ..., $\dot{lab}_{\dot{nat}}$ are pairwise distinct, or
3. $\dot{obj}$ has the form $(\dot{ele}_1, \ldots, \dot{ele}_{\dot{nat}0})$, or
4. $\dot{obj}$ has the form $\{\dot{ele}_1, \ldots, \dot{ele}_{\dot{nat}0}\}$.

Elements of the forms 2, 3, and 4 are called labelled, ordered and unordered (element) structures, respectively. Let *labStr*, *ordStr*, *unoStr* and *eleStr* = *labStr* $\cup$ *ordStr* $\cup$ *unoStr* be sets of labelled structures, ordered structures, unordered structures and element structures, respectively. The elements *()* and *{}* are called empty structures (the empty ordered structure and the empty unordered structure, respectively).

Let $1 \le \dot{nat}' \le \dot{nat}$. Let *ėleStr(*$\dot{nat}'$*)* and *ėleStr(%*$lab_{\dot{nat}'}$*)* denote $\dot{ele}_{\dot{nat}'}$, if *ėleStr* has the form $(\dot{ele}_1, \ldots, \dot{ele}_{\dot{nat}})$ and $\{lab_1{:}\dot{ele}_1, \ldots, lab_{\dot{nat}}{:}\dot{ele}_{\dot{nat}}\}$, respectively.

The function *len* $\in$ *ele* $\to$ *nat0* is called a length in $[\![ele{:}]\!]$, if *len(*$\dot{ato}$*)* = *0*, and *len(ėleStr)* is the number of elements in *ėleStr*.

The equality operation = on elements is defined as follows: $\dot{ele} = \dot{ele}'$ if and only if *len(*$\dot{ele}$*)* = *len(*$\dot{ele}'$*)* = $\dot{nat}$, and

- $\dot{ele}$ and $\dot{ele}'$ are equal atoms, or

- $\dot{e}le = (\dot{e}le_1, \ldots, \dot{e}le_{\dot{n}at0})$, $\dot{e}le' = (\dot{e}le'_1, \ldots, \dot{e}le'_{\dot{n}at0})$, $\dot{e}le_1 = \dot{e}le'_1$, ..., $\dot{e}le_{\dot{n}at0} = \dot{e}le'_{\dot{n}at0}$, or
- $\dot{e}le = \{\dot{e}le_1, \ldots, \dot{e}le_{\dot{n}at0}\}$, $\dot{e}le' = \{\dot{e}le'_1, \ldots, \dot{e}le'_{\dot{n}at0}\}$, $\dot{e}le_1 = \dot{e}le'_1$, ..., $\dot{e}le_{\dot{n}at0} = \dot{e}le'_{\dot{n}at0}$, or
- $\dot{e}le = \{\dot{l}ab_1{:}\dot{e}le_1, \ldots, \dot{l}ab_{\dot{n}at}{:}\dot{e}le_{\dot{n}at}\}$, $\dot{e}le' = \{\dot{l}ab'_1{:}\dot{e}le'_1, \ldots, \dot{l}ab'_{\dot{n}at}{:}\dot{e}le'_{\dot{n}at}\}$, $\dot{l}ab_1 = \dot{l}ab'_1$, ..., $\dot{l}ab_{\dot{n}at} = \dot{l}ab'_{\dot{n}at}$, $\dot{e}le_1 = \dot{e}le'_1$, ..., $\dot{e}le_{\dot{n}at} = \dot{e}le'_{\dot{n}at}$.

Conceptuals in $[\![\dot{c}ts]\!]$ are the special kind of elements which specify ontological elements of $\dot{s}ys$. An element $\dot{l}abStr$ is called a conceptual in $[\![\dot{c}ts]\!]$, if all its labels are integers. Let *con* be a set of conceptuals.

Example. Let $\dot{c}on = (-3{:}10, -2{:}inch, -1{:}area, 0{:}\dot{g}eoEle, 1{:}triangle, 2{:}Euclidean, 3{:}2)$. Then the following properties hold:

- $\dot{c}on$ is a conceptual in $\dot{c}ts$;
- $\dot{c}on$ specifies the area (the label -1) of the triangle (the label *1*) $\dot{g}eoEle$ (the label *0*) in three-dimensional (the label *3*) Euclidean (the label *2*) space, measured in inches (the label -2) in the decimal system (the label -3).

•

A function $\dot{f}un \in con \to ele$ is called a conceptual state in $[\![\dot{c}ts]\!]$. A state in $[\![\dot{c}ts]\!]$ is called conceptual because it specifies the conceptual structure of the system $\dot{s}ys$, associating conceptuals with their values.

A function $sem \in con \times sta \to ele$ is called a semantics in $[\![con{:}]\!]$, if $sem(\dot{c}on, \dot{s}ta) = \dot{s}ta(\dot{c}on)$. The element $sem(\dot{c}on, \dot{s}ta)$ is called the value in $[\![\dot{c}on, \dot{s}ta]\!]$.

Example. Let $\dot{c}on = (-3{:}10, -2{:}inch, -1{:}area, 0{:}\dot{g}eoEle, 1{:}triangle, 2{:}Euclidean, 3{:}2)$, and $\dot{s}ta(\dot{c}on) = 3$. Then the following properties hold:

- $sem(\dot{c}on, \dot{s}ta) = 3$;
- *3* is the value in $[\![\dot{c}on, \dot{s}ta]\!]$;
- the area of the triangle $\dot{g}eoEle$ in two-dimensional Euclidean space is equal to *3* inches in the decimal system in $[\![\dot{s}ta]\!]$.

•

3.3. Structure of conceptuals

An element $\dot{e}le$ is called an element in $[\![\dot{c}on, \dot{i}nt]\!]$, if $\dot{e}le{=}\ \dot{c}on(\dot{i}nt)$. A number $\dot{i}nt$ is called an element order in $[\![\dot{c}on, \dot{e}le]\!]$, if $\dot{e}le{=}\ \dot{c}on(\dot{i}nt)$. Let *eleOrd* be a set of element orders.

Example. Let $\dot{c}on = (-3{:}10, -2{:}inch, -1{:}area, 0{:}\dot{g}eoEle, 1{:}triangle, 2{:}Euclidean, 3{:}2)$. Then the following properties hold:

- *10*, *inch*, *area*, *ġeoEle*, *triangle*, *Euclidean*, *2* are elements in $[\![\dot{con}]\!]$ in $[\![-3]\!]$, $[\![-2]\!]$, $[\![-1]\!]$, $[\![0]\!]$, $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$, respectively;
- -3, -2, -1, *0*, *1*, *2*, *3* are element orders in $[\![\dot{con}]\!]$ in $[\![10]\!]$, $[\![inch]\!]$, $[\![area]\!]$, $[\![\dot{geoEle}]\!]$, $[\![triangle]\!]$, $[\![Euclidean]\!]$, $[\![3]\!]$, respectively.

•

Proposition 1. The value ω is not an element in $[\![\dot{con}]\!]$.

Proof. This follows from the fact that ω is not an element in $[\![\dot{cts}]\!]$. □

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

Proof. This follows from the fact that *sup(ċon)* is finite, and ω is not an element in $[\![\dot{con}]\!]$. □

Proposition 3. If *ėle* is an element in $[\![\dot{con}]\!]$, then the number of element orders in $[\![\dot{con}, \dot{ele}]\!]$ is finite.

Proof. This follows from the fact that *sup(ċon)* is finite, and ω is not an element in $[\![\dot{con}]\!]$. □

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:10)*. Then the following properties hold:

- -3 and *3* are element orders in $[\![\dot{con}, 10]\!]$;
- if *ėle* is an element in $[\![\dot{con}]\!]$ which is distinct from *10*, then there is the unique element order in $[\![\dot{con}, \dot{ele}]\!]$.

•

Proposition 4. The number of element orders in $[\![\dot{con}]\!]$ is finite.

Proof. This follows from the fact that *sup(ċon)* is finite. □

An order $\dot{eleOrd}[\![\dot{con}, \dot{ele}]\!]$ is called a minimal element order in $[\![\dot{con}, \dot{ele}]\!]$, if $\dot{int}$ is not an element order in $[\![\dot{con}, \dot{ele}]\!]$ for each $\dot{int}$ such that $\dot{int} < \dot{eleOrd}$. An order $\dot{eleOrd}[\![\dot{con}]\!]$ is called a minimal element order in $[\![\dot{con}]\!]$, if $\dot{int}$ is not an element order in $[\![\dot{con}]\!]$ for each $\dot{int}$ such that $\dot{int} < \dot{eleOrd}$. An element *ėle* is called a minimal element in $[\![\dot{con}]\!]$, if there exists $\dot{eleOrd}[\![\dot{con}, \dot{ele}]\!]$ such that $\dot{eleOrd}$ is a minimal element order in $[\![\dot{con}]\!]$.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:10)*. Then the following properties hold:

- -3, -2, -1, *0*, *1*, *2*, *3* are element orders in $[\![\dot{con}]\!]$ in $[\![10]\!]$, $[\![inch]\!]$, $[\![area]\!]$, $[\![\dot{geoEle}]\!]$, $[\![triangle]\!]$, $[\![Euclidean]\!]$, $[\![10]\!]$, respectively;
- -3 is a minimal element order in $[\![\dot{con}]\!]$;

- *10* is a minimal element in ⟦*ċon*⟧.

•

An order *ėleOrd*⟦*ċon, ėle*⟧ is called a maximal element order in ⟦*ċon, ėle*⟧, if $\dot{int}$ is not an element order in ⟦*ċon, ėle*⟧ for each $\dot{int}$ such that *ėleOrd* < $\dot{int}$. An order *ėleOrd*⟦*ċon*⟧ is called a maximal element order in ⟦*ċon*⟧, if $\dot{int}$ is not an element order in ⟦*ċon*⟧ for each $\dot{int}$ such that *ėleOrd* < $\dot{int}$. An element *ėle* is called a maximal element in ⟦*ċon*⟧, if there exists *ėleOrd*⟦*ċon, ėle*⟧ such that *ėleOrd* is a maximal element order in ⟦*ċon*⟧.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:10)*. Then the following properties hold:

- *−3*, *−2*, *−1*, *0*, *1*, *2*, *3* are element orders in ⟦*ċon*⟧ in ⟦*10*⟧, ⟦*inch*⟧, ⟦*area*⟧, ⟦*ġeoEle*⟧, ⟦*triangle*⟧, ⟦*Euclidean*⟧, *10*, respectively;
- *3* is a maximal element order in ⟦*ċon*⟧;
- *10* is a maximal element in ⟦*ċon*⟧;
- *10* is both minimal and maximal element in ⟦*ċon*⟧.

•

An element *ėle* is called a null element in ⟦*ċon*⟧, if *ėle* is an element in ⟦*ċon, 0*⟧.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*. Then *ġeoEle* is a null element in ⟦*ċon*⟧. •

3.4. Conceptuals and elements of states

A conceptual *ċon* is called a conceptual in ⟦*ṡta*⟧, if *sem(ċon, ṡta)* ≠ *ω*.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta(ċon) = 3*. Then *ċon* is a conceptual in ⟦*ṡta*⟧. •

An element *ėle* is called an element in ⟦*ṡta, $\dot{int}$, ċon*⟦*ṡta*⟧⟧, if *ėle* is an element in ⟦*ċon, $\dot{int}$*⟧. The number $\dot{int}$ is called an order in ⟦*ėle, ṡta, ċon*⟧. The conceptual *ċon* is called a concretization conceptual in ⟦*ėle, ṡta, $\dot{int}$*⟧.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta(ċon) = 3*. Then the following properties hold:

- *10*, *inch*, *area*, *ġeoEle*, *trianle*, *Euclidean*, *3* are elements in ⟦*ṡta*⟧ in ⟦*−3*⟧, ⟦*−2*⟧, ⟦*−1*⟧, ⟦*0*⟧, ⟦*1*⟧, ⟦*2*⟧, ⟦*3*⟧ in ⟦*ċon*⟧, respectively;
- *−3*, *−2*, *−1*, *0*, *1*, *2*, *3* are orders in ⟦*10*⟧, ⟦*inch*⟧, ⟦*area*⟧, ⟦*ġeoEle*⟧, ⟦*triangle*⟧, ⟦*Euclidean*⟧, ⟦*2*⟧ in ⟦*ṡta*⟧ in ⟦*ċon*⟧, respectively;

- $\dot{con}$ is a concretization conceptual in $[\![10]\!]$, $[\![inch]\!]$, $[\![area]\!]$, $[\![\dot{geoEle}]\!]$, $[\![triangle]\!]$, $[\![Euclidean]\!]$, $[\![2]\!]$ in $[\![\dot{sta}]\!]$ in $[\![-3]\!]$, $[\![-2]\!]$, $[\![-1]\!]$, $[\![0]\!]$, $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$, respectively.

•

Proposition 5. For all $\dot{ele}$ and $\dot{int}$ there exist $\dot{sta}$ and $\dot{con}[\![\dot{sta}]\!]$ such that $\dot{ele}$ is an element in $[\![\dot{sta}, \dot{int}, \dot{con}]\!]$.

Proof. We define $\dot{sta}$ and $\dot{con}$ as follows: $\dot{con}(\dot{int}) = \dot{ele}$, and $\dot{sta}(\dot{con}) \neq \omega$. Then $\dot{ele}$ is an element in $[\![\dot{sta}, \dot{int}, \dot{con}]\!]$. □

4. Classification of elements of states

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

Individuals in $[\![\dot{sta}]\!]$ specify elements of $\dot{sys}$. An element $\dot{ele}[\![\dot{sta}]\!]$ is called an individual in $[\![\dot{sta}, \dot{con}[\![\dot{sta}]\!]]\!]$, if $\dot{ele}$ is an element in $[\![\dot{sta}, 0, \dot{con}]\!]$.

Example. Let $\dot{con} = (-3{:}10, -2{:}inch, -1{:}area, 0{:}\dot{geoEle}, 1{:}triangle, 2{:}Euclidean, 3{:}2)$, and $\dot{sta} = (\dot{con}{:}3)$. Then $\dot{geoEle}$ is an individual in $[\![\dot{sta}]\!]$ in $[\![\dot{con}]\!]$. •

Concepts in $[\![\dot{sta}]\!]$ generalizes the usual concepts of the ontology of *sys* which are interpreted as sets of elements of *sys*. An element $\dot{ele}[\![\dot{sta}]\!]$ is called a concept in $[\![\dot{sta}, \dot{nat}, \dot{con}[\![\dot{sta}]\!]]\!]$, if $\dot{ele}$ is an element in $[\![\dot{sta}, \dot{nat}, \dot{con}]\!]$. Let *conc* be a set of concepts.

Example. Let $\dot{con} = (-3{:}10, -2{:}inch, -1{:}area, 0{:}\dot{geoEle}, 1{:}triangle, 2{:}Euclidean, 3{:}2)$, and $\dot{sta} = (\dot{con}{:}3)$. Then *triangle*, *Euclidean*, *3* are concepts in $[\![\dot{sta}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$ in $[\![\dot{con}]\!]$, respectively. •

Attributes in $[\![\dot{sta}]\!]$ generalizes the usual attributes of the ontology of *sys* which are interpreted as characteristics of elements of *sys*. An element $\dot{ele}[\![\dot{sta}]\!]$ is called an attribute in $[\![\dot{sta}, \dot{nat}, \dot{con}[\![\dot{sta}]\!]]\!]$, if $\dot{ele}$ is an element in $[\![\dot{sta}, -\dot{nat}, \dot{con}]\!]$. A number $\dot{nat}$ is called an order in $[\![atr{:}\dot{ele}, \dot{sta}, \dot{con}]\!]$. The label *atr* is used to distinguish orders of concepts from orders of attributes, since the element $\dot{ele}$ can be both a concept and an attribute in $[\![\dot{sta}, \dot{nat}, \dot{con}]\!]$. The conceptual $\dot{con}$ is called a concretization conceptual in $[\![atr{:}\dot{ele}, \dot{sta}, \dot{nat}]\!]$. Let *atr* be a set of attributes.

Example. Let $\dot{con} = (-3{:}10, -2{:}inch, -1{:}area, 0{:}\dot{geoEle}, 1{:}triangle, 2{:}Euclidean, 3{:}2)$, and $\dot{sta} = (\dot{con}{:}3)$. Then the following properties hold:

- *area*, *inch*, *10* are attributes in $[\![\dot{sta}]\!]$ in $[\![1]\!]$, $[\![2]\!]$, $[\![3]\!]$ in $[\![\dot{con}]\!]$, respectively;
- *1*, *2*, *3* are orders in $[\![atr{:}area]\!]$, $[\![atr{:}inch]\!]$, $[\![atr{:}10]\!]$ in $[\![\dot{sta}]\!]$ in $[\![\dot{con}]\!]$, respectively;
- $\dot{con}$ is a concretization conceptual in $[\![atr{:}area]\!]$, $[\![atr{:}inch]\!]$, $[\![atr{:}10]\!]$ in $[\![\dot{sta}]\!]$ in $[\![1]\!]$, $[\![2]\!]$,

$[\![3]\!]$, respectively.

•

Concepts and attributes are considered in detail below.

4.1. Concepts

The usual concepts of the ontology of *ṡys* which are interpreted as sets of elements of *ṡys* are specified by the special kind of concepts in $[\![\dot{s}ta]\!]$ – direct concepts in $[\![\dot{s}ta]\!]$. An element $\dot{e}le[\![\dot{s}ta]\!]$ is called a direct concept in $[\![\dot{s}ta,\ \dot{c}on[\![\dot{s}ta]\!]]\!]$, if $\dot{e}le$ is a concept in $[\![\dot{s}ta,\ 1,\ \dot{c}on]\!]$. Let *dirConc* be a set of direct concepts.

Example. Let $\dot{c}on$ = *(−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and $\dot{s}ta$ = *(ċon:3)*. Then *triangle* is a direct concept in $[\![\dot{s}ta]\!]$ in $[\![\dot{c}on]\!]$. It specifies the element *ġeoEle* as a triangle in $[\![\dot{s}ta]\!]$. •

An element $\dot{e}le$ is called an element in $[\![conc{:}\dot{c}onc,\ \dot{s}ta,\ concOrd{:}\dot{n}at_1,\ eleOrd{:}\dot{n}at_2,\ \dot{c}on[\![\dot{s}ta]\!]]\!]$, if $\dot{c}onc$ is a concept in $[\![\dot{s}ta,\ \dot{n}at_1,\ \dot{c}on]\!]$, $\dot{e}le$ is an element in $[\![\dot{c}on,\ \dot{n}at_2]\!]$, and $\dot{n}at_2 < \dot{n}at_1$. Thus, elements of the concept $\dot{c}onc$ can be concepts of orders which are less than the order of $\dot{c}onc$, individuals and attributes of any orders.

Example. Let $\dot{c}on$ = *(−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and $\dot{s}ta$ = *(ċon:3)*. Then the following properties hold:

1. *10*, *inch*, *area*, *ġeoEle* are elements in $[\![conc{:}triangle]\!]$ in $[\![\dot{s}ta]\!]$ in $[\![concOrd{:}1]\!]$ in $[\![eleOrd{:}\text{-}3]\!]$, $[\![eleOrd{:}\text{-}2]\!]$, $[\![eleOrd{:}-1]\!]$, $[\![eleOrd{:}0]\!]$ in $[\![\dot{c}on]\!]$, respectively. In particular, this means that the triangle *ġeoEle* has the area which is measured in inches represented in the decimal system in $[\![\dot{s}ta]\!]$;
2. *10*, *inch*, *area*, *ġeoEle*, *triangle* are elements in $[\![conc{:}Eucludian]\!]$ in $[\![\dot{s}ta]\!]$ in $[\![concOrd{:}2]\!]$ in $[\![eleOrd{:}\text{-}3]\!]$, $[\![eleOrd{:}\text{-}2]\!]$, $[\![eleOrd{:}-1]\!]$, $[\![eleOrd{:}0]\!]$, $[\![eleOrd{:}1]\!]$ in $[\![\dot{c}on]\!]$, respectively. In particular, this means that the triangle *ġeoEle* belongs to Euclidean space in $[\![\dot{s}ta]\!]$;
3. *10*, *inch*, *area*, *ġeoEle*, *triangle*, *Eucludian* are elements in $[\![conc{:}2]\!]$, in $[\![\dot{s}ta]\!]$ in $[\![concOrd{:}3]\!]$ in $[\![eleOrd{:}-3]\!]$, $[\![eleOrd{:}-2]\!]$, $[\![eleOrd{:}-1]\!]$, $[\![eleOrd{:}0]\!]$, $[\![eleOrd{:}1]\!]$, $[\![eleOrd{:}2]\!]$ in $[\![\dot{c}on]\!]$, respectively. In particular, this means that the triangle *ġeoEle* belongs to two-dimensional space in $[\![\dot{s}ta]\!]$.

•

Proposition 6. If $\dot{c}onc$ is a concept in $[\![\dot{s}ta]\!]$, and $\dot{e}le$ is an element in $[\![conc{:}\dot{c}onc,\ \dot{s}ta,\ concOrd{:}1]\!]$, then $\dot{e}le$ is either an individual in $[\![\dot{s}ta]\!]$, or $\dot{e}le$ is an attribute in $[\![\dot{s}ta]\!]$.

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

A set $\dot{set}$ is called the content in $[\![conc{:}\dot{conc},\ \dot{sta},\ concOrd{:}\dot{nat}_1,\ eleOrd{:}\dot{nat}_2,\ \dot{con}[\![\dot{sta}]\!]]\!]$, if $\dot{set}$ is a set of elements in $[\![conc{:}\dot{conc},\ \dot{sta},\ concOrd{:}\dot{nat}_1,\ eleOrd{:}\dot{nat}_2,\ \dot{con}]\!]$. The content of a concept describes its semantics.

Example. Let $\dot{con} = (-3{:}10,\ -2{:}inch,\ -1{:}area,\ 0{:}\dot{geoEle},\ 1{:}triangle,\ 2{:}Euclidean,\ 3{:}2)$, and $\dot{sta} = (\dot{con}{:}3)$. Then the following properties hold:

- $\{2,\ inch,\ area,\ \dot{geoEle}\}$ is the content in $[\![conc{:}triangle]\!]$ in $[\![\dot{sta}]\!]$;
- $\{2,\ inch,\ area,\ \dot{geoEle},\ triangle\}$ is the content in $[\![conc{:}Eucludian]\!]$ in $[\![\dot{sta}]\!]$;
- $\{2,\ inch,\ area,\ \dot{geoEle},\ triangle,\ Eucludian\}$ is the content in $[\![conc{:}2]\!]$ in $[\![\dot{sta}]\!]$.

•

Example. Let $\dot{con}_1 = (-3{:}10,\ -2{:}inch,\ -1{:}area,\ 0{:}\dot{geoEle}_1,\ 1{:}triangle,\ 2{:}Euclidean,\ 3{:}3)$, $\dot{con}_2 = (-3{:}10,\ -2{:}inch,\ -1{:}area,\ 0{:}\dot{geoEle}_2,\ 1{:}triangle,\ 2{:}Riemannian,\ 3{:}3)$, $\dot{con}_3 = (-3{:}10,\ -2{:}inch,\ -1{:}area,\ 0{:}\dot{geoEle}_1,\ 3{:}2)$, and $\dot{sta} = (\dot{con}_1{:}3,\ \dot{con}_2{:}4,\ \dot{con}_3{:}2)$. Then the following properties hold:

- $\{\dot{geoEle}_1,\ \dot{geoEle}_2\}$ is the content in $[\![conc{:}triangle,\ \dot{sta},\ concOrd{:}1,\ eleOrd{:}0]\!]$. This means that the individuals $\dot{geoEle}_1$ and $\dot{geoEle}_2$ are triangles in $[\![\dot{sta}]\!]$;
- $\{triangle\}$ is the content in $[\![conc{:}Eucludian]\!]$, $[\![conc{:}Riemannian]\!]$ in $[\![\dot{sta},\ concOrd{:}2,\ eleOrd{:}1]\!]$, respectively. This means that Euclidean and Riemannian spaces can include triangles in $[\![\dot{sta}]\!]$;
- $\{Eucludian,\ Riemannian\}$ is the content in $[\![conc{:}3,\ \dot{sta},\ concOrd{:}3,\ eleOrd{:}2]\!]$. This means that three-dimensional space can be either Euclidean or Riemannian in $[\![\dot{sta}]\!]$;
- $\{\dot{geoEle}_1\}$ is the content in $[\![conc{:}2,\ \dot{sta},\ concOrd{:}3,\ eleOrd{:}0]\!]$. This means that two-dimensional space includes the individual $\dot{geoEle}_1$ in $[\![\dot{sta}]\!]$.

•

An element $\dot{ele}$ is called an element in $[\![conc{:}\dot{conc},\ \dot{sta},\ concOrd{:}\dot{nat}_1,\ eleOrd{:}\dot{nat}_2,\ \dot{con}[\![\dot{sta}]\!],\ med{:}\dot{nat}0]\!]$, if $\dot{ele}$ is an element in $[\![\dot{conc},\ \dot{sta},\ concOrd{:}\dot{nat}_1,\ eleOrd{:}\dot{nat}_2,\ \dot{con}]\!]$, and $\dot{nat}0$ is the number of element orders $\dot{nat}$ in $[\![\dot{con}]\!]$ such that $\dot{nat}_2 < \dot{nat} < \dot{nat}_1$. The integer $\dot{nat}0$ is called a mediatorial decree in $[\![\dot{ele},\ \dot{conc},\ \dot{sta},\ concOrd{:}\dot{nat}_1,\ eleOrd{:}\dot{nat}_2,\ \dot{con}]\!]$. It specifies how many mediators separate $\dot{ele}$ from $\dot{conc}$ in $\dot{con}$. The element $\dot{ele}'$ is called a mediator in $[\![\dot{ele},\ \dot{conc},\ \dot{sta},\ concOrd{:}\dot{nat}_1,\ eleOrd{:}\dot{nat}_2,\ \dot{con}[\![\dot{sta}]\!]]\!]$, if $\dot{ele}'$ is an element in $[\![\dot{con},\ \dot{nat}]\!]$, and $\dot{nat}_2 < \dot{nat} < \dot{nat}_1$.

Example. Let $\dot{con} = (-3{:}10,\ -2{:}inch,\ -1{:}area,\ 0{:}\dot{geoEle},\ 1{:}triangle,\ 2{:}Euclidean,\ 3{:}2)$,

and *ṡta = (ċon:3)*. Then *ġeoEle* is an element in the following contexts:

- ⟦*conc:triangle, ṡta, concOrd:1, eleOrd:0, ċon*⟧ with the mediatorial decree *0* and without mediators;
- ⟦*conc:Euclidean, ṡta, concOrd:2, eleOrd:0, ċon*⟧ with the mediatorial decree *1* and the mediator *triangle*;
- ⟦*conc:2, ṡta, concOrd:3, eleOrd:0, ċon*⟧ with the mediatorial decree *2* and the mediators *triangle* and *Euclidean*.

•

An element *ėle* is called a direct element in ⟦*conc:ċonc, ṡta, concOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟦*ṡta*⟧⟧, if *ėle* is an element in ⟦*conc:ċonc, ṡta, concOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon, med:0*⟧.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta = (ċon:3)*. Then the following properties hold:

- *ġeoEle* is a direct element in ⟦*conc:triangle, ṡta, concOrd:1, eleOrd:0*⟧;
- *triangle* is a direct element in ⟦*conc:Eucludian, ṡta, concOrd:2, eleOrd:1*⟧;
- *Eucludian* is a direct element in ⟦*conc:2, ṡta, concOrd:3, eleOrd:2*⟧.

•

Example. Let *ċon*$_1$ *= (−3:10, −2:inch, −1:area, 0:ġeoEle*$_1$*, 1:triangle, 2:Euclidean, 3:3)*, *ċon*$_2$ *= (−3:10, −2:inch, −1:area, 0:ġeoEle*$_2$*, 1:triangle, 2:Riemannian, 3:3)*, *ċon*$_3$ *= (−3:10, −2:inch, −1:area, 0:ġeoEle*$_1$*, 3:2)*, and *ṡta = (ċon*$_1$*:3, ċon*$_2$*:4, ċon*$_3$*:2)*. Then the following properties hold:

1. *ġeoEle*$_1$ and *ġeoEle*$_2$ are direct elements in ⟦*conc:triangle, ṡta*⟧;
2. *triangle* is a direct element in ⟦*conc:Eucludian*⟧ and ⟦*conc:Riemannian*⟧ in ⟦*ṡta*⟧;
3. *Eucludian* and *Riemannian* are direct elements in ⟦*conc:3, ṡta*⟧;
4. *ġeoEle*$_1$ is a direct element in ⟦*conc:2, ṡta*⟧.

•

A set *ṡet* is called the direct content in ⟦*conc:ċonc, ṡta, concOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟦*ṡta*⟧⟧, if *ṡet* is a set of direct elements in ⟦*conc:ċonc, ṡta, concOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟧.

Example. Let *ċon*$_1$ *= (−3:10, −2:inch, −1:area, 0:ġeoEle*$_1$*, 1:triangle, 2:Euclidean, 3:3)*, *ċon*$_2$ *= (−3:10, −2:inch, −1:area, 0:ġeoEle*$_2$*, 1:triangle, 2:Riemannian, 3:3)*, *ċon*$_3$ *= (−3:10, −2:inch, −1:area, 0:ġeoEle*$_1$*, 3:2)*, and *ṡta = (ċon*$_1$*:3, ċon*$_2$*:4, ċon*$_3$*:2)*. Then the following properties hold:

- *{ġeoEle*$_1$*, ġeoEle*$_2$*}* is the direct content in ⟦*conc:triangle, ṡta*⟧;

- *{triangle}* is the direct content in $[\![$*conc:Eucludian*$]\!]$, $[\![$*conc:Riemannian*$]\!]$ in $[\![\dot{s}ta]\!]$, respectively;
- *{Eucludian, Riemannian}* is the direct content in $[\![$*conc:3,* $\dot{s}ta]\!]$;
- *{*$\dot{g}eoEle_1$*}* is the direct content in $[\![$*conc:2,* $\dot{s}ta]\!]$.

•

A set $\dot{s}et$ is called the content in $[\![$*conc:*$\dot{c}onc$*,* $\dot{s}ta$*, concOrd:*$\dot{n}at_1$*, eleOrd:*$\dot{n}at_2$*,* $\dot{c}on[\![\dot{s}ta]\!]$*, med:*$\dot{n}at_3]\!]$, if $\dot{s}et$ is a set of elements in $[\![$*conc:*$\dot{c}onc$*,* $\dot{s}ta$*, concOrd:*$\dot{n}at_1$*, eleOrd:*$\dot{n}at_2$*,* $\dot{c}on$*, med:*$\dot{n}at_3]\!]$.

Example. Let $\dot{c}on_1$ *= (−3:10, −2:inch, −1:area, 0:*$\dot{g}eoEle_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{c}on_2$ *= (−3:10, −2:inch, −1:area, 0:*$\dot{g}eoEle_2$*, 1:triangle, 2:Riemannian, 3:2)*, $\dot{c}on_3$ *= (−3:10, −2:inch, −1:perimeter, 0:*$\dot{g}eoEle_3$*, 2:Euclidean, 3:2)*, and $\dot{s}ta$ *= (*$\dot{c}on_1$*:3,* $\dot{c}on_2$*:4,* $\dot{c}on_3$*:2).* Then the following properties hold:

- *{*$\dot{g}eoEle_1$*,* $\dot{g}eoEle_2$*}* is the content in $[\![$*conc:2,* $\dot{s}ta$*, concOrd:3, eleOrd:0, med:2*$]\!]$;
- *{*$\dot{g}eoEle_3$*}* is the content in $[\![$*conc:2,* $\dot{s}ta$*, concOrd:3, eleOrd:0, med:1*$]\!]$;
- *{area}* is the content in $[\![$*conc:2,* $\dot{s}ta$*, concOrd:3, eleOrd:−1, med:3*$]\!]$;
- *{perimeter}* is the content in $[\![$*conc:2,* $\dot{s}ta$*, concOrd:3, eleOrd:−1, med:2*$]\!]$.

•

4.2. Classification and interpretation of concepts

Concepts are classified according to their orders.

A concept $\dot{c}onc$ in $[\![\dot{s}ta$*, concOrd:1*$]\!]$ specifies a usual concept of the ontology of $\dot{s}ys$. Elements in $[\![$*conc:*$\dot{c}onc$*,* $\dot{s}ta$*, concOrd:1)*$]\!]$ are attributes and individuals in $[\![\dot{s}ta]\!]$.

Example. Let $\dot{c}on_1$ *= (−3:10, −2:inch, −1:area, 0:*$\dot{g}eoEle_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{c}on_2$ *= (−3:2, −2:cm, −1:perimeter, 0:*$\dot{g}eoEle_2$*, 1:triangle, 2:Euclidean, 3:2)*, and $\dot{s}ta$ *= (*$\dot{c}on_1$*:3,* $\dot{c}on_2$*:4).* Then the following properties hold:

1. The direct concept *triangle* specifies triangles in $[\![\dot{s}ta]\!]$.
2. The individuals $\dot{g}eoEle_1$ and $\dot{g}eoEle_2$ are elements of the order *0* of the direct concept *triangle* in $[\![\dot{s}ta]\!]$. This means that $\dot{g}eoEle_1$ and $\dot{g}eoEle_2$ are triangles in $[\![\dot{s}ta]\!]$.
3. The attributes *area* and *perimeter* are elements of the order *−1* of the direct concept *triangle* in $[\![\dot{s}ta]\!]$. This means that triangles can have area and perimeter in $[\![\dot{s}ta]\!]$.
4. The attributes *inch* and *cm* are elements of the order *−2* of the direct concept *triangle* in $[\![\dot{s}ta]\!]$. This means that numerical characteristics of triangles can be measured in inches

and centimetres in $[\![\dot{s}ta]\!]$.

5. The attributes *10* and *2* are elements of the order -3 of the direct concept *triangle* in $[\![\dot{s}ta]\!]$. This means that the values of numerical characteristics of triangles can be represented in decimal and binary systems in $[\![\dot{s}ta]\!]$.

•

A concept $\dot{c}onc$ in $[\![\dot{s}ta,\ concOrd{:}2]\!]$ specifies a concept space of the ontology of $\dot{s}ys$. Elements in $[\![conc{:}\dot{c}onc,\ \dot{s}ta,\ concOrd{:}2]\!]$ are attributes, individuals and direct concepts in $[\![\dot{s}ta]\!]$.

Example. Let $\dot{c}on_1$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{g}eoEle_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{c}on_2$ = *(−3:2, −2:cm, −1:perimeter, 0:*$\dot{g}eoEle_2$*, 1:square, 2:Euclidean, 3:2)*, and $\dot{s}ta$ = *(*$\dot{c}on_1$*:3,* $\dot{c}on_2$*:4)*. Then the following properties hold:

1. The concept space *Euclidean* specifies Euclidean space in $[\![\dot{s}ta]\!]$.
2. The direct concepts *triangle* and *square* are elements of the order *1* of the concept space *Euclidean* in $[\![\dot{s}ta]\!]$. This means that triangles and squares can belong to Euclidean space in $[\![\dot{s}ta]\!]$.
3. The individuals $\dot{g}eoEle_1$ and $\dot{g}eoEle_2$ are elements of the order *0* of the concept space *Euclidean* in $[\![\dot{s}ta]\!]$. This means that the geometric elements $\dot{g}eoEle_1$ and $\dot{g}eoEle_2$ belong to Euclidean space in $[\![\dot{s}ta]\!]$.
4. The attributes *area* and *perimeter* are elements of the order -1 of the concept space *Euclidean* in $[\![\dot{s}ta]\!]$. This means that geometric elements from Euclidean space can have area and perimeter in $[\![\dot{s}ta]\!]$.
5. The attributes *inch* and *cm* are elements of the order -2 of the concept space *Euclidean* in $[\![\dot{s}ta]\!]$. This means that numerical characteristics of geometric elements from Euclidean space can be measured in inches and centimetres in $[\![\dot{s}ta]\!]$.
6. The attributes *10* and *2* are elements of the order -3 of the concept space *Euclidean* in $[\![\dot{s}ta]\!]$. This means that values of numerical characteristics of geometric elements from Euclidean space can be represented in decimal and binary systems in $[\![\dot{s}ta]\!]$.

•

A concept $\dot{c}onc$ in $[\![\dot{s}ta,\ concOrd{:}3]\!]$ specifies a space of concept spaces of the ontology of $\dot{s}ys$. Elements in *conc:*$\dot{c}onc$*, ṡta, concOrd:3* are attributes, individuals, direct concepts and concept spaces in $[\![\dot{s}ta]\!]$.

Example. Let $\dot{c}on_1$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{g}eoEle_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{c}on_2$ = *(−3:2, −2:cm, −1:perimeter, 0:*$\dot{g}eoEle_2$*, 1:square, 2:Riemannian, 3:2)*, and $\dot{s}ta$ =

(ċon$_1$:3, ċon$_2$:4). Then the following properties hold:

1. The concept space space *2* specifies two-dimensional space in ⟦*ṡta*⟧.
2. The concept spaces *Euclidean* and *Riemannian* are elements of the order *2* of the concept space space *2* in ⟦*ṡta*⟧. This means that Euclidean and Riemannian spaces can be two-dimensional in ⟦*ṡta*⟧.
3. The direct concepts *triangle* and *square* are elements of the order *1* of the concept space space *2* in ⟦*ṡta*⟧. This means that triangles and squares can belong to two-dimensional space in ⟦*ṡta*⟧.
4. The individuals *ġeoEle*$_1$ and *ġeoEle*$_2$ are elements of the order *0* of the concept space space *2* in ⟦*ṡta*⟧. This means that geometric elements *ġeoEle*$_1$ and *ġeoEle*$_2$ belong to two-dimensional space in ⟦*ṡta*⟧.
5. The attributes *area* and *perimeter* are elements of the order -1 of the concept space space *2* in ⟦*ṡta*⟧. This means that geometric elements from two-dimensional space can have area and perimeter in ⟦*ṡta*⟧.
6. The attributes *inch* and *cm* are elements of the order -2 of the concept space space *2* in ⟦*ṡta*⟧. This means that numerical characteristics of geometric elements from two-dimensional space can be measured in inches and centimetres in ⟦*ṡta*⟧.
7. The attributes *10* and *2* are elements of the order -3 of the concept space space *2* in ⟦*ṡta*⟧. This means that values of numerical characteristics of geometric elements from two-dimensional space can be represented in decimal and binary systems in ⟦*ṡta*⟧.

•

A concept *ċonc* in ⟦*ṡta, concOrd:i̇nt*⟧, where *i̇nt* > *3*, is classified and interpreted in the similar way (by the introduction of the space of concept space spaces and so on.).

4.3. The attributes

Attributes use the same terminology as concepts.

The usual attributes of the ontology of *ṡys* which are interpreted as characteristics of elements of *ṡys* are specified by the special kind of attributes in ⟦*ṡta*⟧ – direct attributes in ⟦*ṡta*⟧.

An element *ėle*⟦*ṡta*⟧ is called a direct attribute in ⟦*ṡta, ċon*⟦*ṡta*⟧⟧, if *ėle* is an attribute in ⟦*ṡta, concOrd:1, ċon*⟧.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta = (ċon:3)*. Then the following properties hold:

- *area* is a direct attribute in $[\![\dot{s}ta, \dot{c}on]\!]$. It specifies the individual *ġeoEle* as the element which has an area in $[\![\dot{s}ta]\!]$;
- *area* is a direct attribute in $[\![\dot{s}ta]\!]$. It specifies the set of elements which have an area in $[\![\dot{s}ta]\!]$.

•

An element *ėle* is called an element in $[\![att{:}\dot{a}tt,\ \dot{s}ta,\ attOrd{:}\dot{n}at_1,\ eleOrd{:}\dot{n}at_2,\ \dot{c}on[\![\dot{s}ta]\!]]\!]$, if *ȧtt* is an attribute in $[\![\dot{s}ta,\ \dot{n}at_1,\ \dot{c}on]\!]$, *ėle* is an element in $[\![\dot{c}on,\ \dot{n}at_2]\!]$, and $-\dot{n}at_1 < \dot{n}at_2$. Thus, elements of the attribute *ȧtt* can be attributes of orders which is less than the order of *ȧtt*, individuals and concepts of all orders.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta = (ċon:3)*. Then the following properties hold:

1. *ġeoEle*, *triangle*, *Euclidean* and *2* are elements in $[\![att{:}area,\ \dot{s}ta,\ attOrd{:}1]\!]$ in $[\![eleOrd{:}0]\!]$, $[\![eleOrd{:}1]\!]$, $[\![eleOrd{:}2]\!]$, $[\![eleOrd{:}3]\!]$ in $[\![\dot{c}on]\!]$, respectively. In particular, this is means that the triangle *ġeoEle* from two-dimensional Euclidean space has an area in $[\![\dot{s}ta]\!]$.
2. *area*, *ġeoEle*, *triangle*, *Euclidean* and *2* are elements in $[\![att{:}inch,\ \dot{s}ta,\ attOrd{:}2]\!]$ in $[\![eleOrd{:}-1]\!]$, $[\![eleOrd{:}0]\!]$, $[\![eleOrd{:}1]\!]$, $[\![eleOrd{:}2]\!]$, $[\![eleOrd{:}3]\!]$ in $[\![\dot{c}on]\!]$, respectively. In particular, this means that the area of the triangle *ġeoEle* from two-dimensional Euclidean space is measured in inches in $[\![\dot{s}ta]\!]$.
3. *inch*, *area*, *ġeoEle*, *triangle*, *Euclidean* and *2* are elements in $[\![att{:}10,\ \dot{s}ta,\ attOrd{:}3]\!]$ in $[\![eleOrd{:}-2]\!]$, $[\![eleOrd{:}-1]\!]$, $[\![eleOrd{:}0]\!]$, $[\![eleOrd{:}1]\!]$, $[\![eleOrd{:}2]\!]$, $[\![eleOrd{:}3]\!]$ in $[\![\dot{c}on]\!]$, respectively. This means that the area of the triangle *ġeoEle* from two-dimensional Euclidean space measured in inches is represented in the decimal system in $[\![\dot{s}ta]\!]$.

•

Proposition 7. If *ȧtt* is an attribute in $[\![\dot{s}ta]\!]$, and *ėle* is an element in $[\![att{:}\dot{a}tt,\ \dot{s}ta,\ attOrd{:}1]\!]$, then *ėle* is either an individual or a concept in $[\![\dot{s}ta]\!]$.

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

A set *ṡet* is called the content in $[\![att{:}\dot{a}tt,\ \dot{s}ta,\ attOrd{:}\dot{n}at_1,\ eleOrd{:}\dot{n}at_2,\ \dot{c}on[\![\dot{s}ta]\!]]\!]$, if *ṡet* is a set of elements in $[\![att{:}\dot{a}tt,\ \dot{s}ta,\ attOrd{:}\dot{n}at_1,\ eleOrd{:}\dot{n}at_2,\ \dot{c}on]\!]$. The content of an attribute describes its semantics.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta = (ċon:3)*. Then the following properties hold:

- *{ġeoEle, triangle, Euclidean, 2}* is the content in $[\![att{:}area,\ \dot{s}ta]\!]$;

- *{area, ġeoEle, triangle, Euclidean, 2}* is the content in ⟦*att:inch, ṡta*⟧;
- *{inch, area, ġeoEle, triangle, Euclidean, 2}* is the content in ⟦*att:10, ṡta*⟧.

•

An element *ėle* is called an element in ⟦*att:ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟦*ṡta*⟧*, med:ṅat0*⟧, if *ėle* is an element in ⟦*ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟧, and *ṅat0* is the number of element orders *ṅat* in ⟦*ċon*⟧ such that $\dot{n}at_2 < \dot{n}at < \dot{n}at_1$. A number *ṅat0* is called a mediatorial degree in ⟦*ėle, att:ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟧. It specifies how many mediators separate *ėle* from *ȧtt* in *ċon*. An element *ėle′* is called a mediator in ⟦*ėle, att:ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟦*ṡta*⟧⟧, if *ėle′* is an element in ⟦*ċon, ṅat*⟧, and $\dot{n}at_2 < \dot{n}at < \dot{n}at_1$.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta = (ċon:3)*. Then *ġeoEle* is an element in the following contexts:

- ⟦*att:area, ṡta, attOrd:1, eleOrd:0, ċon*⟧ with the mediatorial degree *0* and without mediators;
- ⟦*att:inch, ṡta, attOrd:2, eleOrd:0, ċon*⟧ with the mediatorial degree *1* and the mediator *area*;
- ⟦*att:10, ṡta, attOrd:3, eleOrd:0, ċon*⟧ with the mediatorial degree *2* and the mediators *area* and *inch*.

•

An element *ėle* is called a direct element in ⟦*att:ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟦*ṡta*⟧⟧, if *ėle* is an element in ⟦*att:ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon, med:0*⟧.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)*, and *ṡta = (ċon:3)*. Then the following properties hold:

1. *ġeoEle* is a direct element in ⟦*att:area, ṡta, attOrd:1, eleOrd:0*⟧. This means that the individual *ġeoEle* has an area in ⟦*ṡta*⟧.
2. *area* is a direct element in ⟦*att:inch, ṡta, attOrd:2, eleOrd:1*⟧. This means that an area can be measured in inches in ⟦*ṡta*⟧.
3. *inch* is a direct element in ⟦*att:10, ṡta, attOrd:3, eleOrd:2*⟧. This means that values of numerical characteristics of geometric elements measured in inches can be represented in the decimal system in ⟦*ṡta*⟧.

•

A set *ṡet* is called the direct content in ⟦*att:ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟦*ṡta*⟧⟧, if *ṡet* is a set of direct elements in ⟦*att:ȧtt, ṡta, attOrd:ṅat*$_1$*, eleOrd:ṅat*$_2$*, ċon*⟧.

Example. Let $\dot{con}_1$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{g}$*eoEle*$_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_2$ = *(−3:10, −2:cm, −1:area, 0:*$\dot{g}$*eoEle*$_2$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_3$ = *(−3:2, 0:*$\dot{g}$*eoEle*$_1$*, 3:2)*, and $\dot{sta}$ = *(*$\dot{con}_1$*:3,* $\dot{con}_2$*:4,* $\dot{con}_3$*:2)*. Then the following properties hold:

- *{*$\dot{g}$*eoEle*$_1$*,* $\dot{g}$*eoEle*$_2$*}* is the direct content in ⟦*att:area,* $\dot{sta}$⟧;
- *{area}* is the direct content in ⟦*att:inch*⟧ and ⟦*att:cm*⟧ in ⟦$\dot{sta}$⟧;
- *{inch, cm}* is the direct content in ⟦*att:10,* $\dot{sta}$⟧;
- *{*$\dot{g}$*eoEle*$_1$*}* is the direct content in ⟦*att:2,* $\dot{sta}$⟧.

•

A set $\dot{set}$ is called the content in ⟦*att:*$\dot{att}$*,* $\dot{sta}$*, concOrd:*$\dot{n}$*at*$_1$*, eleOrd:*$\dot{n}$*at*$_2$*,* $\dot{con}$⟦$\dot{sta}$⟧*, med:*$\dot{n}$*at0*⟧, if $\dot{set}$ is a set of elements in ⟦*att:*$\dot{att}$*,* $\dot{sta}$*, attOrd:*$\dot{n}$*at*$_1$*, eleOrd:*$\dot{n}$*at*$_2$*,* $\dot{con}$*, med:*$\dot{n}$*at0*⟧.

Example. Let $\dot{con}_1$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{g}$*eoEle*$_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_2$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{g}$*eoEle*$_2$*, 1:triangle, 2:Riemannian, 3:2)*, $\dot{con}_3$ = *(−3:10, −2:inch, 0:*$\dot{g}$*eoEle*$_3$*, 1:square, 2:Euclidean, 3:2)*, and $\dot{sta}$ = *(*$\dot{con}_1$*:3,* $\dot{con}_2$*:4,* $\dot{con}_3$*:2)*. Then the following properties hold:

- *{*$\dot{g}$*eoEle*$_1$*,* $\dot{g}$*eoEle*$_2$*}* is the content in ⟦*att:10,* $\dot{sta}$*, attOrd:3, eleOrd:0, med:2*⟧;
- *{*$\dot{g}$*eoEle*$_3$*}* is the content in ⟦*att:10,* $\dot{sta}$*, attOrd:3, eleOrd:0, med:1*⟧;
- *{triangle}* is the content in ⟦*att:10,* $\dot{sta}$*, concOrd:3, eleOrd:1, med:3*⟧;
- *{square}* is the content in ⟦*att:10,* $\dot{sta}$*, attOrd:3, eleOrd:1, med:2*⟧.

•

4.4. Classification and interpretation of attributes

Attributes are classified according to their orders.

An attribute $\dot{att}$ in ⟦$\dot{sta}$*, attOrd:1*⟧ specifies a usual attribute of the ontology of $\dot{sys}$. Elements in ⟦*att:*$\dot{att}$*,* $\dot{sta}$*, attOrd:1*⟧ are individuals and concepts in ⟦$\dot{sta}$⟧.

Example. Let $\dot{con}_1$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{g}$*eoEle*$_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_2$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{g}$*eoEle*$_2$*, 1:square, 2:Riemannian, 3:3)*, and $\dot{sta}$ = *(*$\dot{con}_1$*:3,* $\dot{con}_2$*:4)*. Then the following properties hold:

1. The direct attribute *area* specifies an area of geometric elements in ⟦$\dot{sta}$⟧.
2. The individuals $\dot{g}$*eoEle*$_1$ and $\dot{g}$*eoEle*$_2$ are elements of the order *0* of the direct attribute *area* in ⟦$\dot{sta}$⟧. This means that $\dot{g}$*eoEle*$_1$ and $\dot{g}$*eoEle*$_2$ has an area in ⟦$\dot{sta}$⟧.
3. The concepts *triangle* and *square* are elements of the order *1* of the direct attribute *area* in ⟦$\dot{sta}$⟧. This means that triangles and squares can have an area in ⟦$\dot{sta}$⟧.

4. The concept spaces *Euclidean* and *Riemannian* are elements of the order *2* of the direct attribute *area* in $[\![\dot{sta}]\!]$. This means that numerical characteristics of geometric elements from Euclidean and Riemannian spaces can have an area in $[\![\dot{sta}]\!]$.
5. The concept space spaces *2* and *3* are elements of the order *3* of the direct attribute *area* in $[\![\dot{sta}]\!]$. This means that values of numerical characteristics of geometric elements from two-dimensional and three-dimensional spaces can have an area in $[\![\dot{sta}]\!]$.

•

An attribute *ȧtt* in $[\![\dot{sta}, attOrd{:}2]\!]$ specifies an attribute space of the ontology of *ṡys*. Elements in $[\![att{:}\dot{att}, \dot{sta}, attOrd{:}2]\!]$ are direct attributes, individuals and concepts in $[\![\dot{sta}]\!]$.

Example. Let $\dot{con}_1 = (-3{:}10, -2{:}inch, -1{:}area, 0{:}\dot{geoEle}_1, 1{:}triangle, 2{:}Euclidean, 3{:}2)$, $\dot{con}_2 = (-3{:}10, -2{:}inch, -1{:}perimeter, 0{:}\dot{geoEle}_2, 1{:}square, 2{:}Riemannian, 3{:}3)$, and $\dot{sta} = (\dot{con}_1{:}3, \dot{con}_2{:}4)$. Then the following properties hold:

1. The attribute space *inch* specifies numerical characteristics of geometric elements measured in inches in $[\![\dot{sta}]\!]$.
2. The direct attributes *area* and *perimeter* are elements of the order -1*1* of the attribute space *inch* in $[\![\dot{sta}]\!]$. This means that areas and perimeters of geometric elements can be measured in inches in $[\![\dot{sta}]\!]$.
3. The individuals $\dot{geoEle}_1$ and $\dot{geoEle}_2$ are elements of the order *0* of the attribute space *inch* in $[\![\dot{sta}]\!]$. This means that geometric elements $\dot{geoEle}_1$ and $\dot{geoEle}_2$ can have numerical characteristics measured in inches in $[\![\dot{sta}]\!]$.
4. The concepts *triangle* and *square* are elements of the order *1* of the attribute space *inch* in $[\![\dot{sta}]\!]$. This means that numerical characteristics of triangles and squares can be measured in inches in $[\![\dot{sta}]\!]$.
5. The concept spaces *Euclidean* and *Riemannian* are elements of the order *2* of the attribute space *inch* in $[\![\dot{sta}]\!]$. This means that numerical characteristics of geometric elements from Euclidean and Riemannian spaces can be measured in inches in $[\![\dot{sta}]\!]$.
6. The concept space spaces *2* and *3* are elements of the order *3* of the attribute space *inch* in $[\![\dot{sta}]\!]$. This means that numerical characteristics of geometric elements from two-dimensional and three-dimensional spaces can be measured in inches in $[\![\dot{sta}]\!]$.

•

An attribute *ȧtt* in $[\![\dot{sta}, attOrd{:}3]\!]$ specifies a space of attribute spaces of the ontology of *ṡys*. Elements in $[\![att{:}\dot{att}, \dot{sta}, attOrd{:}1]\!]$ are attribute spaces, direct attributes, individuals and

concepts in $[\![\dot{s}ta]\!]$.

Example. Let $\dot{c}on_1$ = *(−3:10, −2:inch, −1:area, 0:ġeoEle*$_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{c}on_2$ = *(−3:10, −2:cm, −1:perimeter, 0:ġeoEle*$_2$*, 1:square, 2:Riemannian, 3:3)*, and $\dot{s}ta$ = *(*$\dot{c}on_1$*:3,* $\dot{c}on_2$*:4)*. Then the following properties hold:

1. The attribute space space *10* specifies geometric elements, values of numerical characteristics of which are represented in the decimal system.
2. The attribute spaces *inch* and *cm* are elements of the order *−2* of the attribute space space *10* in $[\![\dot{s}ta]\!]$. This means that values of numerical characteristics of geometric figures measured in inches can be represented in the decimal system in $[\![\dot{s}ta]\!]$.
3. The direct attributes *area* and *perimeter* are elements of the order *−11* of the attribute space space *10* in $[\![\dot{s}ta]\!]$. This means that values of areas and perimeters of geometric elements can be represented in the decimal system in $[\![\dot{s}ta]\!]$.
4. The individuals $\dot{g}eoEle_1$ and $\dot{g}eoEle_2$ are elements of the order *0* of the attribute space space *10* in $[\![\dot{s}ta]\!]$. This means that values of numerical characteristics of geometric elements $\dot{g}eoEle_1$ and $\dot{g}eoEle_2$ can be represented in the decimal system in $[\![\dot{s}ta]\!]$.
5. The concepts *triangle* and *square* are elements of the order *1* of the attribute space space *10* in $[\![\dot{s}ta]\!]$. This means that values of numerical characteristics of triangles and squares can be represented in the decimal system in $[\![\dot{s}ta]\!]$.
6. The concept spaces *Euclidean* and *Riemannian* are elements of the order *2* of the attribute space space *10* in $[\![\dot{s}ta]\!]$. This means that numerical characteristics of geometric elements from Euclidean and Riemannian spaces can be represented in the decimal system in $[\![\dot{s}ta]\!]$.
7. The concept space spaces *10* and *2* are elements of the order *3* of the attribute space space *10* in $[\![\dot{s}ta]\!]$. This means that the values of numerical characteristics of geometric elements from two-dimensional and three-dimensional spaces can be represented in the decimal system in $[\![\dot{s}ta]\!]$.

•

An attribute $\dot{a}tt$ in $[\![\dot{s}ta,\ attOrd{:}\dot{n}at]\!]$, where $\dot{n}at > 3$, is classified and interpreted in the similar way (by the introduction of spaces of attribute space space and so on.).

4.5. Notes about elements of states

Concepts and attributes are opposite (symmetric in some respects) entities. Concepts generalize (combine into groups) elements of $\dot{c}ts$. In contrast, attributes concretize (divide into

sub-elements) elements of $\dot{cts}$.

In addition to specification of elements of $\dot{sys}$, an individual $\dot{ele}[\![\dot{sta}]\!]$ can be interpreted in two ways:

- in the attribute context, $\dot{ele}$ is interpreted as an attribute in $[\![\dot{sta}, \mathit{attOrd{:}0}]\!]$. In this case, it specifies a global attribute of $\dot{sys}$;
- in the concept context, $\dot{ele}$ is interpreted as a concept in $[\![\dot{sta}, \mathit{concOrd{:}0}]\!]$. In this case, it specifies a concept which has the single instance $\dot{ele}$.

5. Classification of conceptuals

5.1. General principles and definitions

The two-level scheme of classification of conceptuals is used. 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.

5.1.1. Concretization orders of conceptuals

The number *0* is called an order in $[\![\dot{con}]\!]$, if the minimal order in $\dot{con}$ is greater than or equal to *0*. A number $\dot{nat}$ is called an order in $[\![\dot{con}]\!]$, if $-\dot{nat}$ is a minimal order in $\dot{con}$.

Example. Let $\dot{con}_1$ = *(−3:10, −2:inch, −1:area, 0:*$\dot{geoEle}_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_2$ = *(−2:inch, −1:area, 0:*$\dot{geoEle}_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_3$ = *(−1:area, 0:*$\dot{geoEle}_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_4$ = *(0:*$\dot{geoEle}_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_5$ = *(1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_6$ = *(2:Euclidean, 3:2)*, and $\dot{con}_7$ = *(3:2)*. Then the following properties hold:

1. The conceptuals $\dot{con}_1$, $\dot{con}_2$, $\dot{con}_3$ have the orders *3*, *2*, *1*, respectively.
2. The conceptuals $\dot{con}_4$, $\dot{con}_5$, $\dot{con}_6$, $\dot{con}_7$ have the order *0*.

•

Conceptuals of the order $\dot{nat}$ concretizes conceptuals of the orders which are less than $\dot{nat}$. They define the special kinds of such conceptuals and are used to classify them. Concretization is performed by attributes of the order $\dot{nat}$ and their values. Therefore, the order of a conceptual is also called the concretization order of the conceptual.

5.1.2. Integral orders of conceptuals

A set *ṡet* is called an integral order in $[\![\dot{con}]\!]$, if *ṡet* is a set of all element orders in $[\![\dot{con}]\!]$. Let *intOrd* be a set of integral orders.

Proposition 8. A conceptual *ċon* has the single integral order.

Proof. This follows from the definition of the integral order of a conceptual. □

Example. Let $\dot{con}_1$ = *(−3:10, −2:inch, −1:area, 0:ġeoEle*$_1$*, 1:triangle, 2:Euclidean, 3:2)*, $\dot{con}_1$ = *(−3:10, −1:area, 1:triangle, 3:2)*, $\dot{con}_1$ = *(−2:inch, −1:area, 2:Euclidean, 3:2)*. Then $\dot{intOrd}[\![\dot{con}_1]\!]$ = *{−3, −2, −1, 0, 1, 2, 3}*, $\dot{intOrd}[\![\dot{con}_2]\!]$ = *{−3, −1, 1, 3}*, and $\dot{intOrd}[\![\dot{con}_3]\!]$ = *{−2, −1, 2, 3}*. •

A set *ṡet* is called a refined integral order in $[\![\dot{con}]\!]$, if *ṡet* is a result of replacement of zero or more element orders $\dot{eleOrd}[\![\dot{con}]\!]$ in the set $\dot{intOrd}[\![\dot{con}]\!]$ by objects *ėleOrd:ċon(ėleOrd)*. A refined integral order in *ċon* refines an integral order in *ċon*, providing information on some elements of *ċon* with their orders. Let *ċon:*$\dot{intOrd}$ denote a conceptual *ċon* which has the refined integral order $\dot{intOrd}$.

Example. Let *ċon = (−3:10, −2:inch, −1:area, 0:ġeoEle*$_1$*, 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:ġeoEle*$_1$*, 1:triangle, 2:Euclidean, 3:2}* are refined integral orders in $[\![\dot{con}]\!]$. •

Proposition 9. A conceptual *ċon* has a finite set of refined integral orders.

Proof. This follows from the definition of the refined integral order and the finite number of element orders of conceptuals. □

Proposition 10. The integral order in $[\![\dot{con}]\!]$ is a refined integral order in $[\![\dot{con}]\!]$.

Proof. This follows from the definition of the refined integral order of a conceptual. □

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 classify ontological elements in detail. Each kind of conceptuals specifies a separate kind of ontological elements.

5.2. Correlation between ontological elements and conceptuals of the order *0*

In this section conceptuals of the order *0* is classified according to their integral orders and kinds of conceptuals of this classification are correlated with the corresponding kinds of ontological elements.

A conceptual *ċon:{0}* specifies the individual *ċon(0)*.

Example. The conceptual *(0:ġeoEle)* specifies the geometric element *ġeoEle*. •

A conceptual *ċon:{0, 1}* specifies the individual *ċon(0)* from the concept *ċon(1)*.

Example. The conceptual *(0:ġeoEle, 1:triangle)* specifies the triangle *ġeoEle*. •

A conceptual *ċon:{1}* specifies the concept *ċon(1)*.

Example. A conceptual *(1:triangle)* specifies triangles. •

A conceptual *ċon:{1, 2}* specifies the concept *ċon(1)* from the concept space *ċon(2)*.

Example. The conceptual *(1:triangle, 2:Euclidean)* specifies triangles from Euclidean space. •

A conceptual *ċon:{2}* specifies the concept space *ċon(2)*.

Example. The conceptual *(2:Euclidean)* specifies Euclidean space. •

A conceptual *ċon:{0, 2}* specifies the individual *ċon(0)* from the concept space *ċon(2)*.

Example. The conceptual *(0:ġeoEle, 2:Euclidean)* specifies the geometric element *ġeoEle* from Euclidean space. •

A conceptual *ċon:{0, 1, 2}* specifies the individual *ċon(0)* from the concept *ċon(1)* from the concept space *ċon(2)*.

Example. The conceptual *(0:ġeoEle, 1:triangle, 2:Euclidean)* specifies the triangle *ġeoEle* from Euclidean space. •

Correlation between other kinds of conceptuals of the order *0* and the corresponding kinds of ontological elements is performed in a similar way. For example, a conceptual *ċon:{0, 1, 2, 3}* specifies the individual *ċon(0)* from the concept *ċon(1)* from the concept space *ċon(2)* from the concept space space *ċon(3)*.

Example. The conceptual *(0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)* specifies the triangle *ġeoEle* from two-dimensional Euclidean space. •

5.3. Correlation between ontological elements and conceptuals of the order *1*

In this section conceptuals of the order *1* is classified according to their integral orders and kinds of conceptuals of this classification are correlated with the corresponding kinds of ontological elements.

A conceptual *ċon:{-1}* specifies the attribute *ċon(-1)*.

Example. The conceptual *(−1:area)* specifies an area of geometric elements. •

A conceptual *ċon:{-1, 0}* specifies the attribute *ċon(-1)* of the individual *ċon(0)*.

Example. The conceptual *(−1:area, 0:ġeoEle)* specifies the area of the geometric element *ġeoEle*. •

A conceptual *ċon:{-1, 0, 1}* specifies attribute *ċon(-1)* of the individual *ċon(0)* from the concept *ċon(1)*.

Example. The conceptual *(−1:area, 0:ġeoEle, 1:triangle)* specifies an area of the triangle *ġeoEle*. •

A conceptual *ċon:{-1, 1}* specifies attribute *ċon(-1)* of the concept *ċon(1)*.

Example. The conceptual *(−1:area, 1:triangle)* specifies areas of triangles. •

A conceptual *ċon:{-1, 0, 1, 2}* specifies attribute *ċon(-1)* of the individual *ċon(0)* from the concept *ċon(1)* from the concept space *ċon(2)*.

Example. The conceptual *(−1:area, 0:ġeoEle, 1:triangle, 2:Euclidean)* specifies the area of the triangle *ġeoEle* from Euclidean space. •

A conceptual *ċon:{-1, 1, 2}* specifies attribute *ċon(-1)* of the concept *ċon(1)* from the concept space *ċon(2)*.

Example. The conceptual *(−1:area, 1:triangle, 2:Euclidean)* specifies areas of triangles from Euclidean space. •

A conceptual *ċon:{-1, 0, 2}* specifies attribute *ċon(-1)* of the individual *ċon(0)* from the concept space *ċon(2)*.

Example. The conceptual *(−1:area, 0:ġeoEle, 2:Euclidean)* specifies the area of the geometric element *ġeoEle* from Euclidean space. •

A conceptual *ċon:{-1, 2}* specifies attribute *ċon(-1)* of the concept space *ċon(2)*.

Example. The conceptual *(−1:area, 2:Euclidean)* specifies areas of geometric elements from 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.

5.4. Correlation between ontological elements and conceptuals of the order *2*

In this section conceptuals of the order *2* is classified according to their integral orders and kinds of conceptuals of this classification are correlated with the corresponding kinds of ontological elements.

A conceptual *ċon:{-2, -1}* specifies the attribute *ċon(-1)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area)* specifies areas in inches. •

A conceptual *ċon:{-2, -1, 0}* specifies the attribute *ċon(-1)* of the individual *ċon(0)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area, 0:ġeoEle)* specifies the area of the geometric element *ġeoEle* in inches. •

A conceptual *ċon:{-2, -1, 0, 1}* specifies the attribute *ċon(-1)* of the individual *ċon(0)* from the concept *ċon(1)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area, 0:ġeoEle, 1:triangle)* specifies the area of the triangle *ġeoEle* in inches. •

A conceptual *ċon:{-2, -1, 1}* specifies the attribute *ċon(-1)* of the concept *ċon(1)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area, 1:triangle)* specifies areas of triangles in inches. •

A conceptual *ċon:{-2, -1, 0, 1, 2}* specifies the attribute *ċon(-1)* of the individual *ċon(0)* from the concept *ċon(1)* from the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean)* specifies the area of the triangle *ġeoEle* from Euclidean space in inches. •

A conceptual *ċon:{-2, -1, 1, 2}* specifies the attribute *ċon(-1)* of the concept *ċon(1)* from the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area, 1:triangle, 2:Euclidean)* specifies areas of triangles from Euclidean space in inches. •

A conceptual *ċon:{-2, -1, 0, 2}* specifies the attribute *ċon(-1)* of the individual *ċon(0)* from the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area, 0:ġeoEle, 2:Euclidean)* specifies the area of the geometric element *ġeoEle* from Euclidean space in inches. •

A conceptual *ċon:{−2, −1, 2}* specifies the attribute *ċon(-1)* of the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, −1:area, 2:Euclidean)* specifies areas of geometric elements from Euclidean space in inches. •

A conceptual *ċon:{-2, 0}* specifies the individual *ċon(0)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, 0:ġeoEle)* specifies the attributes of the geometric element *ġeoEle* measured in inches. •

A conceptual *ċon:{-2, 0, 1}* specifies the individual *ċon(0)* from the concept *ċon(1)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, 0:ġeoEle, 1:triangle)* specifies the attributes of the triangle *ġeoEle* measured in inches. •

A conceptual *ċon:{-2, 1}* specifies the concept *ċon(1)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, 1:triangle)* specifies attributes and individuals of triangles measured in inches. •

A conceptual *ċon:{−2, 1, 2}* specifies the concept *ċon(1)* from the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, 1:triangle, 2:Euclidean)* specifies attributes and individuals of triangles from Euclidean space measured in inches. •

A conceptual *ċon:{-2, 2}* specifies the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, 2:Euclidean)* specifies attributes, individuals and kinds of geometric figures from Euclidean space measured in inches. •

A conceptual *ċon:{-2, 0, 2}* specifies the individual *ċon(0)* from the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, 0:ġeoEle, 2:Euclidean)* specifies the geometric element *ġeoEle* and its attributes from Euclidean space measured in inches. •

A conceptual *ċon:{-2, 0, 1, 2}* specifies the individual *ċon(0)* from the concept *ċon(1)* from the concept space *ċon(2)* in the attribute space *ċon(-2)*.

Example. The conceptual *(−2:inch, 0:ġeoEle, 1:triangle, 2:Euclidean)* specifies the triangle *ġeoEle* and its attributes from Euclidean space 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.

5.5. Correlation between ontological elements and conceptuals of the order *3* or higher

Correlation between other kinds of conceptuals of the order *3* or higher and the corresponding kinds of ontological elements is performed in a similar way (by the introduction of the attribute space space and so on.).

Example. The conceptual *(−3:10, −2:inch, −1:area, 0:ġeoEle, 1:triangle, 2:Euclidean, 3:2)* specifies the area of the triangle *ġeoEle* from two-dimensional Euclidean space measured

in inches in the decimal system. •

6. Modelling of ontological elements

The ontological elements which are directly represented in terms of elements and conceptuals of states were considered in previous sections. The ontological elements which are not directly represented in these terms are modelled in this section.

6.1. Relations and their instances

Binary relations are modelled by direct concepts, and their instances are modelled by elements of the order *0* of these concepts, represented by pairs of elements.

Relations of the arity $\dot{n}at$ are modelled by direct concepts, and their instances are modelled by elements of the order *0* of these concepts, represented by tuples of the length $\dot{n}at$.

Relations of the variable arity are modelled by direct concepts, and their instances are modelled by elements of the order *0* of these concepts, represented by tuples of the variable length.

6.2. Types and domains

Types are modelled by direct concepts, and their values are modelled by elements of the order *0* of these concepts.

Domains as the special kind of types are also modelled by direct concepts, and their values are modelled by elements of the order *0* of these concepts.

Types of attributes of the order $\dot{n}at$ are modelled by the special attribute *type* of the order $\dot{n}at + 1$. Values of this attribute are types modelled by direct concepts.

6.3. Inheritance

The usual inheritance relation on concepts is generalized to the inheritance relation on elements of the same order in $[\![\dot{s}ta]\!]$. It is modelled by the special direct concept *inherits*, and their instances are modelled by elements of the order *0* of the concept *inherits*, represented by triples of elements. Elements of the triple specify the inheriting element, the inherited element and their order, respectively.

An element $\dot{e}le[\![\dot{s}ta]\!]$ inherits from $\dot{e}le'[\![\dot{s}ta]\!]$ in $[\![\dot{s}ta, \dot{i}nt]\!]$, if $\dot{s}ta(0{:}(\dot{e}le, \dot{e}le', \dot{i}nt), 1{:}inherits) \neq \omega$.

Inheritance on elements redefines semantics of conceptuals *sem* as follows:

- if $\dot{sta}(\dot{con}) \neq \omega$, then $sem(\dot{con}, \dot{sta}) = \dot{sta}(\dot{con})$;
- if
  - $\dot{sta}(\dot{con}) = \omega$,
  - $\dot{int}$ is a maximal order in $[\![\dot{con}]\!]$,
  - $\dot{set}$ is a set of $\dot{ele}[\![\dot{sta}]\!]$ such that $\dot{con}(\dot{int})$ inherits from $\dot{ele}$ in $[\![\dot{sta}, \dot{int}]\!]$,
  - $\dot{set} \neq \emptyset$,
  - $sem(\dot{con}(\dot{int} \leftarrow \dot{ele}), \dot{sta}) = sem(\dot{con}(\dot{int} \leftarrow \dot{ele}'), \dot{sta})$ for all $\dot{ele}, \dot{ele}' \in \dot{set}$,

  then $sem(\dot{con}, \dot{sta}) = sem(\dot{con}(\dot{int} \leftarrow \dot{ele}), \dot{sta})$, where $\dot{ele} \in \dot{set}$;
- otherwise, $sem(\dot{con}, \dot{sta}) = \omega$.

The special case of inheritance on direct concepts is defined. A concept $\dot{dirConc}[\![\dot{sta}]\!]$ inherits from a concept $\dot{dirConc}'[\![\dot{sta}]\!]$ in $[\![\dot{sta}]\!]$, if $\dot{dirConc}$ inherits from $\dot{dirConc}'$ in $[\![\dot{sta}, 1]\!]$.

The inheritance relation on elements of the same order in $[\![\dot{sta}]\!]$ is generalized to the inheritance relation on ordered structures of elements of the same length in $[\![\dot{sta}]\!]$. The corresponding elements of these structures have the same order. This relation is modelled by the special direct concept *inheritsStr*, and their instances are modelled by elements of the order *0* of this concept, represented by triples of sorted structures of the same length. The elements of the triple specify the ordered structures of inheriting elements, inherited elements and their orders, respectively.

An element $\dot{ordStr}_1$ inherits from $\dot{ordStr}_2$ in $[\![\dot{sta}, \dot{ordStr}]\!]$, if the following properties hold:

- $\dot{ordStr} = (\dot{int}_1, ..., \dot{int}_{\dot{nat}})$;
- $\dot{int}_1 < \ldots < \dot{int}_{\dot{nat}}$;
- $len(\dot{ordStr}_1) = len(\dot{ordStr}_2) = \dot{nat}$;
- $sem((0{:}(\dot{ordStr}_1, \dot{ordStr}_2, \dot{ordStr}), 1{:}inheritsStr), \dot{sta}) \neq \omega$.

Inheritance on ordered structures redefines semantics of conceptuals *sem*:

- if $\dot{sta}(\dot{con}) \neq \omega$, then $sem(\dot{con}, \dot{sta}) = \dot{sta}(\dot{con})$;
- if
  - $\dot{sta}(\dot{con}) = \omega$,
  - $\dot{int}_1 < \ldots < \dot{int}_{\dot{nat}}$ are element orders in $\dot{con}$,
  - for all $\dot{int}$ if $\dot{int} \geq int_1$, and $\dot{int}$ is an element order in $\dot{con}$, then $\dot{int}$ coincides with one of the numbers $\dot{int}_1$, ..., $\dot{int}_{\dot{nat}}$,
  - $\dot{set}$ is a set of $\dot{ele}[\![\dot{sta}]\!]$ such that $(\dot{con}(\dot{int}_1), \ldots, \dot{con}(\dot{int}_{\dot{nat}}))$ inherits from $\dot{ele}$ in $[\![\dot{sta}, (\dot{int}_1, \ldots, \dot{int}_{\dot{nat}})]\!]$,
  - $\dot{set} \neq \emptyset$,

– for all $\dot{ele}$, $\dot{ele}' \in \dot{set}$

$$\dot{sem}(\dot{con}(\dot{int}_1 \leftarrow \dot{ele}(\dot{int}_1), \ldots, \dot{int}_{\dot{nat}} \leftarrow \dot{ele}(\dot{int}_{\dot{nat}})), \dot{sta}) = \dot{sem}(\dot{con}(\dot{int}_1 \leftarrow \dot{ele}'(\dot{int}_1), \ldots, \dot{int}_{\dot{nat}} \leftarrow \dot{ele}'(\dot{int}_{\dot{nat}})), \dot{sta}),$$

then $\dot{sem}(\dot{con}, \dot{sta}) = \dot{sem}(\dot{con}(\dot{int}_1 \leftarrow \dot{ele}(\dot{int}_1), \ldots, \dot{int}_{\dot{nat}} \leftarrow \dot{ele}(\dot{int}_{\dot{nat}})), \dot{sta})$, where $\dot{ele} \in \dot{set}$;

- otherwise, $\dot{sem}(\dot{con}, \dot{sta}) = \omega$.

7. 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.

7.1. The main definitions

Let $* \in ato$. A conceptual $\dot{con}[\![\dot{sta}]\!]$ is called a generic conceptual in $[\![\dot{sta}]\!]$, if there exists $\dot{eleOrd}$ such that $\dot{con}(\dot{eleOrd}) \in \{*, (*, \dot{ele}_2), (*, \dot{ele}_2, \dot{ele}_3), (*, *, \dot{ele}_3)\}$. The element ele of the form $\dot{con}(\dot{eleOrd})$ from this definition is called a substitution place in $[\![\dot{con}, \dot{sta}, \dot{eleOrd}]\!]$. Let $genCon$ and pla be sets of generic conceptuals and substitution places, respectively. The number $\dot{eleOrd}$ is called an order in $[\![\dot{pla}, \dot{con}, \dot{sta}]\!]$. The elements $\dot{ele}_2$ and $\dot{ele}_3$ are called a type and parameter in $[\![\dot{pla}, \dot{con}, \dot{sta}, \dot{eleOrd}]\!]$, respectively. Let $type$ and par be sets of types and parameters in $[\![\dot{pla}, \dot{con}, \dot{sta}, \dot{eleOrd}]\!]$, respectively.

A conceptual $\dot{genCon}$ is called partially typed in $[\![\dot{sta}]\!]$, if there exist $\dot{pla}$, $\dot{type}$ and $\dot{eleOrd}$ such that $\dot{pla}$ is a substitution place in $[\![\dot{genCon}, \dot{sta}, \dot{eleOrd}]\!]$, and $\dot{type}$ is a type in $[\![\dot{pla}, \dot{genCon}, \dot{sta}, \dot{eleOrd}]\!]$.

A conceptual $\dot{genCon}$ is called typed in $[\![\dot{sta}]\!]$, if for all $\dot{pla}$ and $\dot{eleOrd}$, if $\dot{pla}$ is a substitution place in $[\![\dot{genCon}, \dot{sta}, \dot{eleOrd}]\!]$, then there exists $\dot{type}$ such that $\dot{type}$ is a type in $[\![\dot{pla}, \dot{genCon}, \dot{sta}, eleOrd]\!]$.

A conceptual $\dot{genCon}$ is called parametric in $[\![\dot{sta}]\!]$, if there exist $\dot{pla}$, $\dot{par}$ and $\dot{eleOrd}$ such that $\dot{pla}$ is a substitution place in $[\![\dot{genCon}, \dot{sta}, \dot{eleOrd}]\!]$, and $\dot{par}$ is a parameter in $[\![\dot{pla}, \dot{genCon}, \dot{sta}, \dot{eleOrd}]\!]$.

A conceptual $\dot{con}$ is called an instance in $[\![\dot{genCon}, \dot{sta}]\!]$, if the following properties hold:

- if $\dot{genCon}(\dot{int})$ is not a substitution place in $[\![\dot{genCon}, \dot{sta}, \dot{int}]\!]$, then $\dot{con}(\dot{int}) = \dot{genCon}(\dot{int})$;
- if $\dot{genCon}(\dot{int})$ is a substitution place in $[\![\dot{genCon}, \dot{sta}, \dot{int}]\!]$, then $\dot{con}(\dot{int})$ is an element

in $[\![\dot{sta},\ \dot{int}]\!]$;

- if $\dot{genCon(int)} \in \{(*,\ \dot{type}),\ (*,\ \dot{type},\ \dot{par})\}$, then $\dot{con(int)}$ is an element in $[\![conc{:}\dot{type},\ \dot{sta},\ concOrd{:}1,\ eleOrd{:}0]\!]$;
- if $\dot{par}$ is a parameter in $[\![\dot{pla}_1,\ \dot{genCon},\ \dot{sta},\ \dot{eleOrd}_1]\!]$ and $[\![\dot{pla}_2,\ \dot{genCon},\ \dot{sta},\ \dot{eleOrd}_2]\!]$, then $\dot{con(eleOrd}_1) = \dot{con(eleOrd}_2)$.

A CTS *cts* is called a CTS in $[\![genCon{:}]\!]$, if the following properties hold:

- **(the consistency property)** if $\dot{genCon}_1 \neq \dot{genCon}_2$, then there is no $\dot{con}$ such that $\dot{con}$ is an instance of $\dot{genCon}_1$ and $\dot{genCon}_2$ in $[\![\dot{sta}]\!]$;
- semantics of conceptuals *sem* is redefined as follows:
  - if $\dot{sta(con)} \neq \omega$, then $sem(\dot{con},\ \dot{sta}) = \dot{sta(con)}$;
  - if $\dot{sta(con)} = \omega$, and $\dot{con}$ is an instance in $[\![\dot{genCon},\ \dot{sta}]\!]$, then $sem(\dot{con},\ \dot{sta}) = \dot{sta(genCon)}$;
  - otherwise, $sem(\dot{con},\ \dot{sta}) = \omega$.

7.2. Examples of generic conceptuals

A conceptual $\dot{genCon}{:}\{-1,\ 0{:}*,\ 1\}$ specifies the property that the value of the attribute $\dot{genCon(-1)}$ of individuals from the concept $\dot{genCon(1)}$ is equal to $\dot{sta(genCon)}$ in $[\![\dot{sta}]\!]$, if it is not defined explicitly.

Example. The conceptual $\dot{genCon}{:}\{-1{:}area,\ 0{:}*,\ 1{:}triangle\}$ specifies the property that the area of triangles is equal to $\dot{sta(genCon)}$ in $[\![\dot{sta}]\!]$, if it is not defined explicitly. •

A conceptual $\dot{genCon}{:}\{-1,\ 0{:}*\}$ specifies the property that the value of the attribute $\dot{genCon(-1)}$ of individuals is equal to $\dot{sta(genCon)}$ in $[\![\dot{sta}]\!]$, if it is not defined explicitly.

Example. The conceptual $\dot{genCon}{:}\{-1{:}area,\ 0{:}*\}$ specifies the property that the area of geometric elements is equal to $\dot{sta(genCon)}$ in $[\![\dot{sta}]\!]$, if it is not defined explicitly. •

A conceptual $\dot{genCon}{:}\{0{:}*,\ 1\}$ specifies the property that the value of individuals from the concept $\dot{genCon(1)}$ is equal to $\dot{sta(genCon)}$ in $[\![\dot{sta}]\!]$, if it is not defined explicitly.

Example. The conceptual $\dot{genCon}{:}\{0{:}*,\ 1{:}triangle\}$ specifies the property that the value of triangles is equal to $\dot{sta(genCon)}$ in $[\![\dot{sta}]\!]$, if it is not defined explicitly. What is the value of a triangle depends on interpretation. •

7.3. Classification of ontological elements and their properties based on generic conceptuals

Generic conceptuals together with attributes allow to classify ontological elements and their properties in more detail.

A conceptual *ġenCon:{-2:type, -1, 0*, 1}* specifies the property that the type of the attribute *ġenCon(-1)* of individuals from the concept *ġenCon(1)* is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for individuals explicitly.

Example. The conceptual *ġenCon:{-2:type, −1:area, 0:*, 1:triangle}* specifies the property that the type of the attribute *area* of triangles is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for triangles explicitly. •

A conceptual *ġenCon:{-2:type, -1, 0:*}* specifies the property that the type of the attribute *ġenCon(-1)* of individuals is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for individuals explicitly.

Example. The conceptual *ġenCon:{-2:type, −1:area, 0:*}* specifies the property that the type of the attribute *area* of geometric elements is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for geometric elements explicitly. •

A conceptual *ġenCon:{-2:type, 0:*}* specifies the property that the type of individuals is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for individuals explicitly.

Example. The conceptual *ġenCon:{-2:type, 0:*}* specifies the property that the type of geometric elements is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for geometric elements explicitly. •

A conceptual *ġenCon:{-2:type, 0:*, 1}* specifies the property that the type of individuals from the concept *ġenCon(1)* is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for such individuals explicitly.

Example. The conceptual *ġenCon:{-2:type, 0:*, 1:triangle}* specifies the property that the type of triangles is equal to *ṡta(ġenCon)* in $[\![\dot{s}ta]\!]$, if it is not defined for triangles explicitly. •

8. Justification of requirements for conceptual transition systems

In this section, we establish that CTSs meet the requirements stated in section 1:

1. *The formalism describes the conceptual structure of the specified system.* The conceptual structure of *sys* is described by elements (attributes, concepts, individuals) and, in more detail, usual and generic conceptuals of states of *ċts*.
2. *The formalism describes the content of the conceptual structure of the specified system, i. e. it describes the specified system in the context of the conceptual structure.* The content

of the conceptual structure of *ṡys* is described by conceptual states of *ċts*.

3. *The formalism describes the change of the conceptual structure of the specified system.* The change of the conceptual structure of *ṡys* is described by the transition relation *traRel* on conceptual states of *ċts* which specify conceptual structures of *ṡys* with different sets of ontological elements.
4. *The formalism describes the change of the content of the conceptual structure of the specified system, i. e. it describes the change of the specified system in the context of the conceptual structure.* The change of the content of the conceptual structure of *ṡys* is described by the transition relation *traRel* on conceptual states of *ċts* which specify the same conceptual structure of *ṡys*. In fact, the distinction between requirements 3 and 4 is relative, for conceptuals allow to define classifications of ontological elements with different granularity.
5. *The formalism is quite universal to specify typical ontological elements (concepts, attributes, concept instances, relations, relation instances, individuals, types, domains, and so on.).* Specification of typical ontological elements is presented in sections 4 and 6.
6. *The formalism provides a quite complete classification of ontological elements, including the determination of their new kinds and subkinds.* Classification of ontological elements based on the two-level scheme is presented in section 5.
7. *The formalism is based on the conception 'state – transition' of the usual transition systems, keeping their simplicity and universality and adding a conceptual 'filling'.* CTSs are the special kind of transition systems in which transitions are defined in the ordinary way, and states are quite simple functions specifying the conceptual structure of the specified systems. Therefore, they keep simplicity and universality of the usual transition systems.
8. *The formalism supports reflection of any order, i. e. allows to specify: the system (reflection of the order 0), the specification of the system (reflection of the order 1), the specification of the specification of the system (reflection of the order 2) and so on. Specifications of the higher order (with reflection of the higher order) impose restrictions on the specifications of the lower order (with reflection of the lower order).* The order of reflection in the specification of *ṡys* is defined by the maximal (concretization) order of conceptuals in states of *ċts*, i. e. the maximal order of attributes in states of *ċts*. The formal description of this property requires additional definitions which are given below.

We extend the (concretization) order of conceptuals on states, the transition relation and CTSs.

A number $\dot{n}at$ is called an order in $[\![\dot{s}ta]\!]$, if the following properties hold:

- there is no $\dot{c}on[\![\dot{s}ta]\!]$ such that the order in $[\![\dot{c}on]\!]$ is greater than $\dot{n}at$;
- there exists $\dot{c}on[\![\dot{s}ta]\!]$ such that $\dot{n}at$ is an order in $[\![\dot{c}on]\!]$.

A state $\dot{s}ta$ is called admissible in $[\![\dot{c}ts]\!]$, if there exists $\dot{s}ta'$ such that *traRel(*$\dot{s}ta$*,* $\dot{s}ta'$*)*, or *traRel(*$\dot{s}ta'$*,* $\dot{s}ta$*)*.

A number $\dot{n}at$ is called an order in $[\![traRel]\!]$, if the following properties hold:

- there is no $\dot{s}ta$ such that $\dot{s}ta$ is admissible in $[\![\dot{c}ts]\!]$, and the order in $[\![\dot{s}ta]\!]$ is greater than $\dot{n}at$;
- there exists $\dot{s}ta$ such that $\dot{s}ta$ is admissible in $[\![\dot{c}ts]\!]$, and $\dot{n}at$ is an order in $[\![\dot{s}ta]\!]$.

A number $\dot{n}at$ is called an order in $[\![\dot{c}ts]\!]$, if $\dot{n}at$ is an order in $[\![traRel[\![\dot{c}ts]\!]]\!]$.

A system $\dot{c}ts$ is called a specification in $[\![\dot{s}ys,\ \dot{n}at0]\!]$, if $\dot{n}at0$ is an order in $[\![\dot{c}ts]\!]$. A number *nat0* is called an order (of specification or reflection) in $[\![\dot{s}ys,\ \dot{c}ts]\!]$.

States, the transition relation and CTSs of greater orders concretize states, the transition relation and CTSs of lower orders, define the special their kinds and are used to classify them.

Thus, the requirements stated in section 1 are met for CTSs.

9. Related formalisms

We compare CTSs with three related formalisms: abstract state machines [1, 2], ontological transition systems [5] and domain-specific transition systems [7]. The comparison is based on the requirements stated in section 1.

9.1. Abstract state machines

Abstract state machines [1, 2] are the special kind of transition systems in which transitions are defined in the ordinary way, and states are algebraic systems. The application of abstract state machine to specifying various systems can be found in [8]. In contrast to CTSs which have no implementation language, abstract state machines have two implementation languages: ASML [9] and XASM [10].

We consider the fulfillment of the requirements for this formalism:

1. *The formalism describes the conceptual structure of the specified system.* The conceptual structure of the specified system is modelled by the appropriate choice of symbols of

the signature of an algebraic system. Thus, both abstract state machines and CTSs describe the conceptual structure of specified systems, but CTSs make it by more natural ontological way.

2. *The formalism describes the content of the conceptual structure of the specified system.* The content of the conceptual structure of the specified system is modelled by the interpretation of signature symbols in a particular state.
3. *The formalism describes the change of the conceptual structure of the specified system.* The change of the conceptual structure of the specified system is described by the transition relation on algebraic structures of different signatures.
4. *The formalism describes the change of the content of the conceptual structure of the specified system.* The change of the content of the conceptual structure of the specified system is described by the transition relation on algebraic structures of the same signature.
5. *The formalism is quite universal to specify typical ontological elements.* In contrast to CTSs, typical ontological elements are not naturally specified by abstract state machines.
6. *The formalism provides a quite complete classification of ontological elements, including the determination of their new kinds and subkinds.* In contrast to CTSs, abstract state machines do not allow to classify naturally ontological elements and define their new kinds and subkinds.
7. *The formalism is based on the conception 'state – transition' of the usual transition systems, keeping their simplicity and universality and adding a conceptual 'filling'.* Abstract state machines are the special kind of transition systems in which transitions are defined in the usual way, and states are functions of algebraic systems. Therefore, they keep simplicity and universality of the usual transition systems sufficiently. The difference between abstract state machines and CTSs consists in that they are based on different (ontological and algebraic, respectively) approaches to the definition of states.
8. *The formalism supports reflection of any order.* In contrast to CTSs, abstract state machines do not support reflection of any order in natural way.

9.2. Ontological transition systems

Ontological transition systems [5] are the special kind of labelled transition systems in which transitions are defined in the usual way, transition labels are actions which change states, and states are ontology content retrieval functions. The ontology in ontological transition systems

are defined as a set of concepts and relations on the universe of objects. The content of a concept is defined as a set of sequences of objects from the universe, and the content of a relation is defined as a set of pairs of sequences of objects form the universe. In contrast to CTSs, ontological transition systems have their associated notation language OTSL [5, 6] which specifies states and transition actions.

We consider the fulfillment of the requirements for this formalism:

1. *The formalism describes the conceptual structure of the specified system.* Ontological transition systems describe the conceptual structure of the specified system by concepts and relations.
2. *The formalism describes the content of the conceptual structure of the specified system.* Ontological transition systems describe the content of the conceptual structure of the specified system by ontology content retrieval functions.
3. *The formalism describes the change of the conceptual structure of the specified system.* In contrast to CTSs, ontological transition sytems do not change the conceptual structure of the specified system.
4. *The formalism describes the change of the content of the conceptual structure of the specified system.* The change of the content of the conceptual structure of the specified system is described by actions of the corresponding ontological transition system.
5. *The formalism is quite universal to specify typical ontological elements.* In contrast to CTSs, only some typical ontological elements are naturally specified by ontological transition systems.
6. *The formalism provides a quite complete classification of ontological elements, including the determination of their new kinds and subkinds.* In contrast to CTSs, a set of kinds of ontological elements which are naturally defined by ontological transition systems is restricted.
7. *The formalism is based on the conception 'state – transition' of the usual transition systems, keeping their simplicity and universality and adding a conceptual 'filling'.* Ontological transition systems are the special kind of transition systems in which transitions are defined in the usual way, and states are quite simple ontology content retrieval functions. Therefore, they keep simplicity and universality of the usual transition systems sufficiently. The language of specification of ontological transition systems OTSL includes a specific set of actions that restrict the transition relation. Therefore, it is not as much

universal as CTSs.

8. *The formalism supports reflection of any order.* In contrast to CTSs, ontological transition systems do not support reflection.

9.3. Domain-specific transition systems

Domain-specific transition systems [7] are the special kind of transition systems in which states are defined by parametric forms and their values, and transitions are defined by the special kind of these forms – transition rules. A parametric form is characterized by a sample, evaluated and quoted parameters, a parameter constraint, a return value constraint and rules of propagation of indeterminate values of parameters. Instances of the form are defined by the pattern matching algorithm. Applying transition rules are described by the algorithms of pattern matching and rule execution.

We consider the fulfillment of the requirements for this formalism:

1. *The formalism describes the conceptual structure of the specified system.* The conceptual structure of the specified system is described by the special kind of domain-specific transition systems – ontological domain-specific transition systems. Concepts in such systems are modelled by forms which represent characteristic functions of these concepts and define their content. The sample of a rule of such system is represented by a parameter, and the parameter constraint defines the concept for which values of the parameter are instances.
2. *The formalism describes the content of the conceptual structure of the specified system.* Ontological domain-specific transition systems describe the content of the conceptual structure of the specified system by concepts.
3. *The formalism describes the change of the conceptual structure of the specified system.* In contrast to CTSs, domain-specific transition systems do not change the conceptual structure of the specified system.
4. *The formalism describes the change of the content of the conceptual structure of the specified system.* The change of the content of the conceptual structure of the specified system is described by rules of the corresponding domain-specific transition system.
5. *The formalism is quite universal to specify typical ontological elements.* In contrast to CTSs, only concepts and their instances are naturally specified by ontological domain-specific transition systems.

6. *The formalism provides a quite complete classification of ontological elements, including the determination of their new kinds and subkinds.* In contrast to CTSs, domain-specific transition systems do not allow to classify ontological elements and define their new kinds.
7. *The formalism is based on the conception 'state – transition' of the usual transition systems, keeping their simplicity and universality and adding a conceptual 'filling'.* Domain-specific transition systems are the special kind of transition systems in which transitions are defined by sets of transition rules, the quite general pattern matching algorithm which specifies applicability of these rules, and the specific rule execution algorithm, and states are functions on forms with specific attributes. Therefore, domain-specific transition systems do not keep simplicity and universality of the usual transition systems sufficiently.
8. *The formalism supports reflection of any order.* In contrast to CTSs, domain-specific transition systems do not support reflection.

10. Conclusion

In the paper, the main definitions of the theory of CTSs were given, classifications for elements of states of CTSs and for conceptuals and their associated ontological elements were developed, generalization of conceptuals which allows to make more comprehensive classification of ontological elements – generic conceptuals – was proposed.

We plan to develop the special kinds of CTSs concretizing the transition relation, and the language of specification of CTSs that would describe the transition relation concretizations which are important for practical purposes.

Development of formal methods based on this language to solve problems of designing and prototyping software systems as well as specification of operational and axiomatic semantics of programming languages is an important application of CTSs. In the case of specification of operational semantics of a programming language, a CTS specifies the abstract machine of the language. In the case of specification of axiomatic semantics of a programming language, a CTS specifies a generator of verification conditions for programs in the language, based on its axiomatic semantics.

We hope that the use of the ontological approach will reduce the gap between the great potential of formal methods and, with rare exceptions, toy examples of their application for solving the above problems [11].

Because of their simplicity and universality, CTSs can be also used for classification and

formalization of concepts, properties and algorithms in the field of ontology evolution [12–15], as well as designing and prototyping tools to support ontology evolution [16].

References

1. 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.
2. Gurevich Y. Evolving Algebras. Lipari Guide // Specification and Validation Methods. Oxford University Press, 1995. P. 9-36.
3. Anureev I.S. Operational Ontological Approach to Formal Programming Language Specification // Programming and Computer Software. 2009. Vol. 35. N 1. P. 35-42.
4. Gruber T.R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal Human-Computer Studies. 43(5-6). 1995. P. 907-928.
5. Anureev I.S. Ontological Transition Systems // Bulletin of the Novosibirsk Computing Center, Series Computer Science. 2007. Vol. 26. P. 1-17.
6. Anureev I.S. A Language of Actions in Ontological Transition Systems // Bulletin of the Novosibirsk Computing Center, Series Computer Science. 2007. Vol. 26. P. 19-38.
7. Anureev I.S. Domain-Specific Transition Systems and their Application to a Formal Definition of a Model Programming Language // Bulletin of the Novosibirsk Computing Center, Series Computer Science. 2014. Vol. 34. P. 23–42.
8. Huggins J. Abstract State Machines Web Page. URL: http://www.eecs.umich.edu/gasm (accessed: 01.09.2015).
9. AsmL: The Abstract State Machine Language. Reference Manual. 2002. URL: http://research.microsoft.com/fse/asml/doc/AsmL2 Reference.doc (accessed: 01.09.2015).
10. XasM — An Extensible, Component-Based Abstract State Machines Language. URL: http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html (accessed: 01.09.2015).
11. Parnas D.L. Really Rethinking Formal Methods // Computer. IEEE Computer Society. 2010. Vol. 43 (1). P. 28-34.
12. Haase P., Stojanovic L. Consistent Evolution of OWL Ontologies. In Proceedings of the 2nd European Semantic Web Conference. 2005.
13. Heflin J., Hendler J., Luke S.: Coping with Changing Ontologies in a Distributed Environment. In Proceedings of the Workshop on Ontology Management of the 16th National Conference on Artificial Intelligence. 1999. P. 74-79.
14. Noy N.F., Klein M.: Ontology Evolution: Not the Same as Schema Evolution. Knowledge and Information Systems. 6(4). 2004. P. 428-440.
15. Stojanovic L., Maedche A., Stojanovic N., Studer R.: Ontology Evolution as Reconfiguration-Design Problem Solving. In Proceedings of the 2nd International Conference on Knowledge Capture. 2003. P. 162-171.
16. Haase P., Sure Y.: D3.1.1.b State of the Art on Ontology Evolution. 2004. URL:

http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/SEKT-D3.1.1.b.pdf (accessed: 01.09.2015).

УДК 618.518.5

Требования к системе управления квадрокоптером

Тумуров Э.Г. (Институт систем информатики СО РАН),

Шелехов В.И. (Институт систем информатики СО РАН,

Новосибирский государственный университет)

Определяется типовая архитектура системы управления вида human-in-the-loop. Набор функциональных требований к системе управления беспилотным 4-х винтовым вертолетом - квадрокоптером – определяется как конкретизация типовой архитектуры. Рассматривается две задачи: управление полетом квадрокоптера по заданной траектории и слежение за движущимся объектом.

Ключевые слова: *Беспилотный летательный аппарат, квадрокоптер, human-in-the-loop, система управления, определение требований, автоматное программирование.*

1. Введение

Квадрокоптер – беспилотный 4-х винтовой вертолет. Для квадрокоптера предусмотрено два вида управления: ручное, осуществляемое человеком (оператором) дистанционно через Интернет, и автоматическое программное на базе интерфейса, определяющего систему команд управления квадрокоптером.

В настоящей работе определяется общая постановка задачи управления квадрокоптером, интегрирующая различные аспекты автоматического управления квадрокоптером [1-2, 9-10, 12, 14-15, 17-18, 20-21, 24-29, 31]. Вместе с традиционной задачей движения по заданному маршруту рассматривается новая задача слежения за движущимся объектом. Описывается типовая архитектура системы управления вида human-in-the-loop [19] с интеграцией автоматического и ручного управления. Спецификация задачи управления квадрокоптером определяется в виде набора требований.

Контроллер системы управления является реактивной системой. *Определение требований* – первый этап конструирования реактивной системы. *Требования* — совокупность утверждений относительно свойств разрабатываемой реактивной системы. Одним из видов требований являются *функциональные требования*, определяющие поведение программы. Их наиболее популярной формой являются *сценарии использования* (*use case*) [6]. В нашем подходе сценарии использования представлены в виде правил на языке продукций [8],

обычно применяемом для систем искусственного интеллекта. Это простой язык с высокой степенью декларативности. Спецификация автоматной программы в виде набора требований компактна и легко транслируется в автоматную программу, что позволяет использовать язык спецификации требований как язык автоматного программирования [4].

Типовая архитектура системы управления вида human-in-the-loop представлена в разделе 2 автоматными программами автоматического контроллера и мониторинга. Программы специфицированы набором требований. Язык требований кратко описывается в разд. 2.1; его полное описание доступно в [4]. Задача управления беспилотным летающим аппаратом (БПЛА) определена в общем виде в подразделе 3.1 как конкретизация типовой архитектуры. В свою очередь система управления квадрокоптером (подраздел 3.1.1) характеризуется конкретизацией входных и управляющих параметров системы управления БПЛА. Постановка задачи управления движением квадрокоптера по заданной траектории (подраздел 3.2) определяет задание маршрута движения, набор возможных ограничений и препятствий. В подразделе 3.3 описана постановка задачи слежения за движущимся объектом с определением параметров и разнообразных аспектов. В обзоре работ (раздел 4) описываются различные подходы в постановке задачи управления квадрокоптером.

Работа выполнена при поддержке РФФИ, грант № 12-01-00686.

2. Типовая архитектура системы управления

Контроллер системы управления является реактивной системой. Для спецификации и реализации реактивных систем разработана технология автоматного программирования [4]. *Автоматная программа* определяется в виде конечного автомата и состоит из нескольких *сегментов кода*. Вершина автомата – *управляющее состояние* программы. Ориентированная гипердуга автомата соответствует некоторому сегменту кода и связывает одну вершину с одной или несколькими другими вершинами. Исполнение сегмента завершается оператором перехода на начало другого сегмента. Взаимодействие автоматной программы с внешним окружением реализуется через прием и посылку *сообщений*. *Состояние* автоматной программы определяется значениями набора переменных, модифицируемых в программе.

Реактивная система реализует взаимодействие с внешним окружением программы и реагирует на определенный набор событий (сообщений) в окружении программы. Подклассом реактивных систем являются *гибридные системы*, соединяющие дискретное и непрерывное поведение. Часть переменных состояния гибридной системы соответствует непрерывным параметрам (время, координаты в пространстве и др.), изменение которых реализуется независимо от программы гибридной системы по определенным законам,

обычно формулируемым в виде дифференциальных уравнений. Важнейшим подклассом гибридных систем являются контроллеры систем управления.

Система управления реализует взаимодействие с *объектом управления* для поддержания его функционирования в соответствии с поставленной целью. Системы управления используются в аэрокосмической отрасли, энергетике, медицине, массовом транспорте и др. отраслях. На каждом шаге вычислительного цикла *контроллер системы управления* получает входную информацию из окружения и обрабатывает ее. Результаты вычисления используются для передачи управляющего сигнала, воздействующего на объект управления.

Большинство систем управления являются *системами реального времени*, определяющими набор мягких и жестких временных ограничений на взаимодействие с окружением. В системах с жестким ограничением непредоставление результатов вычислений к определенному сроку является фатальной ошибкой.

Наряду с управлением в автоматическом режиме возможно наличие *ручного управления*, осуществляемого человеком – *оператором*, иногда осуществляемого дистанционно через Интернет [29]. Чтобы ручное управление было эффективным, необходима реализация адекватной визуализации процесса управления. Автоматическое и ручное управление могут существовать раздельно без интеграции между собой. Определим типовую архитектуру системы управления вида human-in-the-loop [19], в которой автоматическое управление объектом интегрировано с ручным управлением. Контроллер системы управления состоит из двух параллельных процессов, определяемых автоматными программами AutoControl и Advisor. Архитектура системы управления представлена набором требований (разд. 2.2 и 2.3).

2.1. Язык требований

Требование определяет один из вариантов функционирования автоматной программы и имеет следующую структуру:

$$\langle\text{условие}_1\rangle, \langle\text{условие}_2\rangle, \ldots, \langle\text{условие}_n\rangle \rightarrow \langle\text{действие}_1\rangle, \ldots, \langle\text{действие}_m\rangle$$

Условиями являются: управляющие состояния, получаемые сообщения, логические выражения. *Действиями* являются: простые операторы, вызовы программ, посылаемые сообщения и итоговые управляющие состояния. Семантика требования следующая: если в данный момент времени истинны все условия в левой части требования, то последовательно исполняется набор действий в правой части.

Вызов программы записывается в виде A(x: y), где A – имя программы, x – список аргументов вызова, y – список результатов. Неблокированный прием сообщения реализуется

конструкцией m(y), где m – имя сообщения, y – список переменных. Значение конструкции – **true**, если из окружения получено сообщение с именем m; при этом переменным y присваиваются значения параметров сообщения m. Оператор перехода **goto** M записывается в виде #M.

Полное описание языка требований доступно в работе [4]. Описание требований контроллеров AutoControl и Advisor сопровождается содержательными пояснениями.

2.2. Автоматический контроллер

Контроллер AutoControl реализует циклический пошаговый процесс автоматического управления объектом. На каждом шаге контроллер определяет команду, которой нужно воздействовать на объект для достижения цели управления.

Состояние. Набор *переменных состояния* системы управления обозначается через s. В типичном случае состояние определяется: координатами объекта управления в пространстве, текущим временем и другими параметрами управления.

Окружение. Запуск контроллера, находящегося в спящем режиме, реализуется сообщением start. Срабатывание очередного шага работы контроллера происходит через сообщение next(in), где in – набор *входных параметров*, поступающих от объекта управления. Управляющее воздействие на объект управления реализуется посылкой сообщения control(com), где com – команда управления объектом с целью корректировки его поведения. Команда com определяет *вид команды* и набор параметров. Сообщение stop используется для перевода контроллера в спящий режим.

Управляющие состояния:

- idle – режим ожидания начала работы (спящий режим);
- run – основной рабочий режим.

Требования.

idle, start → run
run, next(in) → Step(in, s: s', com), control(com)
run, stop → idle

В управляющем состоянии idle контроллер ожидает сообщения start от процесса Advisor. При получении сообщения start контроллер переходит в основной режим – управляющее состояние run. В этом режиме ожидается сообщение next(in), запускающее программу Step. Она анализирует информацию in о текущем состоянии объекта управления и вычисляет управляющую команду com для модификации поведения объекта в соответствии с целями

управления. В случае перехода на режим ручного управления посылается сообщение stop, переводящее контроллер в спящий режим.

Приведенные требования транслируются в следующую автоматную программу:

```
Программа. process AutoControl(…) {
              idle: if (start) #run
                    else #idle
              run:  if (next(in)) { Step(in, s: s', com);
                                    send control(com);
                    } else if (stop) #idle
                    #run
            }
```

Сообщение next(in) обычно генерируется через определенный фиксированный промежуток времени с последними значениями параметров in, регулярно поставляемыми от объекта управления.

Задачей программы Step является достижение целевых значений параметров par объекта управления. По значениям in и par требуется вычислить соответствующие значения параметров команды com. Обычно это реализуется решением дифференциальных уравнений E(in, par, com), определяющих закон функционирования управляемого объекта. В дополнении к этому, возможны случайные воздействия на объект управления – *возмущения*; например, ветер. Информация о возникающих возмущениях может накапливаться в состоянии s в целях адекватной реакции.

Система управления должна быть *устойчивой*: при отсутствии возмущений ее параметры in должны приближаться к требуемым значениям par, а при ограниченных возмущениях параметры in должны оставался в ограниченной зоне вблизи этих значений [28]. Свойство устойчивости является подвидом свойства безопасности реактивной системы.

2.3. Мониторинг процесса управления

Автоматическое и ручное управление может проводиться одновременно лишь в очень редких случаях. Переход с автоматического управления на ручное предполагает отключение автоматического управления. Ручное управление блокируется при переходе на автоматическое управление, однако визуализация процесса управления для оператора сохраняется. Переключение с одного режима на другой поддерживается процессом Advisor, который реализует общий мониторинг процесса управления, работая параллельно с автоматическим контроллером AutoControl.

Решение о переходе на ручное управление принимает оператор на основе данных визуализации процесса управления. Функцией контроллера Advisor является оценка степени

расхождения между плановым и фактическим функционированием объекта с определением желательности перехода на ручное управление. Отметим, что фиксация аварийного или катастрофического состояния должна быть упреждающей с тем, чтобы оператор имел время проанализировать угрожающую ситуацию для проведения адекватной модификации управления. Прогностический характер оценки процесса управления предполагает анализ истории функционирования объекта на определенном промежутке времени.

Одной из функций процесса Advisor является обеспечение безопасности от постороннего вмешательства в процесс управления применением современных методов криптографии, т.е. аутентификации, шифрования посылаемой через Интернет информации и обеспечение ее целостности (защиту от модификации).

Окружение. Запуск контроллера системы управления реализуется сообщением init(passw, task), где passw – пароль для доступа в систему управления, task – исходное задание на процесс функционирования объекта управления. Сообщение show(viz) передает дополнительные данные viz для визуализации оператору результатов анализа процесса управления контроллером Advisor.

Состояние. Набор переменных состояния s тот же самый, что и для процесса AutoControl.

Управляющие состояния: a0, a1, a2.

Требования.

a0, init(passw, task) → Launch(passw, task**:** s #a1**:** "wrong password" #a0)
a1 → start, a2
a2 → Monitoring(s**:** s', viz), show(viz)

В управляющем состоянии a0 контроллер ожидает сообщения init(passw, task), которое генерируется при начальном запуске системы управления оператором. При получении сообщения контроллер запускает программу-гиперфункцию Launch, которая проверяет правильность пароля passw, набранного оператором. Если пароль passw не совпадает с хранящимся в контроллере, то выдается сообщение оператору о неправильном пароле с предложением ввести его снова, и управление возвращается в состояние a0. При правильном пароле контроллер переносит информацию о задании task в переменные состояния s и переходит в управляющее состояние a1.

В управляющем состоянии a1 контроллер посылает сообщение start процессу AutoControl, тем самым переводя его из спящего состояние в рабочее, и переходит в управляющее состояние a2, в котором реализуется мониторинг системы управления. Программа Monitoring(s**:** s', viz) сравнивает плановое и фактическое функционирование

объекта управления, оценивает удовлетворение существующих ограничений и определяет возможность аварийных сценариев. Результат работы viz поставляет данные для визуализации оператору в дополнении к существующей визуализации процесса управления.

Программа. **process** Advisor(…) {

```
a0:  if (init(passw, task)) Launch(passw, task: s #a1: "wrong password"#a0)
     else #a0
a1:  start;
a2:  Monitoring(s: s', viz);
     show(viz);
     #a2
}
```

Определенная выше типовая архитектура системы управления является простейшей. Рост сложности возможен по нескольким направлениям: числу управляющих состояний, набору компонент системы, степени их параллельного взаимодействия. В системах реального времени запуск программы Step обычно реализуется независимой подзадачей. Детальное описание полного набора требований для реальных больших сложных систем занимает несколько сот страниц.

3. Постановка задачи управления квадрокоптером

3.1. Беспилотные летательные аппараты

Беспилотный летательный аппарат (БПЛА) - летательный аппарат без экипажа на борту. Управляется либо автономной бортовой системой, либо человеком дистанционно. Типы аппаратов: самолет, вертолет или ракета.

Области применения БПЛА: аэрофотосъемка, разведка, боевое применение, охрана сельхозугодий, картография, дистанционный химико-физический анализ, контроль всхожести и спелости урожая, химическая обработка, исследования птиц и животных, исследование стратосферы, работа в радиационных и токсичных зонах, поиск выживших, доставка лекарств в труднодоступные места и др. Производители БПЛА в России: Вега, Текнол, Zala, Иркут, Транзас, Аэрокон, Новик-XXI век и др.

Входные параметры (параметры in в обозначениях разд. 2.2) определяют положение и ориентацию аппарата: координаты x, y, z центра масс аппарата в неподвижной декартовой системе координат; вектор скорости v_x, v_y, v_z; вектор ускорения a_x, a_y, a_z; углы Эйлера (Тэйта-Брайана, корабельные углы) ψ, φ, θ; угловые скорости ψ', φ', θ'; угловые ускорения ψ'', φ'', θ''. С аппаратом жестко связана локальная система координат (X, Y, Z). Ее начало совпадает с центром масс

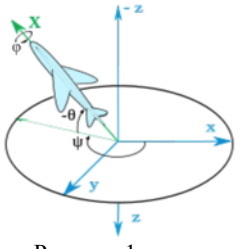


Рисунок 1 – углы Эйлера (Тэйта-Брайана)

аппарата. Ось X направлена вдоль продольной оси, Y – вдоль поперечной, Z – вдоль вертикальной. Чтобы перейти из неподвижной системы координат в локальную, нужно сначала сделать поворот вокруг оси z на *угол рысканья* ψ $(-\infty;+\infty)$, потом вокруг оси Y на *угол тангажа* θ $(-\pi/2; \pi/2)$, и, наконец, вокруг оси X на *угол крена* φ $(-\pi/2; \pi/2)$.

Для определения положения и ориентации аппарата используется малогабаритная инерциальная интегрированная система, имеющая в своем составе инерциальные датчики (микромеханический гироскоп и акселерометр), а также барометрический высотомер и трехосный магнитометр. Интегрируя данные этих датчиков с данными приемника GPS / ГЛОНАСС, система вырабатывает полное навигационное решение по координатам и углам ориентации.

Управляющие параметры (com в обозначениях разд. 2.2) определяют команду и ее параметры для установки требуемых показателей ориентации и скорости БПЛА.

3.1.1. Квадрокоптеры

Мультикоптер – вертолет с количеством несущих винтов более двух. Мультикоптеры с количеством винтов больше 4 используются, как правило, для повышения мощности, а также для отказоустойчивости – такие аппараты могут продолжить движение либо совершить мягкую посадку в случае отказа нескольких винтов.

Квадрокоптеры – мультикоптеры с четырьмя несущими винтами – получили широкое распространение в качестве гражданских легких недорогих аппаратов. В них обычно используются простые винты с *постоянным шагом* – углом наклона лопастей. Каждый винт управляется собственным двигателем. Два винта вращаются по часовой стрелке, два других – против часовой. Повороты реализуются изменением скорости вращения винтов, что ограничивает маневренность из-за инерции. Квадрокоптеры, как и другие вертолеты, способны двигаться в любую сторону, зависать на месте, осуществлять вертикальный взлет и посадку.

Ведущие лаборатории по изучению динамики и систем управления квадрокоптеров Aerospace Controls Lab of MIT, Flying machine arena of ETH Zurich, GRASP Lab of University of Pennsylvania проводят исследования оптимального управления экстремальных маневров, крутых виражей [27], управлением группой квадрокоптеров. Исследуются экспериментальные модели с изменяемым шагом винтов [20], обладающих лучшей маневренностью.

Входные параметры квадрокоптера AR.DRONE: x, y, z, ψ, φ, θ, v_{xb}, v_{yb}. Вместо v_x, v_y используются v_{xb}, v_{yb} – проекции вектора скорости (v_x, v_y, v_z) на локальные оси X, Y.

Остальные параметры те же, что и для БПЛА. Аппарат 200 раз в секунду передает параметры на устройство дистанционного управления. Значения координат x, y, z поступают от внешних камер, либо вычисляются бортовой инерциальной системой [26] или с помощью компьютерного зрения [28].

Управляющие параметры квадрокоптера AR.DRONE: φ, θ, ψ', v_z. Значения этих параметров задаются в ограниченных диапазонах. Управляющие параметры трансформируются в команду управления квадрокоптером. Аппарат может принимать команды управления каждые 30мс.

3.2. Задача управления полетом квадрокоптера по заданному маршруту

Маршрут полета квадрокоптера определяется набором *контрольных точек* $P_1, P_1, \ldots, P_n$ в пространстве координат x, y, z; где P_1 – начальная точка маршрута, а P_n – конечная точка. На первом этапе перед началом полета строится *траектория полета* – непрерывная кривая $T(u)=(x(u), y(u), z(u)), u\in[0,1]$, проходящая по контрольным точкам, причем $T(0)=P_1$ и $T(1)=P_n$. В случае, когда для каждой контрольной точки дополнительно фиксируется время ее прохождения, траектория принимает вид: $T(t)=(x(t), y(t), z(t)), t\in[0,t_{max}]$. Простейшая траектория – ломаная линия не оптимальна по времени, поскольку в контрольных точках аппарат будет вынужден полностью останавливаться. Оптимальные по времени траектории – гладкие кривые, построенные с учетом маневренности квадрокоптера.

Построение траектории полета обычно совмещается с планированием полета; например, определяется скорость движения на разных участках маршрута. В критических ситуациях построенная траектория может корректироваться в процессе полета [21].

Задача *управления полетом по траектории* часто ставится при наличии ограничений различного рода. Это могут быть как общие ограничения по времени, скорости, угловой скорости, высоте и т.д; так и ограничения в контрольных точках по времени, скорости [12] и ориентации аппарата. Более серьезными являются пространственные ограничения. Их нарушения оказываются фатальными. Требуется аккуратное планирование, гарантирующее полет на достаточном удалении от препятствий с запасом на возможные возмущения. Неподвижные препятствия могут задаваться в виде трехмерных объектов [12, 15, 24, 31]. Иногда фиксируется допустимый коридор отклонения траектории от ломаной, соединяющей контрольные точки [9]. Подвижные препятствия могут быть представлены трехмерными объектами с заданным законом движения [9]. В общем случае положение подвижного объекта необходимо непрерывно отслеживать в процессе полета в целях корректировки траектории полета.

Дополнительные трудности возникают при длительной потере сигнала со спутников, обслуживающих приемники GPS / ГЛОНАСС, когда при навигации лишь через инерциальные датчики [26] постепенно накапливаются ошибки в определении координат местоположения. Актуальной становится задача автоматического ориентирования в пространстве с использованием средств компьютерного зрения [28].

На очередной шаге управления полетом программа Step по входным параметрам in (координатам, ориентации, скорости и угловой скорости квадрокоптера) вычисляет управляющие параметры com в целях изменения скорости и угловой скорости для достижения целевых параметров par (координат квадрокоптера через определенное время). В качестве par может выступать одна из точек на траектории полета. Возможны разные стратегии выбора такой точки как вблизи квадрокоптера, так и на достаточно большом удалении от него. Входные параметры in посылаются квадрокоптером в некоторый момент времени t_0, а управляющие команды com срабатывают на квадрокоптере в другой момент времени t_1, когда его параметры уже другие. В принципе, значение параметров во время t_1 можно было бы вычислить, используя уравнения движения квадрокоптера [9, 15, 24], однако на практике такое редко реализуется, что порождает т.н. ошибку инерции. Сопоставляя фактические координаты с рассчитанными для момента времени t_1 и запоминая эту информацию в параметрах состояния s можно эффективно прогнозировать характер возмущений в целях внесения соответствующих компенсирующих поправок.

3.3. Задача слежения за движущимся объектом

Задача слежения за движущимся объектом заключается в сопровождении объекта квадрокоптером на фиксированном расстоянии от него [7, 16, 23, 30]. Предполагается, что объект перемещается по земле. К квадрокоптеру прикреплена видео камера на вращающемся подвесе. Задача – реализовать управление квадрокоптером и подвесом, чтобы объект всегда был в кадре видеокамеры. Варианты слежения:

1. «Вид от третьего лица»: следование за объектом сверху-сзади. Параметры: расстояние до объекта R, относительная высота H.
2. «Вид сбоку»: следование за объектом со стороны. Параметры: направление по компасу A, расстояние R, относительная высота H.
3. «Вращение»: квадрокоптер вращается вокруг объекта. Параметры: период вращения T, расстояние R, относительная высота H.

Слежение завершается:

- при достижении заданного местоположения;
- после истечения заданного времени полета;
- после получения команды с пульта управления.

По завершении слежения квадрокоптер должен мягко сесть на землю.

Развернутая спецификация задачи слежения должна учитывать множество аспектов. Некоторые из них перечислены ниже.

Модель камеры и подвеса:

- Углы обзора кадра: горизонтальный $F\psi$ и вертикальный $F\theta$.
- Углы ориентации подвеса: $G\psi$, $G\varphi$, $G\theta$.

Подсистема локации объекта. Положение объекта относительно квадрокоптера определяется параметрами: направлением и расстоянием. Подсистема локации может быть реализована одним из способов:

- в виде системы компьютерного зрения;
- ближний радар и радиомаяк (например, RFID);
- объект имеет высокоточный GPS приемник.

Предсказание траектории объекта. Необходимо предсказывать траекторию объекта. Подвижность традиционных объектов видеосъемки – автомобилей, мотоциклов, танков и другой наземной техники, лыжников, сноубордистов, велосипедистов, а также ускорения и угловые ускорения объекта – ограничены.

Несмотря на то, что объект передвигается по земле, он может совершать прыжки в несколько метров (лыжник, мотоциклист). Также следует учитывать перепады высоты ландшафта.

Аварийные случаи. Система управления должна уметь определить аварийную ситуацию и предпринять меры для безопасного прекращения работы. Примеры аварийных ситуаций: потеря связи, сильный ветер, низкий заряд батареи, техническая неисправность. Безопасное прекращение работы обычно включает в себя мягкую посадку в определенной зоне, уведомление оператора о неисправности и местоположении аппарата.

Система управления квадрокоптером соединяет в себе элементы жесткого и мягкого реального времени. Так как траектория движения объекта заранее не известна, система управления должна планировать и корректировать траекторию движения квадрокоптера и подвеса камеры на несколько секунд вперед. При этом необходимо учитывать препятствия – горы, деревья, здания.

Качество работы системы управления определяется параметрами:

- минимальным отклонением от целевых параметров (расстояние до объекта, высота и тд);
- нахождение объекта как можно ближе к центру кадра;
- плавность траектории БПЛА;
- плавность перемещения объекта в полученном видео;
- правильный «горизонт» в кадре.

4. Обзор работ

Интеграция автоматического и ручного управления в стиле human-in-the-loop [19] реализована в автомобильной, авиационной и космической отраслях. Определены пять уровней автоматизации вождения автомобиля от ручного до полностью автоматического [22]. Переход на тотальное автоматическое управление всего потока автомобилей планируется с 2020г. в рамках грандиозных программ США и Европы по развитию автомобильной отрасли [11, 22]. Отметим, что ручное и автоматическое управление квадракоптеров реализовано независимо. Их интеграция изначально не планировалась. В работах по системам управления квадрокоптерами просматриваются лишь отдельные элементы такой интеграции.

Движение квадрокоптера описывает плоская система дифференциальных уравнений [9, 15, 24], позволяющая заменить нелинейную систему уравнений линеаризованной системой в окрестности положения равновесия [12, 17, 26]. Для линейной системы строится асимптотически устойчивое решение с ограничением на угол крена. При больших углах крена линеаризованная модель дает значительную ошибку.

Нелинейные системы решаются методами обратного шага (back-stepping) нелинейного программирования [9, 10, 18, 25].

Для построения траектории для исходного маршрута полета строится гладкая кривая, заданная полиномиальными сплайнами. Далее траектория параметризуется временем так, чтобы траектория была осуществима, а общее время полета стало минимальным [12, 15, 31].

Планирование оптимальных траекторий с учетом препятствий реализуется методом графа видимости [12], быстрым маршевым методом [12] или алгоритмом RRT* [24].

Исследования по управлению квадрокоптером AR.DRONE проводились в Институте автоматики и электрометрии СО РАН (г. Новосибирск) [1, 2]. Построена модель бортовой системы управления, система управления и измерения положения в пространстве, метод фильтрации шумов с помощью фильтра Калмана.

Слежение за подвижными наземными целями с помощью бортовых видеокамер БПЛА активно изучается [7, 16, 23, 30]. В 2015 г. планируется выпуск коммерческих квадрокоптеров для личного пользования с функцией видеосъемки и слежения за объектом «AirDog» и «Hexo+».

В мировой практике технология определения требований разрабатывается и применяется в основном для больших интернетовских информационных систем и систем телекоммуникации, а также в системной инженерии (системотехнике). Технология спецификации требований отражена в стандарте [13]. В нашем подходе определение требований рассматривается для всего класса реактивных систем.

5. Заключение

Разработка программного обеспечения, реализующего системы управления БПЛА в различных приложениях, становится все более популярной отраслью; см., например, [5]. Актуальной становится не только разработка отдельных методов управления БПЛА, но и исследование постановки задачи управления БПЛА в целом с анализом всего множества разнообразных аспектов, таких как интеграция автоматического и ручного управления или обеспечение безопасности от постороннего вмешательства.

Настоящая работа возникла в рамках исследований по технологии автоматного программирования [4]. Цель работы – сформулировать задачу построения произвольной системы управления на примере задачи управления квадракоптером. Данный подход предопределил метод построения типовой архитектуры системы управления с последующей ее специализацией для задач управления БПЛА. Анализ методов построения для других подклассов систем управления, особенно для автомобильной отрасли, позволил обнаружить архитектуру систем управления вида human-in-the-loop [19] как базисную для многих подклассов систем управления, в т.ч. и для систем управления квадракоптерами, несмотря на то, что в настоящее время не наблюдается интеграции автоматического и ручного управления квадрокоптерами. Однако есть свидетельства того, что такая интеграция скоро состоится. Для этого в частности необходимо будет разработать методы предсказания поведения БПЛА на базе анализа предыдущей истории. Сходные методы будут также применимы и для компенсации возмущений.

Список литературы

1. Белоконь С.А., Золотухин Ю.Н., Мальцев А.С., Нестеров А.А., Филиппов М.Н., Ян А.П. Управление параметрами полёта квадрокоптера при движении по заданной траектории // Автометрия, 2012. С. 32-42.
2. Белоконь С.А., Золотухин Ю.Н., Котов К.Ю., Мальцев А.С., Нестеров А.А., Филиппов М.Н., Ян А.П. Управление квадрокоптером AR.DRONE при движении по заданной траектории // Проблемы управления и моделирования в сложных системах: Труды XV международной конференции, 25-28 июня 2013 г., Самара, С. 506-514.
3. Новиков Д.А. Теория управления организационными системами / МПСИ, Москва. 2005. 584 с.
4. Шелехов В.И. Разработка автоматных программ на базе определения требований // Системная информатика, №4, 2014. ИСИ СО РАН, Новосибирск. С. 1-29. URL: http://persons.iis.nsk.su/files/persons/pages/req_tech.pdf (дата обращения: 24.11.2015)
5. “Хозяин, напиши для нас приложение”. Требуется разработчик софта и железа для дронов DJI. // Блог “Хакспейс Neuron” на сайте habrahabr.ru. URL: http://habrahabr.ru/company/neuronspace/blog/259527/ (дата обращения: 20.08.2015).
6. A. Cockburn. Writing effective use cases / Addison-Wesley. 2001. 270 P.
7. C. Teuliere, L. Eck, and E. Marchand. Chasing a moving target from a flying uav. // IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.
8. D. Klahr, P. Langley, R. Neches. Production system models of learning and development. Cambridge, Mass.: The MIT Press. 1987. 467 P.
9. D. Mellinger, V. Kumar. Minimum snap trajectory generation and control for quadrotors // Robotics and Automation (ICRA), 2011 IEEE International Conf.
10. D. Xu, L. Wang, G. Li, L. Guo. Modeling and trajectory control of a quad-rotor UAV // ICCASM-12, Advances in Intelligent Systems Research, July 2012.
11. European Roadmap Smart Systems for Automated Driving. Version 1.2. Berlin, April 1st, 2015, 39 P.
12. G.M. Hoffman, S.L. Waslander, C.J. Tomlin. Quadrotor helicopter trajectory tracking control // In proc. AIAA Guidance, Navigation and Control Conf, 2008.
13. IEEE recommended practice for software requirements specifications. Revision: 29/Dec/11.
14. I.D. Cowling, J.F. Whidborne, A.K. Cooke. Optimal trajectory planning and LQR control for a quadrotor UAV. // in International Control Conference, 2006.

15. I.D. Cowling, O.A. Yakimenko, J.F. Whidborne. A prototype of an autonomous controller for a quadrotor UAV // In European Control Conference, 2007.
16. J. Xiao, C. Yang, F. Han, H. Cheng, and Sarnoff Corporation. Vehicle and person tracking in aerial videos. // Multimodal Technologies for Perception of Humans, 2008. P. 203–214.
17. K. Miller. Path tracking control for quadrotor helicopters, 2008.
18. L. Lai, C. Yang, C. Wu. Time-optimal control of a hovering quad-rotor helicopter // Journal of Intelligent and Robotic Systems, 45(2), June 2006. P. 115-135.
19. Li W., Sadigh D., Sastry S., Seshia S. Synthesis for human-in-the-loop control systems // Tools and Algorithms for the Construction and Analysis of Systems. LNCS 8413, 2014, P. 470–484.
20. M. Cutler, J.P. How. Actuator constrained trajectory generation and control for variable-pitch quadrotors, 2012.
21. M. Hehn, R. D'Andrea. Quadrocopter trajectory generation and control // IFAC World Congress, 2011, P. 1485-1491.
22. National Highway Traffic Safety Administration. Preliminary statement of policy concerning automated vehicles. May 2013, 14 P.
23. P. Theodorakopoulos and S. Lacroix. Uav target tracking using an adversarial iterative prediction. // IEEE International Conference on Robotics and Automation, 2009.
24. R. Charles, A. Bry, N. Roy. Polynomial trajectory planning for quadrotor flight // Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2013.
25. R. Cunha, D. Cabecinhas, C. Silvestre. Nonlinear trajectory tracking control of a quadrotor vehicle. // In Proc. European Control Conference, 2009.
26. R.W. Beard. Quadrotor dynamics and control // Brigham Young University, Feb 2008
27. S. Lupashin, A. Schollig, M. Sherback, and R. D'Andrea. A simple learning strategy for high-speed quadro-copter multi-flips. // IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2010. P. 1642–1648.
28. S. Shen, Y. Mulgaonkar, N. Michael, V. Kumar. Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor // Proceedings of Robotics: Science and Systems IX, June 2013.
29. S.H. Yang, L.S. Tan, and X. Chen. Requirements specification and architecture design for Internet-based control systems // Computer Software and Applications Conference, 2002. Proceedings. 26th Annual International, 26-29 Aug. 2002. P. 75-80.
30. S. Zhang. Object tracking in unmanned aerial vehicle (uav) videos using a combined approach // IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.

31. Y. Bouktir, M. Haddad, T. Chettibi. Trajectory planning for a quadrotor helicopter // In Mediterranean Conference on Control and Automation. IEEE, June 2008.

УДК 004.8

Kinds and language of conceptual transition systems*

Anureev I.S. (Institute of Informatics Systems)

The language CTSL of specification of conceptual transition systems which are a formalism for description of dynamic discrete systems on the basis of their conceptual structure is proposed. The basic kinds of conceptual transition systems are considered. The basic predefined elements and operations of the CTSL language are defined.

Keywords: transition systems, conceptual structures, ontologies, ontological elements, conceptual transition systems, conceptuals, CTSL

1. Introduction

Development of formalisms, languages and tools for describing the conceptual structure of various systems is an important problem of the modern knowledge industry. Description of changes of the conceptual structure of the system when it functions is an another important problem.

The formalism of description (specification) of systems – conceptual transition systems (CTSs) – that solves these problems was proposed in [1]. To our knowledge, CTSs are the only formalism which meets the following requirements (as is shown in [1]):

1. It describes the conceptual structure of the specified system.
2. It describes the content of the conceptual structure of the specified system, i. e. it describes the specified system in the context of the conceptual structure.
3. It describes the change of the conceptual structure of the specified system.
4. It describes the change of the content of the conceptual structure of the specified system, i. e. it describes the change of the specified system in the context of the conceptual structure.
5. It is quite universal to specify typical ontological elements (concepts, attributes, concept instances, relations, relation instances, individuals, types, domains, and so on.).
6. It provides a quite complete classification of ontological elements, including the determination of their new kinds and subkinds.
7. It is based on the conception 'state – transition' of the usual transition systems, keeping

*Partially supported by RFBR under grants 15-01-05974 and 15-07-04144.

their simplicity and universality and adding a conceptual 'filling'. This requirement is important since the simplicity of determination of transition systems makes them an universal formalism to describe the behavior of various systems (algorithms, programs, software models, computer systems, and so on.).

8. It supports reflection of any order, i. e. allows to specify: the system (reflection of the order 0), the specification of the system (reflection of the order 1), the specification of the specification of the system (reflection of the order 2) and so on. Specifications of the higher order (with reflection of the higher order) impose restrictions on the specifications of the lower order (with reflection of the lower order).

Comparison of CTSs with the formalisms such that abstract state machines [2, 3], ontological transition systems [5, 6] and domain-specific transition systems [7] which partially meet these requirements was made in [1].

In contrast to abstract state machines [11, 12] and ontological transition systems [6], there is no specification language which describes CTSs. The language CTSL (Conceptual Transition System Language) of specification of CTSs is defined in this paper.

Contrary to state which has the detailed conceptual structure in CTSs, the transition relation in CTSs is quite general. Kinds of CTSs which concretize the transition relation are considered in this paper. They are defined in CTSL that thus describes concretizations of the transition relation which are important in practice.

The paper has the following structure. The preliminary concepts and notation are given in section 2. The main constructs of the CTSL language are described in section 3. The basic kinds of CTSs, predefined elements and operations in CTSL are defined in sections 4, 5 and 6, respectively.

2. Preliminaries

Let *bool* = *{true, false}*; *int*, *nat* and *nat0* denote the sets of integers, natural numbers and natural numbers with zero, respectively; *obj*, *fun*, *set*, *lab*, *arg*, and *val* denote sets of objects, functions, sets, labels, function arguments and function values, respectively.

The names of the variables which take the values from the set with the name aw, where a is a symbol, and w is a word, are denoted by $\dot{a}w$, $\dot{a}w_1$, $\dot{a}w'$ and so forth. For example, $\dot{s}et$, $\dot{s}et_1$, $\dot{s}et'$ are the names of the variables which take the values from the set *set*. Depending on the context, the name of a variable is interpreted as either the variable, or the value of the variable.

Let $sup(\dot{fun})$ and ω denote the support of $\dot{fun}$ and the indeterminate value of $\dot{fun}$, respectively.

Let $\dot{fun}(\dot{arg}_1 \leftarrow \dot{val}_1, \ldots, \dot{arg}_{\dot{nat}} \leftarrow \dot{val}_{\dot{nat}})$ denote the function $\dot{fun}'$ such that $\dot{fun}'(\dot{arg}) = \dot{fun}(\dot{arg})$, if $\dot{arg}$ is distinct from $\dot{arg}_1$, ..., $\dot{arg}_{\dot{nat}}$, and $\dot{fun}'(\dot{arg}_{\dot{nat}'}) = \dot{val}_{\dot{nat}'}$, if $1 \leq \dot{nat}' \leq \dot{nat}$.

Let $\{\dot{arg}_1{:}\dot{val}_1, \ldots, \dot{arg}_{\dot{nat}}{:}\dot{val}_{\dot{nat}}\}$ denote the function $\dot{fun}$ such that $sup(\dot{fun}) = \{\dot{arg}_1, \ldots, \dot{arg}_{\dot{nat}}\}$, and $\dot{fun}(\dot{arg}_1) = \dot{val}_1$, ..., $\dot{fun}(\dot{arg}_{\dot{nat}}) = \dot{val}_{\dot{nat}}$. The arguments $\dot{arg}_1$, ..., $\dot{arg}_{\dot{nat}}$ are pairwise distinct.

The terms used in the paper are context-dependent. Contexts have the form $[\![\dot{obj}_1, \ldots, \dot{obj}_{\dot{nat}}]\!]$, where the embedded contexts $\dot{obj}_1$, ..., $\dot{obj}_{\dot{nat}}$ have the form: $\dot{lab}{:}\dot{obj}$, $\dot{lab}{:}$ or $\dot{obj}$.

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 $\dot{obj}[\![\dot{obj}_1, \ldots, \dot{obj}_{\dot{nat}}]\!]$ denote the object $\dot{obj}$ in the context $[\![\dot{obj}_1, \ldots, \dot{obj}_{\dot{nat}}]\!]$.

Let *cts* denote a set of conceptual transition states [8]. Let *ato*, *ele*, *eleStr*, *ordStr* and *sta* denote sets of atoms, elements, element structures, ordered structures and conceptual states in $[\![\dot{cts}]\!]$.

Let *samp*, *body*, *cond* and *var* be sets of elements called samples, bodies, conditions and variables, respectively. Let *varSet* be a set of unsorted structures. Elements of $\dot{varSet}$ are called variables.

3. The CTSL language

The CTSL language is a language of specification of CTSs. Atoms, elements and the transition relation are key notions of CTSL.

3.1. Atoms and elements

Atoms in CTSL represent atoms of CTSs which are specified by CTSL. An object $\dot{obj}$ is called an atom in CTSL, if

- either $\dot{obj}$ is a sequence of Unicode symbols except for the whitespace symbols and the symbols *″*, *’ }*, *}*, *(*, *)*, *[*, *]*, *:*, *;* and *,*;
- or $\dot{obj}$ has the form *″*$\dot{obj}_1$*″*, where $\dot{obj}_1$ is a sequence of Unicode symbols in which each occurrence of the character *″* is preceded by the symbol *’*.

Elements in CTSL represent elements of CTSs which are specified by CTSL. They are defined as in CTSs [1] except that the whitespace symbols and the semicolon are element delimiters along with comma. For example, $(\dot{e}le_1,\ \dot{e}le_2)$, $(\dot{e}le_1;\ \dot{e}le_2)$ and $(\dot{e}le_1\ \ \dot{e}le_2)$ represent the same element.

The definition of the transition relation in CTSL uses the notion of substitution.

3.2. Substitutions

A function $\dot{f}un \in ele \to ele$ is called a substitution. Let sub be a set of substitutions.

A function $subF \in ele \to ele$ is called a substitution function in $[\![\dot{s}ub]\!]$, if the following properties hold:

- if $\dot{e}le \in sup(\dot{s}ub)$, then $subF[\![\dot{s}ub]\!](\dot{e}le) = \dot{s}ub(\dot{e}le)$;
- if $\dot{e}le' \in sup(\dot{s}ub)$, and $\dot{s}ub(\dot{e}le') = (\dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0})$, then
$$subF[\![\dot{s}ub]\!](\dot{e}le\ .::\ \dot{e}le') = (subF[\![\dot{s}ub]\!](\dot{e}le),\ \dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0}));$$
- if $\dot{e}le' \in sup(\dot{s}ub)$, and $\dot{s}ub(\dot{e}le') \notin ordStr \cup \{()\}$, then $subF[\![\dot{s}ub]\!](\dot{e}le\ .::\ \dot{e}le') = [\![\omega]\!]$;
- if $\dot{e}le' \in sup(\dot{s}ub)$, and $\dot{s}ub(\dot{e}le') = (\dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0})$, then
$$subF[\![\dot{s}ub]\!](\dot{e}le'\ ::.\ \dot{e}le) = (\dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0}),\ subF[\![\dot{s}ub]\!](\dot{e}le));$$
- if $\dot{e}le' \in sup(\dot{s}ub)$, and $\dot{s}ub(\dot{e}le') \notin ordStr \cup \{()\}$, then $subF[\![\dot{s}ub]\!](\dot{e}le'\ ::.\ \dot{e}le) = [\![\omega]\!]$;
- if $\dot{e}le' \in sup(\dot{s}ub)$, $\dot{s}ub(\dot{e}le') = \{\dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0}\}$, and $\mathrm{subF}[\![\dot{s}ub]\!](\dot{e}le),\ \dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0}$ are pairwise distinct, then $subF[\![\dot{s}ub]\!](\dot{e}le\ .::u\ \dot{e}le') = \{subF[\![\dot{s}ub]\!](\dot{e}le),\ \dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0}\}$;
- if $\dot{e}le' \in sup(\dot{s}ub)$, and $\dot{s}ub(\dot{e}le') \notin unoStr \cup \{\{\}\}$, then $subF[\![\dot{s}ub]\!](\dot{e}le\ .::u\ \dot{e}le') = [\![\omega]\!]$;
- if $\dot{e}le' \in sup(\dot{s}ub)$, $\dot{s}ub(\dot{e}le') = \{\dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0}\}$, and $\mathrm{subF}[\![\dot{s}ub]\!](\dot{e}le),\ \dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at_0}$ are not pairwise distinct, then $subF[\![\dot{s}ub]\!](\dot{e}le\ .::u\ \dot{e}le') = [\![\omega]\!]$;
- if $\dot{a}to \notin sup(\dot{s}ub)$, then $subF[\![\dot{s}ub]\!](\dot{a}to) = \dot{a}to$;
- if $() \notin sup(\dot{s}ub)$, then $subF[\![\dot{s}ub]\!](()) = ()$;
- if $\{\} \notin sup(\dot{s}ub)$, then $subF[\![\dot{s}ub]\!](\{\}) = \{\}$;
- if $\dot{o}rdStr \in sup(\dot{s}ub)$, and $\dot{o}rdStr$ has the form $(\dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at})$, then
$$subF[\![\dot{s}ub]\!](\dot{o}rdStr) = (subF[\![\dot{s}ub]\!](\dot{e}le_1),\ \ldots,\ subF[\![\dot{s}ub]\!](\dot{e}le_{\dot{n}at}));$$
- if $\dot{u}noStr \in sup(\dot{s}ub)$, and $\dot{u}noStr$ has the form $\{\dot{e}le_1,\ \ldots,\ \dot{e}le_{\dot{n}at}\}$, then
$$subF[\![\dot{s}ub]\!](\dot{u}noStr) = \{subF[\![\dot{s}ub]\!](\dot{e}le_1),\ \ldots,\ subF[\![\dot{s}ub]\!](\dot{e}le_{\dot{n}at})\};$$
- if $\dot{l}abStr \in sup(\dot{s}ub)$, and $\dot{l}abStr$ has the form $\{\dot{l}ab_1{:}\dot{e}le_1,\ \ldots,\ \dot{l}ab_{\dot{n}at}{:}\dot{e}le_{\dot{n}at}\}$, then
$$subF[\![\dot{s}ub]\!](\dot{l}abStr) =$$
$$\{subF[\![\dot{s}ub]\!](\dot{l}ab_1){:}subF[\![\dot{s}ub]\!](\dot{e}le_1),\ \ldots,\ subF[\![\dot{s}ub]\!](\dot{l}ab_{\dot{n}at}){:}subF[\![\dot{s}ub]\!](\dot{e}le_{\dot{n}at})\}.$$

An element $\dot{e}le$ is called an instance in $[\![\dot{e}le',\ \dot{s}ub]\!]$, if $subF[\![\dot{s}ub]\!](\dot{e}le') = \dot{e}le$. An element $\dot{e}le'$ is called a sample in $[\![\dot{e}le,\ \dot{s}ub]\!]$, if $subF[\![\dot{s}ub]\!](\dot{e}le') = \dot{e}le$. Let $samp$ be a set of samples.

3.3. The transition relation

The transition relation in CTSL is defined on programs. Programs in CTSL include macros and the executable element. Macros are used to simplify the representation of the executable element. The notion of the executable element is defined in section 4.2.

An element $\dot{e}le$ of the form *(macro* $\dot{v}ar$ $\dot{b}ody$*)* is called a macro. The elements $\dot{v}ar$ and $\dot{b}ody$ are called a variable and a body in $[\![\dot{e}le]\!]$. Let *macro* be a set of macros.

An element $\dot{e}le$ of the form *(prog* $\dot{m}acro_1 \ldots \dot{m}acro_{\dot{n}at0}$ $\dot{e}le'$*)* is called a program in CTSL. The element $\dot{e}le'$ is called an executable element in $[\![\dot{e}le]\!]$. Let *prog* be a set of programs in CTSL.

A function $traRel \in prog \times tra \to bool$ is called the transition relation in *prog:*, if $traRel(\dot{p}rog, \dot{t}ra)$ if and only if either $\dot{p}rog$ has the form *(prog* $\dot{e}le$*)*, and $traRel(\dot{t}ra(1)(\$exeEle \leftarrow \dot{e}le))$, or $\dot{p}rog$ has the form

$$(prog\ (macro\ \dot{v}ar\ \dot{b}ody)\ \dot{m}acro_1 \ldots \dot{m}acro_{\dot{n}at0}\ \dot{e}le),$$

and

$$traRel((prog\ subF[\![\{\dot{v}ar{:}\dot{b}ody\}]\!]\ (\dot{m}acro_1) \ldots subF[\![\{\dot{v}ar{:}\dot{b}ody\}]\!](\dot{m}acro_{\dot{n}at0})$$
$$subF[\![\{\dot{v}ar{:}\dot{b}ody\}]\!](\dot{e}le)),\ \dot{t}ra).$$

The conceptual *$exeEle* is defined in section 4.2

4. Basic kinds of CTSs

Basic kinds of CTSs are defined in this section.

4.1. CTSs with transition values

A CTS with transition values is characterized by the fact that its transitions can return values.

The conceptual *(0:val)* is called a transition value specifier. Let *$val* denote *(0:val)*. An element $\dot{e}le$ is called a value in $[\![\dot{t}ra]\!]$, if $traRel(\dot{t}ra)$, and $\dot{e}le = \dot{t}ra(2)(\$val)$. Thus, the individual *val* specifies a transition value. A transition $\dot{t}ra$ returns a value $\dot{e}le$, if $\dot{e}le$ is a value in $[\![\dot{t}ra]\!]$.

A conceptual $\dot{c}on$ is called an exception, if $\dot{c}on(1) = exception$. Thus, the concept *exception* specifies exceptions. Let *exc* be a set of exceptions. A transition $\dot{t}ra$ returns (or generates) an exception $\dot{e}xc$, if $\dot{e}xc$ is a value in $[\![\dot{t}ra]\!]$. The element $\dot{e}xc(0)$ specifies usually information about the generated exception, and the elements $\dot{e}xc(\dot{i}nt)$, where $\dot{i}nt < 0$, concretizes usually a kind of this information. A transition $\dot{t}ra$ is normally executed, if $\dot{t}ra$ returns no exception.

A system $\dot{cts}$ is called a CTS with transition values, if $\dot{sta}(\$val) \neq \omega$ for all *sta* such that *sta* is admissible in $[\![\dot{cts}]\!]$.

4.2. CTSs with executable elements

A CTS with executable elements is a CTS with transition values which is characterized by the fact that its transitions are associated with executable elements, and executable elements can return values.

A function $traRel \in ele \times tra \to bool$ is called the transition relation in $[\![ele:]\!]$. It specifies transitions initiated by executable elements. An element $\dot{ele}$ is executed in $[\![\dot{tra}, traRel[\![ele:]\!]]\!]$, if $traRel[\![ele:]\!](\dot{ele}, \dot{tra})$. An element $\dot{ele}$ is executed in $[\![\dot{sta}, traRel[\![ele:]\!]]\!]$, if there exists $\dot{sta}'$ such that $\dot{ele}$ is executed in $[\![(\dot{sta}, \dot{sta}'), traRel[\![ele:]\!]]\!]$. An element $\dot{ele}$ executes (initiates) a transition $\dot{tra}$, if $\dot{ele}$ is executed in $[\![\dot{tra}, traRel[\![ele:]\!]]\!]$.

The conceptual *(0:exeEle)* is called an executable element specifier. Let *$exeEle* denote *(0:exeEle)*. Thus, the individual *exeEle* specifies an executable element.

A CTS with transition values $\dot{cts}$ is called a CTS in $[\![traRel[\![ele:]\!]]\!]$, if $traRel(\dot{tra})$ if and only if $\dot{tra}(1)(\$exeEle) \neq \omega$, and

- either $\dot{tra}(1)(\$exeEle)$ is executed in $[\![\dot{tra}, traRel[\![ele:]\!]]\!]$,
- or $\dot{tra}(1)(\$exeEle)$ is not executed in $[\![\dot{tra}, traRel[\![ele:]\!]]\!]$, and
  $\dot{tra}(2) = \dot{tra}(1)(\$val \leftarrow (-1{:}unknownElement, 0{:}\dot{ele}, 1{:}exception))$.

An element $\dot{val}$ is called a value in $[\![\dot{ele}, \dot{tra}]\!]$, if $\dot{ele}$ is executed in $[\![\dot{tra}]\!]$, and $\dot{val}$ is a value in $[\![\dot{tra}]\!]$.

Executable elements in such CTSs can be partitioned into defined and predefined ones. In this case, the transition relation $traRel[\![exe:]\!]$ is defined as the union of the transition relations $traRel[\![predef:]\!]$ and $traRel[\![def:]\!]$ such that $traRel[\![predef:]\!], traRel[\![def:]\!] \in ele \times tra \to bool$. An element $\dot{ele}$ is called predefined in $[\![traRel]\!]$, if there exists $\dot{tra}$ such that $traRel[\![predef:]\!](\dot{ele}, \dot{tra})$. An element $\dot{ele}$ is called defined in $[\![traRel]\!]$, if there exists $\dot{tra}$ such that $traRel[\![def:]\!](\dot{ele}, \dot{tra})$.

In the case of partitioning executable elements into predefined and defined ones, a CTS in $[\![traRel[\![ele:]\!]]\!]$ is redefined as follows: $traRel(\dot{tra})$ if and only if $\dot{tra}(1)(\$exeEle) \neq \omega$, and

- $\dot{tra}(1)(\$exeEle)$ is executed in $[\![\dot{tra}, traRel[\![predef:]\!]]\!]$, or
- $\dot{tra}(1)(\$exeEle)$ is not executed in $[\![\dot{tra}, traRel[\![predef:]\!]]\!]$, and $\dot{tra}(1)(\$exeEle)$ is executed in $[\![\dot{tra}, traRel[\![predef:]\!]]\!]$, or
- $\dot{tra}(1)(\$exeEle)$ is not executed in $[\![\dot{tra}, traRel[\![predef:]\!]]\!]$ and $[\![\dot{tra}, traRel[\![def:]\!]]\!]$, and $\dot{tra}(2) = \dot{tra}(1)(\$val \leftarrow (-1{:}unknownElement, 0{:}\dot{ele}, 1{:}exception))$.

CTSs with executable elements can be used, for example, to specify abstract machines of programming languages. In this case, executable elements are executable constructs of programming languages.

4.3. CTSs with execution contexts

CTSs with execution contexts generalize CTSs with executable elements and transition values. They are characterized by the fact that elements are executed in execution contexts, and their values are stored in these contexts.

An element *ėle* is called an execution context in $[\![\dot{sta}]\!]$, if

$$\dot{sta}((0{:}\dot{e}le,\ 1{:}executionContext)) \neq \omega.$$

Thus, the concept *executionContext* describes execution contexts. Let *exeCont* be a set of execution contexts in $[\![\dot{sta}]\!]$

Specifiers of transition values and executable elements are redefined in CTSs with execution contexts.

A conceptual *(-1:val, 0:ėxeCont, 1:executionContext)* is called a transition value specifier in $[\![\dot{e}xeCont]\!]$. Let *$val* denote *(-1:val, 0:ėxeCont, 1:executionContext)*. An element *ėle* is called a value in $[\![\dot{tra},\ \dot{e}xeCont]\!]$, if *traRel(ṫra)*, and $\dot{e}le = \dot{tra}(2)(\$val)$. Thus, the attribute *val* specifies a transition value in execution contexts. A transition *ṫra* returns a value *ėle* in $[\![\dot{e}xeCont]\!]$, if *ėle* is a value in $[\![\dot{tra},\ \dot{e}xeCont]\!]$. An element *v̇al* is called a value in $[\![\dot{e}le,\ \dot{tra},\ \dot{e}xeCont]\!]$, if *ėle* is executed in $[\![\dot{tra},\ \dot{e}xeCont]\!]$, and $\dot{v}al = \dot{tra}(2)(\$val)$.

A transition *ṫra* returns (or generates) an exception *ėxc* in $[\![exeCont]\!]$, if *ėxc* is a value in $[\![\dot{tra},\ \dot{e}xeCont]\!]$. A transition *ṫra* is normally executed in $[\![exeCont]\!]$, if *ṫra* returns no exception in $[\![exeCont]\!]$.

A conceptual *(-1:exeEle, 0:ėxeCont, 1:executionContext)* is called an executable element specifier in $[\![exeCont]\!]$. Let *$exeEle* denote *(-1:exeEle, 0:ėxeCont, 1:executionContext)*. Thus, the attribute *exeEle* specifies an executable element in execution contexts. The same element can be executed in different execution contexts.

A function $traRel \in ele \times ele \times tra \to bool$ is called the transition relation in $[\![exeCont{:}]\!]$. It specifies transitions initiated by elements which are executed in execution contexts. An element *ėle* is executed in $[\![\dot{tra},\ \dot{e}xeCont]\!]$, if $traRel[\![exeCont{:}]\!](\dot{e}le,\ \dot{e}xeCont,\ \dot{tra})$. An element *ėle* is executed in $[\![\dot{sta},\ \dot{e}xeCont]\!]$, if there exists *ṡta* such that *ėle* is executed in $[\![(\dot{sta},\ \dot{sta}'),\ \dot{e}xeCont]\!]$. An element *ėle* executes (initiates) a transition *ṫra* in $[\![\dot{e}xeCont]\!]$, if *ėle* is executed in $[\![\dot{tra},\ \dot{e}xeCont]\!]$.

A CTS $\dot{cts}$ is called a CTS in $[\![traRel[\![exeCont:]\!]]\!]$, if $\dot{sta}(\$val) \neq \omega$ for all $\dot{sta}$ and $\dot{exeCont}[\![\dot{sta}]\!]$ such that $\dot{sta}$ is admissible in $[\![\dot{cts}]\!]$, and $traRel(\dot{tra})$ if and only if there exists $\dot{exeCont}[\![\dot{sta}]\!]$ such that $\dot{tra}(1)(\$exeEle) \neq \omega$, and either $\dot{tra}(1)(\$exeEle)$ is executed in $[\![\dot{tra}, \dot{exeCont}]\!]$, or $\dot{tra}(1)(\$exeEle)$ is not executed in $[\![\dot{tra}, \dot{exeCont}]\!]$, and

$$\dot{tra}(2) = \dot{tra}(1)(\$val \leftarrow (-1{:}unknownElement, 0{:}\dot{ele}, 1{:}exception)).$$

4.4. CTSs with counters

CTSs with counters are CTS with executable elements and transition values which are characterized by the fact that they can define named counters and generate new elements based on them.

A conceptual $(0{:}\dot{ele}, 1{:}counter)$ is called a counter specifier with name $\dot{ele}$. Let cou and nam be sets of counter specifiers and their names, respectively. An element $\dot{cou}$ is called a counter in $\dot{sta}$, if $\dot{sta}(\dot{cou}) \neq \omega$. Thus, the concept $counter$ defines counters. The element $\dot{sta}(\dot{cou})$ is called a value in $[\![cou{:}\dot{cou}, \dot{sta}]\!]$.

An element $\dot{con}$ is called generated in $[\![\dot{cou}]\!]$, if $\dot{con}$ has the form $(0{:}\dot{nat}, 1{:}\dot{nam})$, and $\dot{nam}$ is a name of $\dot{cou}$. Thus, the name of a counter is a concept for elements generated by this counter.

An element $\dot{ele}$ of the form $(newCount\ \dot{nam})$ is called an element generator in $[\![conc{:}\dot{nam}]\!]$ and defined as follows: $traRel(\dot{ele}, \dot{tra})$ if and only if either $\dot{tra}(1)(\dot{cou}) \neq \omega$, and $\dot{tra}(2) = \dot{tra}(1)(\dot{cou} \leftarrow \dot{tra}(1)(\dot{cou}) + 1, \$val \leftarrow (0{:}\dot{tra}(1)(\dot{cou}) + 1, 1{:}\dot{nam}))$, or $\dot{tra}(1)(\dot{cou}) = \omega$, and $\dot{tra}(2) = \dot{tra}(1)(\dot{cou} \leftarrow 1, \$val \leftarrow (0{:}1, 1{:}\dot{nam}))$.

An element $\dot{nam}$ is called a name in $[\![\dot{ele}]\!]$. An element generator generates a new element by the counter with the name which coincides with the name of the generator, and adds this counter, if it was not.

A CTS with executable elements and transition values $\dot{cts}$ is called a CTS with counters, if the element $(newCount\ \dot{nam})$ is predefined in $[\![\dot{cts}]\!]$.

4.5. CTSs with history varibles

CTSs with history varibles are CTSs with counters which are characterized by the fact that they can define variables storing the history of values of $\$val$.

A conceptual $(0{:}\dot{nat}, 1{:}hvar)$ is called a history variable specifier. Let $hvar$ be a set of history variable specifiers. An element $\dot{hvar}$ is called a history variable in $\dot{sta}$, if $\dot{sta}(\dot{hvar}) \neq \omega$. Thus, the concept $hvar$ defines history variables. An element $\dot{sta}(\dot{hvar})$ is called a value in $[\![hvar{:}\dot{hvar}, \dot{sta}]\!]$.

An element $\dot{e}le$ of the form *(hvar* ($\dot{e}le_1$, ..., $\dot{e}le_{\dot{n}at}$) $\dot{e}le'$*)* is called a history variable generator in $[\![$*(*$\dot{e}le_1$, ..., $\dot{e}le_{\dot{n}at}$*),* $\dot{e}le'$$]\!]$ and defined as follows: *traRel(*$\dot{e}le$, $\dot{t}ra$*)* if and only if there exist $\dot{s}ta_0$, $\dot{s}ta_1$, ..., $\dot{s}ta_{\dot{n}at}$ such that $\dot{s}ta_0 = \dot{t}ra(1)$, *traRel((newCount hvar),* ($\dot{s}ta_{\dot{n}at'-1}$, $\dot{s}ta_{\dot{n}at'}$*))* for all $1 \le \dot{n}at' \le \dot{n}at$, and *traRel(*$\dot{s}ta_{\dot{n}at}$*(*$exeEle \leftarrow \dot{e}le''$, $val \leftarrow \dot{t}ra(1)(val)$, $\dot{t}ra(2)$*)*, where $\dot{e}le''$ is the result of replacement $\dot{e}le_1$, ..., $\dot{e}le_{\dot{n}at}$ in $\dot{e}le'$ by $\dot{s}ta_1$*(*$val*)*, ..., $\dot{s}ta_{\dot{n}at}$*(*$val*)*, respectively.

The elements $\dot{e}le_1$, ..., $\dot{e}le_{\dot{n}at}$ are called variables in $[\![\dot{e}le]\!]$, and $\dot{e}le'$ are called a body in $[\![\dot{e}le]\!]$. A history variable generator generates new history variables corresponding to the variables of the generator and replace all occurences of the generator variables in its body by these history variables.

A CTS with counters $\dot{c}ts$ is called a CTS with history variables, if the element *(hvar* ($\dot{e}le_1$, ..., $\dot{e}le_{\dot{n}at}$*)* $\dot{e}le'$*)* is predefined in $[\![\dot{c}ts]\!]$.

4.6. CTSs with defined conceptuals

CTSs with defined conceptuals specify definitions of conceptuals.

A conceptual $\dot{c}on'$ is called a definition in $[\![\dot{c}on,\ \dot{s}ta]\!]$, if $\dot{n}at$ is an order in $[\![\dot{c}on]\!]$, $\dot{c}on' = \dot{c}on \cup \{(\dot{n}at + 1)$*:conDef*$\}$, and $\dot{s}ta(\dot{c}on') \ne \omega$. An element $\dot{s}ta(\dot{c}on')$ is called a body in $[\![$*def:*$\dot{c}on'$*,* $\dot{s}ta]\!]$. Thus, the definition of a conceptual of the order $\dot{n}at$ is characterized by the attribute *conDef* of the order $\dot{n}at + 1$.

A CTS $\dot{c}ts$ is called a CTS with defined conceptuals, if semantics of conceptuals *sem* is redefined by the following way:

- if $\dot{s}ta(\dot{c}on) \ne \omega$, then *sem(*$\dot{c}on$, $\dot{s}ta$*)* = $\dot{s}ta(\dot{c}on)$;
- if $\dot{s}ta(\dot{c}on) = \omega$, $\dot{c}on'$ is a definition in $[\![\dot{c}on,\ \dot{s}ta]\!]$, and $\dot{e}le$ is a body in $[\![$*def:*$\dot{c}on'$*,* $\dot{s}ta]\!]$, then *sem(*$\dot{c}on$, $\dot{s}ta$*)* = *sem(*$\dot{e}le$, $\dot{s}ta$*)*;
- otherwise, *sem(*$\dot{c}on$, $\dot{s}ta$*)* = ω.

4.7. CTS with transition rules

An element $\dot{e}le$ of the form *(rule* $\dot{e}le_1$ *var* $\dot{v}arSet$ *then* $\dot{e}le_2$*)* is called a transition rule, if the elements of $\dot{v}arSet$ are pairwise distinct. The elements $\dot{e}le_1$, $\dot{v}arSet$ and $\dot{e}le_2$ are called a sample, variable specifier and body in $[\![\dot{e}le]\!]$, respectively. The elements of $\dot{v}arSet$ are called variables in $[\![\dot{e}le]\!]$. Let *rul*, *samp*, and *body* be sets of transition rules and their samples and bodies, respectively.

A function $traRel \in ele \times rul \times tra \to bool$ is called the transition relation in $[\![$*rul:,* $\dot{c}ts]\!]$,

if *traRel⟦rule:⟧(ėle, ṙul, ṫra)* if and only if there exists *ṡub* such that *sup(ṡub) = v̇arSet*, *ėle* is an instance in *⟦ṡamp, ṡub⟧*, *traRel⟦ċts⟧(ṫra(1)($exeEle ← subF⟦ṡub⟧(ḃody)), ṫra(2))*, and *ṫra(2)($val) ≠ (-1:ruleNotExecutable, 0:ṙul, 1:exception)*.

A transition rule *ṙul* is a transition rule in *⟦ṡta⟧*, if *ṡta(0:ṙul, 1:rule) ≠ ω*. Thus, the concept *rule* defines a set of transition rules in *⟦ṡta⟧*.

A CTS with executable elements *ċts* is a CTS in *⟦traRel⟦rul:, ċts⟧⟧*, if *traRel⟦def:⟧(ėle, ṫra)* if and only if there exists *ṙul* such that *ṙul* is a transition rule in *ṫra(1)*, and *traRel⟦rule:, ċts⟧(ėle, ṙul, ṫra)*.

There are shortcuts for the frequently used kinds of transition rules.

An element *ėle* of the form *(rule ṡamp var v̇arSet where ėle′ then ḃody)* is called a conditional transition rule, and *ėle′* is called a condition in *⟦ėle⟧*. It is a shortcut for the rule *(rule ṡamp var v̇arSet then (if ėle′ then ḃody else ($val ::= (-1:ruleNotExecutable, 0:ėle, 1:exception))))*. The elements *(... ::= ...)* and *(if ... then ... else ...)* are defined in sections 5.3 and 5.5, respectively.

An element *ėle* of the form *(rule ṡamp var v̇arSet hvar ȯrdStr then ḃody)* is called a transition rule with history variables, the element *ȯrdStr* is called a history variable specifier in *⟦ėle⟧*, and the elements of *ȯrdStr* are called history variables in *⟦ėle⟧*. It is a shortcut for the rule *(rule ṡamp var v̇arSet then (hvar ȯrdStr ḃody))*. The element *(hvar ...)* is defined in section 4.5.

An element *ėle* of the form *(rule ṡamp var v̇arSet catch ėle′ then ḃody)* is called a transition rule with the return handler, and the element *ėle′* is called a return variable specifier. It is a shortcut for the rule *(rule ṡamp var v̇arSet then (seq (catch ėle′) ḃody))*. The elements *(seq ...)* and *(catch ...)* are defined in sections 5.6 and 5.9, respectively.

Combinations of these kinds of rules can include not more than one occurence of the part *catch ėle* defined last, and any number of the parts *where ėle* and *hvar ȯrdStr* defined from right to left. For example, the rule *(rule ṡamp var v̇arSet hvar ȯrdStr′ where ėle hvar ȯrdStr″ catch ėle′ then ḃody)* is defined by the following sequence of transformations:

1. *(rule ṡamp var v̇arSet hvar ȯrdStr′ where ėle hvar ȯrdStr″ catch ėle′ then ḃody)* →
2. *ėle″ ≡ (rule ṡamp var v̇arSet hvar ȯrdStr′ where ėle catch ėle′ then (hvar ȯrdStr″ ḃody))* →
3. *(rule ṡamp var v̇arSet hvar ȯrdStr′ catch ėle′ then (if ėle then (hvar ȯrdStr″ ḃody)) else ($val ::= (-1:ruleNotExecutable, 0:ėle″, 1:exception)))* →
4. *(rule ṡamp var v̇arSet catch ėle′ then (hvar ȯrdStr′ (if ėle then (hvar ȯrdStr″ ḃody)) else*

($val ::= (-1:ruleNotExecutable, 0:ėle″, 1:exception)))) →

5. *(rule ṡamp var v̇arSet then (seq (catch ėle′) (hvar ȯrdStr′ (if ėle then (hvar ȯrdStr″ ḃody)) else ($val ::= (-1:ruleNotExecutable, 0:ėle″, 1:exception))))).*

The element *ṙul* adds the rule *ṙul* to *ṡta*: *traRel(ėle, ṫra)* if and only if *ṫra(2) = ṫra(1)((0:ṙul, 1:rule) ← true)*.

The element *ėle* of the form *(delete ṙul)* removes the rule *ṡta* from *ċts*: *traRel(ėle, ṫra)* if and only if *ṫra(2) = ṫra(1)((0:ṙul, 1:rule) ← ω)*.

The element *ėle* of the form *(deleteRules)* removes all rules from *ṡta*: *traRel(ėle, ṫra)* if and only if *ṫra(2) = ṫra(1)((0:ṙul$_1$, 1:rule) ← ω, ..., (0:ṙul$_{\dot{n}at}$, 1:rule) ← ω)*, where *{ṙul$_1$, ..., ṙul$_{\dot{n}at}$}* is a set of all transition rules in *ṫra(1)*.

4.8. CTSs with types

CTSs with types specify types of elements and literals of these types.

Let *type ⊂ ele*. An element *ṫype* is called a type. A function *lit ∈ type → 2^{ele}* is called a literal function in ⟦*type*⟧. An element *ėle* is called a literal in ⟦*ṫype*⟧, if *ėle ∈ lit(ṫype)*.

An element *(ėle is ṫype)* is called a characteristic element in ⟦*ṫype*⟧ and defined as follows: *traRel(ėle, ṫra)* if and only if either *ėle ∈ lit(ṫype)*, and *ṫra(2)($val) = true*, or *ėle ∉ lit(ṫype)*, and *ṫra(2)($val) = false*.

A CTS *ċts* is called a CTS with types in ⟦*type, lit*⟧, if *type* is a set of types, *lit* is a literal function in ⟦*type*⟧, and *(ėle is ṫype)* is a predefined element in *ċts* for each *ṫype*.

The set *type* of the CTSL language includes the following basic types:

- *element* such that *lit(element) = ele*;
- *atom* such that *lit(atom) = ato*;
- *emptyStr* such that *lit(emptyElement) = {(), {}}*;
- *emptyOrdStr* such that *lit(emptyOrdStr) = {()}*;
- *emptyUnoStr* such that *lit(emptyUnoStr) = {{}}*;
- *eleStr* such that *lit(eleStr) = eleStr*;
- *ordStr* such that *lit(ordStr) = ordStr*;
- *unoStr* such that *lit(unoStr) = unoStr*;
- *labStr* such that *lit(labStr) = labStr*;
- *int* such that *lit(int) = int*;
- *nat* such that *lit(nat) = nat*;

- *nat0* such that *lit(nat0) = nat0*;
- *bool* such that *lit(bool) = bool*;
- *rule* such that *lit(rule) = rule*;
- *macro* such that *lit(macro) = macro*;
- *program* such that *lit(program) = prog.*

5. Basic predefined elements in CTSL

Basic predefined elements in CTSL are defined in this section.

5.1. The element *omega*

The element *(omega)* is called an indeterminate return and defined as follows: $traRel(\dot{e}le, \dot{t}ra)$ if and only if $\dot{t}ra(2) = \dot{t}ra(1)(\$val \leftarrow \omega)$. It returns an indeterminate value.

5.2. The elements *ordStrToUnoStr* and *unoStrToOrdStr*

The element *ele* of the form *(ordStrToUnoStr ȯrdStr)* is called a converter in $[\![ordStr:, unoStr:]\!]$ and defined as follows: $traRel(\dot{e}le, \dot{t}ra)$ if and only if either $\dot{e}le = (\dot{e}le_1, \ldots, \dot{e}le_{\dot{n}at})$, and $\dot{t}ra(2) = \dot{t}ra(1)(\$val \leftarrow \{\dot{e}le_1, \ldots, \dot{e}le_{\dot{n}at}\})$, or $\dot{e}le \notin ordStr$, and $\dot{t}ra(2) = \dot{t}ra(1)(\$val \leftarrow (0{:}\dot{e}le, 1{:}exception))$. It converts elements of *ordStr* into elements of *unoStr*.

The element *ele* of the form *(unoStrToOrdStr u̇noStr)* is called a converter in $[\![unoStr:, ordStr:]\!]$ and defined as follows: $traRel(\dot{e}le, \dot{t}ra)$ if and only if either $\dot{e}le = \{\dot{e}le_1, \ldots, \dot{e}le_{\dot{n}at}\}$, and $\dot{t}ra(2) = \dot{t}ra(1)(\$val \leftarrow (\dot{e}le'_1, \ldots, \dot{e}le'_{\dot{n}at}))$, where $(\dot{e}le'_1, \ldots, \dot{e}le'_{\dot{n}at})$ is a permutation of $(\dot{e}le_1, \ldots, \dot{e}le_{\dot{n}at})$, or $\dot{e}le \notin unoStr$, and $\dot{t}ra(2) = \dot{t}ra(1)(\$val \leftarrow (0{:}\dot{e}le, 1{:}exception))$. It converts elements of *unoStr* into elements of *ordStr*.

5.3. The assignment

The element $\dot{e}le$ of the form $(\dot{c}on ::= \dot{e}le')$ is called an assignment and defined as follows: $traRel(\dot{e}le, \dot{t}ra)$ if and only if there exists $\dot{s}ta$ such that $traRel(\dot{t}ra(1)(\$exeEle \leftarrow \dot{e}le'), \dot{s}ta)$, and $\dot{t}ra(2) = \dot{s}ta(\dot{c}on \leftarrow \dot{s}ta(\$val))$. It assign the value of $\dot{e}le'$ to $\dot{c}on$. The elements $\dot{c}on$ and $\dot{e}le'$ are called the left-hand and right-hand sides in $[\![\dot{e}le]\!]$, respectively.

5.4. The element *skip*

The element $\dot{ele}$ of the form *(skip)* is defined as follows: *traRel($\dot{ele}$, $\dot{tra}$)* if and only if $\dot{tra}(1) = \dot{tra}(2)$. It executes an action which does not change a state. In particular, it does not change the value of *$val*.

5.5. The conditional element

The element $\dot{ele}$ of the form *(if $\dot{cond}$ then $\dot{ele}_1$ else $\dot{ele}_2$)* is called a conditional element and defined as follows: *traRel($\dot{ele}$, $\dot{tra}$)* if and only if there exists $\dot{sta}$ such that *traRel($\dot{tra}$(1)(\$exeEle ← $\dot{cond}$), $\dot{sta}$)*, and

- $\dot{sta}$(\$val) = *true*, and *traRel($\dot{sta}$(\$exeEle ← $\dot{ele}_1$), $\dot{tra}$(2))*, or
- $\dot{sta}$(\$val) = *false*, and *traRel($\dot{sta}$(\$exeEle ← $\dot{ele}_2$), $\dot{tra}$(2))*, or
- $\dot{sta}$(\$val) $\notin$ {*true, false*}, and $\dot{tra}(2) = \dot{sta}$(\$val ← (0:$\dot{ele}$,1:exception)).

The elements $\dot{cond}$, $\dot{ele}_1$ and $\dot{ele}_2$ are called a condition, *then*-branch and *else*-branch in $[\![\dot{ele}]\!]$. The element $\dot{ele}$ executes *then*-branch or *else*-branch depending on the value of the condition.

The element *(if $\dot{cond}$ then $\dot{ele}$)* is a shortcut for the element *(if $\dot{cond}$ then $\dot{ele}$ else (skip))*.

5.6. The sequential composition

The element $\dot{ele}$ of the form *(seq $\dot{ele}_1$ ... $\dot{ele}_{\dot{nat}0}$)* is called a sequential composition. The elements $\dot{ele}_1$, ..., $\dot{ele}_{\dot{nat}}$ are called elements in $[\![\dot{ele}]\!]$, and their sequence is called a body in $[\![\dot{ele}]\!]$. The element $\dot{ele}$ executes its elements sequentially from left to right.

Semantics of the element *(seq)* coincides with semantics of the element *(skip)*.

The element $\dot{ele}$ of the form *(seq $\dot{ele}_1$, ..., $\dot{ele}_{\dot{nat}}$)* is defined as follows: *traRel($\dot{ele}$, $\dot{tra}$)* if and only if there exists $\dot{sta}$ such that *traRel($\dot{tra}$(1)(\$exeEle ← $\dot{ele}_1$), $\dot{sta}$)*, and *traRel($\dot{sta}$(\$exeEle ← (seq $\dot{ele}_2$, ..., $\dot{ele}_{\dot{nat}}$)), $\dot{tra}$(2))*.

5.7. Evaluators

The element *ele* of the form *(* $\dot{body}$ *)* is called an evaluator and defined as follows: *traRel($\dot{ele}$, $\dot{tra}$)* if and only if there exists $\dot{sta}$ such that *traRel($\dot{tra}$(1)(\$exeEle ← $\dot{body}$), $\dot{sta}$)*, and either $\dot{sta}$(\$val) $\notin$ *exc*, and *traRel($\dot{sta}$(\$exeEle ← $\dot{sta}$(\$val)), $\dot{tra}$(2))*, or $\dot{sta}$(\$val) $\in$ *exc*, and $\dot{tra}(2) = \dot{sta}$.

The element $\dot{body}$ is called a body in $[\![\dot{ele}]\!]$. The element $\dot{ele}$ first executes $\dot{body}$, and then executes the value of $\dot{body}$.

5.8. Quoters

The element $\dot{e}le$ of the form *(quote* $\dot{b}ody$*)* is called a quoter and defined as follows: *traRel(*$\dot{e}le$*,* $\dot{t}ra$*)* if and only if $\dot{t}ra(2) = \dot{t}ra(1)(\$val \leftarrow \dot{b}ody)$. The element $\dot{b}ody$ is called a body in $[\![\dot{e}le]\!]$. The element $\dot{e}le$ changes a state only in *$val*, and assign $\dot{b}ody$ to *$val*.

The object $'\dot{b}ody$ is a shortcut for the element *(quote* $\dot{b}ody$*)*. For example, *'true* is a shortcut for *(quote true)*.

5.9. Return handlers

The element $\dot{e}le$ of the form *(catch* $\dot{v}ar$ $\dot{b}ody$*)* is called a return handler and defined as follows: *traRel(*$\dot{e}le$*,* $\dot{t}ra$*)* if and only if there exists $\dot{s}ta$ such that $traRel[\![ele:]\!]((newCount\ retVar), (\dot{t}ra(1)(\$val \leftarrow true), \dot{s}ta))$, and

$$traRel(\dot{s}ta(exeEle \leftarrow subF[\![\{\dot{v}ar \leftarrow \dot{s}ta(\$val)\}]\!](\dot{b}ody)), \dot{t}ra(2)).$$

The elements $\dot{v}ar$ and $\dot{b}ody$ are called a return variable and body in $[\![\dot{e}le]\!]$, respectively. An element of the form *(*$\dot{n}at$*, retVar)* is called a return variable. The element $\dot{e}le$ stores the current value of *$val* into a new return variable $\dot{v}ar'$, resets the value of *$val* to *true* and executes the body of $\dot{e}le$ in which all occurences of the return variable in $[\![\dot{e}le]\!]$ are replaced by $\dot{v}ar'$. It models exception handling in CTSL.

5.10. Selectors

The element $\dot{e}le$ of the form *(select* $\dot{e}le'$ *from* $\dot{s}amp$ *var* $\dot{v}arSet$ *for* $\dot{c}ond$*)* is called a selector and defined as follows: *traRel(*$\dot{e}le$*,* $\dot{t}ra$*)* if and only if $\dot{t}ra(2) = \dot{t}ra(1)(\$val \leftarrow \dot{s}et)$, where $\dot{s}et \in unoStr$ is a set of $subF[\![\dot{s}ub]\!](\dot{e}le')$ such that $subF[\![\dot{s}ub]\!](\dot{s}amp)$ is a conceptual in $[\![\dot{t}ra(1)]\!]$, $sup(\dot{s}ub) = \dot{v}arSet$, and there exists $\dot{s}ta$ such that $traRel(\dot{t}ra(1)(\$exeEle \leftarrow subF[\![\dot{s}ub]\!](\dot{c}ond)), \dot{s}ta)$, and $\dot{s}ta(\$val) \neq (-1{:}notSelected, 1{:}exception)$.

The elements $\dot{e}le'$, $\dot{s}amp$, $\dot{v}arSet$ and $\dot{c}ond$ are called a selection specifier, sample, set of variables and condition in $[\![\dot{e}le]\!]$.

Thus, the element $\dot{e}le$ returns the set of instances of the selection specifier $\dot{e}le'$ for all substitutions $\dot{s}ub$ defined on variables from $\dot{v}arSet$ such that there exists a conceptual in $\dot{s}ta$ which is an instance of the sample $\dot{s}amp$ in $[\![\dot{s}ub]\!]$, and the instance of the condition $\dot{c}ond$ in $[\![\dot{s}ub]\!]$ does not return the exception *(-1:notSelected, 1:exception)*.

The element $\dot{e}le$ of the form *(select* $\dot{e}le'$ *from* $\dot{s}amp$ *var* $\dot{v}arSet$ *where* $\dot{c}ond$*)* is a shortcut for the element *(select* $\dot{e}le'$ *from* $\dot{s}amp$ *var* $\dot{v}arSet$ *for (if* $\dot{c}ond$ *then (skip) else ($val ::= (-1{:}notSelected, 1{:}exception))))*. The element $\dot{c}ond$ is called a condition in $[\![\dot{e}le]\!]$.

5.11. The conditional pattern matching

The element *ėle* of the form *(if ėle′ matches ṡamp var v̇arSet for ċond then ėle*$_1$ *else ėle*$_2$*)* is called a conditional pattern matching and defined as follows: *traRel(ėle, ṫra)* if and only if

1. either there exist *ṡub* and *ṡta* such that *sup(ṡub)* = *v̇arSet*, *ėle′* is an instance in ⟦*ṡamp, ṡub*⟧, *traRel(ṫra($exeEle* ← *subF*⟦*ṡub*⟧*(ċond)), ṡta), ṡta($val)* ≠ *(-1:notMatch, 1:exception)*, and *traRel(ṡta($exeEle* ← *subF*⟦*ṡub*⟧*(ėle*$_1$*)), ṫra(2))*.
2. or the condition 1 does not assert, and *traRel(ṫra(1)($exeEle* ← *ėle*$_2$*), ṫra(2))*.

The elements *ėle′*, *ṡamp*, *v̇arSet*, *ċond*, *ėle*$_1$, and *ėle*$_2$ are called a matched element, pattern, set of variables, condition, *then*-branch and *else*-branch in ⟦*ėle*⟧.

Thus, the element *ėle* executes the instance of the *then*-branch *ėle*$_1$ in ⟦*ṡub*⟧, if *ėle′* is the instance of the sample *ṡamp* in ⟦*ṡub*⟧, and the instance of *ċond* in ⟦*ṡub*⟧ does not return the exception *(-1:notMatch, 1:exception)*. Otherwise, the element *ėle* executes the *else*-branch *ėle*$_2$.

The element *ėle* of the form *(if ėle′ matches ṡamp var v̇arSet for ċond then ėle*$_1$*)* is a shortcut for the element *(if ėle′ matches ṡamp var v̇arSet for ċond then ėle*$_1$ *else (skip))*.

The element *ėle* of the form *(if ėle′ matches ṡamp var v̇arSet where ċond then ėle*$_1$ *else ėle*$_2$*)* is a shortcut for the element *(if ėle′ matches ṡamp var v̇arSet for (if ċond then (skip) else ($val ::= (-1:notSelected, 1:exception))) then ėle*$_1$ *else ėle*$_2$*)*. The element *ċond* is called a condition in ⟦*ėle*⟧.

5.12. Iterators

The element *ėle* of the form *(foreach v̇ar in ėle′ do ḃody)* is called an iterator and defined as follows: *traRel(ėle, ṫra)* if and only if there exists *ṡta* such that *traRel(ṫra(1)($exeEle* ← *ėle′), sta)*, and

- *ṡta($val)* is an empty structure, and *ṫra(2)* = *ṡta*, or
- *ṡta($val)* ∈ *exc*, and *ṫra(2)* = *ṡta*, or
- *ṡta($val)* ∈ *ato* ∪ *labStr* \ *exc*, and *ṫra(2)* = *ṡta($val* ← *(0:ėle, 1:exception))*, or
- *ṡta($val)* ∈ *ordStr* ∪ *unoStr*, and
  traRel(ṡta($exeEle ← *(hvar v̇ar (foreach1 v̇ar in ṡta($val) do ḃody))), ṫra(2))*.

The elements *v̇ar*, *ėle′* and *ḃody* are called an iteration variable, iteration structure specifier and body in ⟦*ėle*⟧, respectively.

The element *(foreach1 v̇ar in ėle′ do ḃody)* is defined by the rules:

```
(if (foreach1 x in () do y) var (x, y) then (skip))
```

```
(if (foreach1 x in {} do y) var (x, y,) then (skip))

(if (foreach1 x in (v .:: w) do z) var (x, y, v, w)
 then (seq (x ::= 'v) z (foreach1 x in w do z)))

(if (foreach1 x in (v .::u w) do z) var (x, y, v, w)
 then (seq (x ::= 'v) z (foreach1 x in w do z)))
```

Thus, the element $\dot{e}le$ executes sequentially $\dot{b}ody$ in $[\![var{:}\dot{v}ar,\ val{:}\dot{e}le'']\!]$ for elements of the structure $\dot{s}tr$, where $\dot{s}tr$ is the value of $\dot{e}le'$. Executing $\dot{b}ody$ in $[\![var{:}\dot{v}ar,\ val{:}\dot{e}le'']\!]$ is executing $\dot{b}ody$ when the value of the variable $\dot{v}ar$ is equal to $\dot{e}le''$.

5.13. The element *throw*

The element $\dot{e}le$ of the form *(throw* $\dot{b}ody$*)* is defined by the rule:

(if (throw x) var (x) where (x is exception) then ($val ::= 'x))

The element $\dot{b}ody$ is called a body in $[\![\dot{e}le]\!]$.

5.14. Branching

The branching elements specify the order of execution of elements called branches and what branches are executed.

The element $\dot{e}le$ of the form *(orBranching* $\dot{e}le_1$ *...* $\dot{e}le_{\dot{n}at0}$*)* is called an or-branching and defined as follows: *traRel(*$\dot{e}le$*,* $\dot{t}ra$*)* if and only if either $\dot{n}at0 = 0$, and $\dot{t}ra(1) = \dot{t}ra(2)$, or $\dot{n}at0 > 0$, and there exists $\dot{s}ta$ such that *traRel(*$\dot{t}ra(1)$*($exeEle* $\leftarrow$ $\dot{e}le_1$*),* $\dot{s}ta$*)*, and

- $\dot{s}ta$*($val) = (-1:failBranch, 1:execution)*, and *traRel(*$\dot{s}ta$*($exeEle* $\leftarrow$ *(orBranching or* $\dot{e}le_2$ *...* $\dot{e}le_{\dot{n}at}$*)),* $\dot{t}ra(2)$*)*, or
- $\dot{s}ta$*($val)* $\neq$ *(-1:failBranch, 1:execution)*, and $\dot{t}ra(2) = \dot{s}ta$.

The elements $\dot{e}le_1$, ..., $\dot{e}le_{\dot{n}at}$ are called branches in $[\![\dot{e}le]\!]$.

Thus, the element $\dot{e}le$ executes branches sequentially until the next branch is normally executed, i. e. is executed without returning the exception *(-1:failBranch, 1:execution)*.

The element $\dot{e}le$ of the form *(andBranching* $\dot{e}le_1$ *...* $\dot{e}le_{\dot{n}at0}$*)* is called an and-branching and defined as follows: *traRel(*$\dot{e}le$*,* $\dot{t}ra$*)* if and only if either $\dot{n}at0 = 0$, and $\dot{t}ra(1) = \dot{t}ra(2)$, or $\dot{n}at0 > 0$, and there exists $\dot{s}ta$ such that *traRel(*$\dot{t}ra(1)$*($exeEle* $\leftarrow$ $\dot{e}le_1$*),* $\dot{s}ta$*)*, and

- *ṡta($val)* $\notin$ *exc* $\setminus$ *{(-1:stopBranch, 1:execution)}*, and *traRel(ṡta($exeEle* $\leftarrow$ *(orBranching or* $\dot{e}le_2$ ... $\dot{e}le_{\dot{n}at}$*)), ṫra(2))*, or
- *ṡta($val)* $\in$ *exc* $\setminus$ *{(-1:stopBranch, 1:execution)}*, and $\dot{t}ra(2) = \dot{s}ta$.

The elements $\dot{e}le_1$, ..., $\dot{e}le_{\dot{n}at}$ are called branches in $[\![\dot{e}le]\!]$.

Thus, the element $\dot{e}le$ executes branches sequentially until the next branch return an exception which is distinct from the exception *(-1:stopBranch, 1:execution)*.

6. Basic operations in CTSL

Basic operations in CTSL are defined in this section. Let *ope* be a set of operations.

6.1. Boolean operations

The set *ato* includes atoms *true* and *false* which specify the corresponding boolean values.

The element $\dot{e}le$ of the form $(\dot{e}le_1$ *and* $\dot{e}le_2)$ specifies the boolean operation of conjunction. Semantics of $\dot{e}le$ coincides with semantics of the element

(if ele_1 *then (if (*ele_2*) then 'true else 'false) else 'false).*

The elements $(\dot{e}le_1$ $\dot{o}pe$ $\dot{e}le_2)$, where $\dot{o}pe \in$ *{or, =>, <=>}* specifying the boolean operations of disjunction, implication and equivalence are defined in the similar way.

The element $\dot{e}le$ of the form $(not\ \dot{e}le')$ specifies the boolean operation of negation. Semantics of $\dot{e}le$ coincides with semantics of the element $(if\ \dot{e}le'\ then$ *'false else 'true).*

6.2. Equality and inequality of elements

The element $\dot{e}le$ of the form $(\dot{e}le_1 = \dot{e}le_2)$ specifies the operation of equality $=$ on elements. Semantics of $\dot{e}le$ coincides with semantics of the pseudoelement

```
(hvar ($x, $y)
 (seq ($x ::= ėle1) ($y ::= ėle2)
      (if ⟦$sta($x) = $sta($y)⟧ then ($val ::= 'true) else ($val ::= 'false)))
```

Pseudoelements are extension of elements by constructs $[\![\dot{o}bj]\!]$, where $\dot{o}bj$ is either a property or an expression. The object *$sta* denotes the current state.

The element $\dot{e}le$ of the form $(\dot{e}le_1$ *!=* $\dot{e}le_2)$ specifies the operation of inequality *!=* on elements. It is defined in the similar way.

6.3. Integer operations and relations

The element *ėle* of the form *(ėle*$_1$ *+ ėle*$_2$*)* specifies the integer addition *+*. Semantics of *ėle* coincides with semantics of the pseudoelement

```
(hvar ($x, $y)
 (seq
  ($x ::= ėle1) (if ⌈$sta($x) ∉ int⌉ then ($val ::= '(0:ėle, 1:exception)))
  ($y ::= ėle1) (if ⌈$sta($y) ∉ int⌉ then ($val ::= '(0:ėle, 1:exception)))
  ($val ::= ⌈$sta($x) + $sta($y)⌉)))
```

The element *ėle* of the form *(ėle*$_1$ *div ėle*$_2$*)* specifies the quotient of *ėle*$_1$ divided by *ėle*$_2$. Semantics of *ėle* coincides with semantics of the pseudoelement

```
(hvar ($x, $y)
 (seq
  ($x ::= ėle1) (if ⌈ṡta($x) ∉ int⌉ then ($val ::=  '(0:ėle, 1:exception)))
  ($y ::= ėle2)
  (if ⌈ṡta($y) ∉ int \ {0}⌉ then ($val ::=  '(0:ėle, 1:exception)))
  ($val ::= ⌈$sta($x) div $sta($y)⌉)))
```

The elements *(ėle*$_1$ *ȯpe ėle*$_2$*)*, where *ope* ∈ *{-, *, mod}* specifying the integer operations -, *** and *mod* are defined in the similar way.

The element *ėle* of the form *(ėle*$_1$ *< ėle*$_2$*)* specifies the integer relation *<*. Semantics *ėle* coincides with semantics of the pseudoelement

```
(hvar ($x, $y)
 (seq
  ($x ::= ėle1) (if ⌈$sta($x) ∉ int⌉ then ($val ::= '(0:ėle, 1:exception)))
  ($y ::= ėle2) (if ⌈$sta($y) ∉ int⌉ then ($val ::= '(0:ėle, 1:exception)))
  (if ⌈$sta($x) < $sta($y)⌉ then 'true else 'false)))
```

The elements *(ėle*$_1$ *ȯpe ėle*$_2$*)*, where *ope* ∈ *{<=, >, >=}* specifying the integer relations *<=*, *>* and *>=* are defined in the similar way.

7. Conclusion

In this paper the language CTSL of CTSs is proposed and the following kinds of CTSs are defined: CTSs with transition values specifying values of transitions, CTSs with executable elements specifying elements which can be executed, CTSs with execution contexts specifying contexts in which elements are executed, CTSs with counters specifying generation of new

elements which are instances of the given concepts, CTSs with history variables specifying variables which store a history of changing the conceptual *$val*, CTSs with defined conceptuals specifying definitions of conceptuals, CTSs with transition rules specifying executed elements based on the pattern matching and reduction of their execution semantics to execution semantics of other elements, CTSs with types specifying types of elements and literals of these types. Basic predefined elements and operations used in applications of CTSs are also presented.

We plan to use the CTSL language to solve problems of designing and prototyping software systems as well as specification of operational and axiomatic semantics of programming languages.

In the case of specification of operational semantics of a programming language, a CTS specifies the abstract machine of the language.

In the case of specification of axiomatic semantics of a programming language, a CTS specifies a generator of verification conditions for programs in the language, based on its axiomatic semantics.

References

1. Anureev I.S. Conceptual Transition Systems // System Informatics. 2015. Vol. 5. P. 1–41.
2. 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.
3. Gurevich Y. Evolving Algebras. Lipari Guide // Specification and Validation Methods. Oxford University Press, 1995. P. 9-36.
4. Anureev I.S. Operational Ontological Approach to Formal Programming Language Specification // Programming and Computer Software. 2009. Vol. 35. N 1. P. 35-42.
5. Anureev I.S. Ontological Transition Systems // Bulletin of the Novosibirsk Computing Center, Series Computer Science. 2007. Vol. 26. P. 1-17.
6. Anureev I.S. A Language of Actions in Ontological Transition Systems // Bulletin of the Novosibirsk Computing Center, Series Computer Science. 2007. Vol. 26. P. 19-38.
7. Anureev I.S. Domain-Specific Transition Systems and their Application to a Formal Definition of a Model Programming Language // Bulletin of the Novosibirsk Computing Center, Series Computer Science. 2014. Vol. 34. P. 23–42.
8. Anureev I.S. Conceptual Transition Systems // System Informatics. 2015. N 1. (In Russian). (To appear).
9. Anureev I.S. Kinds and Language of Conceptual Transition Systems // System Informatics. 2015. N 1. (In Russian). (To appear).
10. Huggins J. Abstract State Machines Web Page. URL: http://www.eecs.umich.edu/gasm (accessed: 01.09.2015).

11. AsmL: The Abstract State Machine Language. Reference Manual. 2002. URL: http://research.microsoft.com/fse/asml/doc/AsmL2 Reference.doc (accessed: 01.09.2015).
12. XasM — An Extensible, Component-Based Abstract State Machines Language. URL: http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html (accessed: 01.09.2015).

УДК 519.72

Новосибирская летняя школа юных программистов им. А.П. Ершова (21 июля – 3 августа 2014 г., Новосибирский Академгородок – «Белый камень»)

Тихонова Т.И. (Институт систем информатики СО РАН)

В статье представлены материалы о Летней школе юных программистов 2014 года. С 1976 года по инициативе Андрея Петровича Ершова была организована Летняя школа юных программистов. С момента организации ЛШЮП менялись формы и методические особенности проведения занятий. Численность участников варьировалась, менялась техника. Изначально были потоковые занятия с начинающими и уже «продвинутыми» школьниками, консультанты из Академгородка. Следующим этапом стали мастерские… От нескольких минут практики на БЭСМ-6 до многочасового программирования на персональных компьютерах прошло 40 лет. Однако самое главное – атмосфера сотрудничества, направленная на формирование устойчивого стремления к самообразованию, помощь в профессиональной ориентации, основанная на практической деятельности, дающей представление о выбранной профессии, осознанный выбор жизненного пути, развитие творческих способностей, социализация – бережно хранятся. Коллектив участников Летней школы юных программистов – это не только соратники по профессии, но и сообщество единомышленников.

Ключевые слова: *летняя школа юных программистов, школьная информатика.*

1. Введение

Первые занятия факультативного курса программирования с практическими работами школьников на ЭВМ относятся к началу 60-х годов уже прошлого столетия. Их проводили сотрудники Института математики, которые впоследствии стали сотрудниками Вычислительного центра СО АН СССР. Проводились эти занятия на базе школы № 10 г. Новосибирска.

По мере объединения усилий педагогов, математиков, программистов в области создания школьного курса, в котором рассматривались средства программирования и происходило знакомство с электронно-вычислительной машиной, началась деятельность двух инициативных групп. Одной из них была группа по применению вычислительной техники

Научного совета по проблемам образования при Президиуме Сибирского отделения АН СССР. Второй была группа школьной информатики Вычислительного центра Сибирского отделения АН СССР. В составе отдела экспериментальной информатики СО АН СССР было создано первое научно-исследовательское подразделение для формирования концепций и разработки программного обеспечения школьной информатики - группа школьной информатики. Научно-методический семинар «ЭВМ и учебный процесс» начал в стенах ВЦ СО АН СССР свою работу по объединению потенциала науки и образования.

Раннее обучение информатике, ярко проявившееся в интенсивный период 70-80-х годов [1] (проведение Школы юных программистов в Академгородке и двухнедельной Летней школы юных программистов (ЛШЮП), на которую съезжались отобранные школьники, педагоги, ведущие специалисты в области кибернетики и вычислительной техники из всех городов и стран), отражает многочисленные достоинства этого подхода.

С 2001 года ЛШЮП проводится во второй половине июля в течение двух недель. Это время удобно для привлечения в качестве преподавательского состава студентов и преподавателей НГУ, научных сотрудников институтов СО РАН. Мероприятие является выездным. Как правило, это близлежащие к Академгородку туристические и детские оздоровительные центры. Примерно раз в два-три года участники выезжают на Алтай. На сегодняшний день методика проведения занятий показывает отличные результаты. Несмотря на многочисленные возможности, Летняя школа востребована, актуальна, результативна и в плане образовательной составляющей, и в качестве неформального общения, формируя культуру, взгляд на науку и круг единомышленников, занимающихся таким интересным и творческим делом – программированием [2, 3].

Существенно, что сотрудники, аспиранты и студенты НГУ, ИСИ и других институтов СО РАН, преподаватели вузов страны и сотрудники компьютерных фирм принимают участие в работе Летней школы. Развивается эксперимент по обучению в форме мастерских, идея которых сформулирована и внедрена сотрудниками ИСИ. Выявлена заинтересованность молодежи в новых формах экспериментальной работы в области систем информатики.

2. Место и сроки проведения

Новосибирская Летняя Школа Юных Программистов (39-я ЛШЮП им. А.П. Ершова) была открыта в Новосибирском Академгородке в Малом зале Дома Ученых и проведена в течение 2 недель на базе «Белый камень» на берегу реки Катунь, с 21 июля по 3 августа 2014 года.

Важное событие началось, как обычно, регистрацией в холле Малого зала Дома Ученых в Академгородке.

После успешно пройденной регистрации участников приветствовали от Министерства науки, образования и инновационной политики Новосибирской области – заместитель директора Центра «Дио-Ген» Захарова Татьяна Николаевна представители Сибирской науки в лице Директора ИСИ СО РАН Александра Гурьевича Марчука, заместителя Председателя Президиума СО РАН Василия Михайловича Фомина. От Новосибирского государственного университета участников приветствовал ректор Михаил Петрович Федорук. Выступили с поздравлением постоянные участники наших ЛШЮП – Ирина Травина («Софтлаб-Нск»), Виталий Саяпин (новосибирский филиал компании «Интел»). Пожелали плодотворной работы, хорошей погоды, новых идей. От фирмы Excelsior сказал добрые слова о ЛШЮП ее сотрудник, который в этом году закончил аспирантуру ИСИ, мастер на ЛШЮП, выросший из бывших учеников Летней школы юных программистов, Владимир Валерьевич Соловьев.

После открытия всех ждали автобусы и маршрутное такси, чтобы везти в «Белый камень».

3. Цели Школы

Основными задачами ЛШЮП является отбор талантливых старшеклассников, заинтересованных в овладении профессиональным программированием, обучение учеников среднего звена навыкам коллективной работы с применением современных информационных технологий и содействие развитию способностей к практическому программированию учащихся младших классов, а также поддержка педагогов, успешно преподающих информатику и программирование в общеобразовательной системе.

На протяжении многих лет (с 1989 года) Новосибирские ЛШЮП проводятся как школы с углубленным изучением отдельных предметов по выбору: в отличие от ряда летних школ в других городах, они имеют целью не начальное обучение основам компьютерной грамотности или программирования, а развитие профессиональной ориентации школьников, преимущественно старшего возраста. Спецификой этого года организаторы считают отбор учащихся – участников Летней школы – преимущественно среднего звена. Это обусловлено необходимостью приобщения детей к коллективной работе, пропедевтическая работа по изучению основ профессиональной деятельности, а также возможность пролонгированной работы со школьниками. Эта деятельность осуществляется через знакомство с программированием, как с производственной деятельностью, с его проблематикой, методологией, творческими и технологическими аспектами [4]. Новыми понятиями и

объектами для изучения становятся программный продукт, технологический процесс разработки, грамотная постановка задачи и ее формализация, рациональное распределение и планирование работ, отладка, оформление, документирование, отчет.

Для отработки этих понятий учебный процесс в Летней школе рассредоточивается по нескольким (10-15) учебно-производственным мастерским различных профилей - локальным носителям технологических циклов разработки, в которых школьники получают знания и навыки в процессе коллективной работы над единым проектом.

Главной целью мастерской ставится полное прохождение всего технологического цикла в рамках поставленной задачи, с обязательным отчетом о проделанной работе в конце Школы. Необходимая для этого интенсивность работ заставляет уделять большее внимание стадиям проектирования, как со стороны постановщика задачи, так и со стороны руководителя проекта и организаторов Школы. Для многих мастеров, привлекавшихся к работе в Школе, оказывается привлекательна именно возможность апробирования новых методик организации работ и обучения в условиях присущего Школам дефицита времени и техники.

Целями вырабатываемой профессиональной ориентации являются:

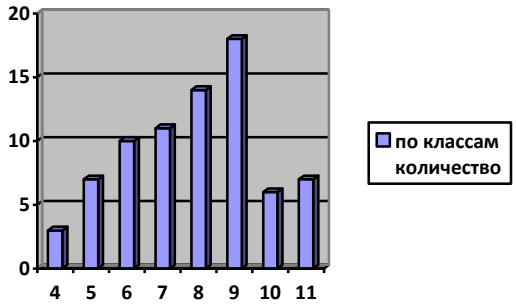
- расширение знаний учащихся о сферах и способах применения компьютерных технологий, типовых задачах и методах их решения;
- определение и уточнение учащимся области приложения своих способностей,
- приобретение специальных знаний и навыков, проба сил в коллективном проекте.

Совокупность тем проектов в Школе подбирается так, чтобы лучше обеспечивать многопрофильность и разноуровневость учебного процесса с целью более адекватной его настройки на индивидуальные наклонности, интересы и способности учащихся.

4. Состав участников

В организации и работе Летней Школы приняли участие 103 человека, 6 человек заезжали как лекторы на краткий срок, а также для подготовки и монтирования компьютерных классов и технической составляющей процесса проведения ЛШЮП, из них 77 – школьники, 10 – студенты, 10 – сотрудники научных институтов и компьютерных фирм, преподаватели вузов, врач.

По возрастным категориям участники были закончившие

4 класс – 3 человека, 5 класс – 7 человек, 6 класс – 10 человек, 7 класс – 11 человек, 8 класс – 14 человек, 9 класс – 18 человек, 10 класс – 6 человек, 11 класс – 7 человек.	

Участники приехали из городов:

Абакана— 2 человека;

Бердска— 5 человек;

Кольцово (Новосибирская область) – 2 человека;

Краснообска (Новосибирская область) 4 человека;

Миасса—5 человек,

Милана (Италия) - 1 человек;

Москвы—2 человека,

Новосибирска— 73 человека,

Санкт-Петербурга – 3 человека,

Самары – 1 человек.

5. Принципы отбора учащихся

Для отработки введения в основы программирования в ЛШЮП традиционно складывается направление раннего вовлечения школьников в процесс обучения как языкам программирования, так и техническим основам информационных технологий. Тем не менее, для сохранения научной основы выполняемых в ЛШЮП работ, а также привлечения технологии создания серьезных производственных проектов, основная возрастная категория – это старшеклассники и продвинутые школьники из среднего звена общеобразовательной школы. Часть детей была приглашена по результатам работы в предыдущих Летних школах. Остальные прошли через отбор посредством участия в командной олимпиаде, мероприятиях

программ центра «Дио-Ген», Новосибирской Областной олимпиады школьников, Каникулярной школы во время осенних каникул, Областной научно-практической конференции школьников в секциях «Информатика» и «Программирование» и других мероприятиях, рекомендованы членами Оргкомитета и преподавательского состава ЛШЮП [5, 6, 7]. Каждый из участников прошел предварительное собеседование и заполнил анкету, разработанную для участников Летней школы в ИСИ СО РАН в мае-июне 2014 года. Собеседование с новосибирскими школьниками проводилось в очном режиме, с иногородними участниками по Скайпу и электронной почте, в Миассе проводили собеседование доверенные лица от ЛШЮП, участвующие неоднократно в ней, в том числе как преподаватели. Очень ценно, что к такого рода мероприятиям, требующим как времени, так и видения действительного состояния подготовки к успешному участию в ЛШЮП, очень активно подключаются выпускники ЛШЮП разных лет, что существенно облегчает процесс отбора школьников.

Отбор областных участников проводился по результатам участия школьников в мероприятиях олимпиадных, в конференциях, а также по результатам работы в Каникулярной школе по информатике, которые организовал ИСИ СО РАН совместно с Центром работы с одаренными детьми администрации Новосибирской области. Младшие школьники были отобраны на командной олимпиаде, проводимой на языке ЛОГО. В основном это ребята, которые учатся в школах, традиционно преподающих информатику с начальных классов (гимназии № 1, гимназии № 3, ЛИТ и некоторых других). Надо отметить активную работу школы г. Обь на ВАСХНИЛе. Ее ученики в начале 2000 годов принимали единичное участие в работе ЛШЮП. Сейчас очень активно ведется работа с раннего школьного возраста по привлечению к программам работы с одаренными детьми.

Информация о вышеупомянутых мероприятиях была распространена через систему повышения квалификации школьных педагогов и семинары по проведению олимпиад по информатике для школьников совместно с районными методистами, размещалась в газетах «Навигатор»», развешивались информационные листки и плакаты в школах, раздавались буклеты на мероприятиях с участием иногородних преподавателей и школьников. Постоянно в течение года информация выкладывалась на сайте Летней школы им. А.П. Ершова (ИСИ СО РАН).

Всего в отборе приняли участие более 140 человек.

6. Состав преподавателей

Традиционно на Школе формируется 14-15 мастерских, исходя из оптимального соотношения "1 мастерская на 3-6 человек". В этом году мастерских было заявлено 15. Но при этом прошлогодний эксперимент – в одной мастерской 2 мастера, распространился на 2 мастерских, что позволило сделать более многочисленный состав, выполнявший большую программистскую задачу. К сожалению, в связи с тем, что накануне ЛШЮП мастер мастерской № 14 заболел, тему не удалось оперативно заменить. Да и, как правило, мастерская – это результат примерно 2-3-недельной подготовки мастера к ЛШЮП заранее, иногда эта подготовка и более длительна, в зависимости от темы и опыта участия. Потому мастер бывает не заменим, возможно, как правило, сменить тему. Что не всегда бывает актуально и правильно.

Список мастерских с описанием приведен в Приложении. Обычно при подборе мастерских основные трудности были связаны с поиском мастеров. Типичными их причинами были: проведение ЛШЮП на удаленной от Академгородка площадке, что некоторым не позволяло совмещать ЛШЮП с производственной деятельностью, июльский пик отпусков, занятость на рабочем месте, и др. В текущем году набор мастеров был осуществлен без видимых трудностей, большую поддержку этому оказал факт того, что бывшие школьники ЛШЮП, став студентами, прекрасно знакомы с методикой преподавания материала и оказываются хорошо подготовленными не только в плане владения необходимым материалом для ведения мастерских, но и психологически готовы работать со школьниками, даже младшего возраста.

7. График работы Школы

Расписанием работы Школы были предусмотрены 1 день – для заезда заранее технических работников с целью открытия компьютерных классов Школы и распределения работ, формирования сети. 1 день для заезда, открытия, презентации мастерских и окончательного формирования списка состава мастерских, 1 день для закрытия ЛШЮП и отъезда участников, 11 учебных дней, 1 день заключительных отчетов мастерских на конференции и демонстрации выполненных проектов, 1 выходной день.

Начинать практическую работу над проектами удалось практически с первого учебного дня, благодаря 2-недельной подготовке техники, установки образов, сервера и т.п. Открытие проведено до выезда в ДУ в торжественной обстановке – этот факт остается важным, в том числе и памятная фотография на ступеньках Дома Ученых. Предварительное распределение по мастерским проводилось до начала ЛШЮП. Корректирование списочного состава

мастерских и вводные занятия проводились в один день. В этом году удалось оперативно справиться с установкой компьютерных классов, что привело к активизации на начале работы в мастерских, где было большее количество учащихся младшего состава. Исходя из опыта прошлых лет, на место проведения ЛШЮП технику вывозили, устанавливали и монтировали сеть заранее, за день до начала учебного процесса на ЛШЮП.

8. Техническая обеспеченность Школы

В распоряжение Школы было порядка 46 ноутбуков и сервер, предоставленных центром «Дио-Ген», ИСИ СО РАН (в том числе мобильный компьютерный класс, переданный в дар от компании HP в 2004 году), НГУ, компании Интел и участниками, – все сохранили работоспособность при переезде на базу – и два принтера.

Компьютеры были распределены по мастерским и эксплуатировались по усмотрению мастеров с 9 до 22 часов дня (3-5 на мастерскую). 1 ноутбук с принтером в распоряжении завуча и преподавательского состава, 1 ноутбук использовался для работы директора.

Конечно, характеристики компьютерной техники быстро устаревают. К сожалению, техническая составляющая ЛШЮП не успевает подстроиться под стремительно развивающееся программное обеспечение. Хочется показать школьникам как можно более новые современные технологии. Но это требует и современного оборудования, которое, к сожалению, не является на сегодняшний день настолько доступным в финансовом плане, чтобы свободно получить достаточное количество для реализации смелых замыслов мастеров. В случае с задачами сложными и большими выручали ноутбуки, которые предоставили в общий доступ некоторые мастера и подмастерья. С мастерскими, выполняющими регулярные учебные проекты в привычных для общеобразовательного процесса средах таких проблем не было.

Благодаря хорошей технической оснащенности все мастерские успешно справились с работой в мастерских и получили хорошие демонстрируемые результаты.

Компьютеры каждой мастерской были объединены в локальную беспроводную сеть. Машины мастерских, которым был необходим доступ к серверу, его получили.

Отмечено, что, несмотря на наличие принтера в общей доступности для оперативной распечатки в начальный период работы мастерских из-за потребности в документации при отсутствии подобранной мастером литературы, им практически пользовалась 1 мастерская.

Возможность распечатки итоговых отчетов была обеспечена всем мастерским. В распоряжении Школы были, на сей раз, ксерокс и сканер. Эти устройства были весьма

полезны при работе со столовой и дирекцией базы. Для обеспечения освещения работы Летней Школы использовались цифровые фотокамеры и видеокамеры, в том числе принадлежавшие участникам.

Кроме связи с родственниками для участников привлекательна перспектива развития Школы в направлении дистанционного взаимодействия через Интернет со специалистами, не имеющими возможности выехать на базу Школы. Хотя, если смотреть глобально на выездную работу со школьниками, очень хочется, чтобы в наличии был постоянный скоростной Интернет для выкладывания на сайт фотографий, видеороликов, да и элементарно каких-то оперативных сообщений для общественности и родителей.

9. Программные средства, использованные на Летней школе

В этом году больше половины мастерских в процессе работы изучали операционную систему Linux. В работе использовались компилятор языков C/C++ (gcc), интерпретаторы языков Perl, Java и др., веб-сервер Apache с поддержкой PHP. На компьютерах под управлением Windows 2000/XP использовались компиляторы Microsoft Visual Studio 6.0 и Borland C++, Builder, а также среда программирования Флэш и Скретч, Лого.

10. Учебный процесс

10.1. Структура учебного процесса

Основной формой работы в ЛШЮП является выполнение поставленной задачи в рамках работы мастерской, где выполняется учебно-производственный процесс. Спектр мастерских этого года получился разнообразным, на любой вкус и начальные знания, всего их было 15.

В общеобразовательный цикл входили лекции и спецкурсы по языкам и системам программирования, обзорные лекции по перспективам и проблемам программирования, истории информатики и дисциплинам, которые позволяют расширить кругозор учащихся во многих областях науки, а также ежедневная «Задача дня» - олимпиада по решению алгоритмических задач. Учебное время экономилось за счет совмещения по времени занятий по языкам программирования, спецкурсов и учебной работы по мастерским.

Лекции – это инструмент, который позволяет формировать не только кругозор, но и формирует культуру восприятия научной информации, проводились в дневное время.

Некоторые занятия сопровождались демонстрацией программных изделий и практикой по работе с ними для желающих.

Работа по мастерским проводилась в виде практических занятий на ноутбуках и семинарских занятий, в ходе которых велась подготовка к практике, разбирались теоретические вопросы, связанные с тематикой мастерских, проводились консультации, анализ работы и изучение необходимых языковых конструкций. Эти занятия проводились по индивидуальным планам мастеров, которые ориентировались как на поставленную задачу, так и на возрастные особенности учащихся и их подготовленность к работе по тематике мастерской.

Учитывая увеличивающуюся долю новичков среди участников Школы и снижение возрастного барьера детей, учебные планы части мастерских были целенаправленно реорганизованы таким образом, чтобы наиболее эффективно работать по практическому изучению отдельных инструментов на примере решения прикладных задач.

10.2. Распределение по мастерским

Предварительное распределение по мастерским было организовано на основе анкетных данных и проводимых собеседований с учащимися. Учитывалось желание школьников работать по той или иной теме. О направлениях работ можно было узнать на сайте Летней школы и по электронной почте. Настоятельно рекомендовалось в случае недостаточной подготовки обратить внимание на подходящую по уровню первоначальных требований мастерскую и организовывались встречи с большинством мастеров (некоторые были доступны только в режиме электронной переписки). Во время организационного собрания (до начала проведения Летней школы) перед учащимися и родителями было сделано представление мастерских мастерами, которые подробнее рассказали о предстоящей работе. В первый день работы Школы состоялись представление проектов и окончательное распределение по мастерским. Тематика и состав мастерских приведены в приложении.

В этом году большинство состоявшихся мастерских подготовили и вывесили заранее на сайте рекламы-информации о мастерской и некоторые заинтересованные в уровне участников своей мастерской специально выделяли время для присутствия на собеседовании со школьниками. У некоторых собеседование мастеров с кандидатами в ученики свелось, в запись всех желающих. В результате в первый день ЛШЮП нескольких учащихся пришлось заново распределять в другие мастерские в связи с тем, что в сложившемся очень сильном коллективе они не могли бы наилучшим образом реализовать свои возможности.

Дифференцирование уровней сложности проектов на ранней стадии позволила провести адекватную дифференциацию уровней подготовки учеников, из-за чего продвинутые мастерские могли хорошо реализоваться в выбранной тематике на Школе. Более слабые по составу мастерские реализовали в полной мере творческий потенциал участников.

Отмечен высокий интерес школьников к работе в мастерских по разным направлениям. Мастера в целом выразили удовлетворение высоким темпом обучения школьников.

10.3. Работа мастерских

Качество выполненных работ в мастерских этого года было очень высоким и презентативным. Некоторые мастерские выполнили работы на уровне бакалаврских. В мастерских по программированию на Флэш и на Лого, несмотря на возрастной состав, также очень значительно выглядел результат, особенно впечатляла проектная работа школьников, а также, безусловно, интересные программистские задачи, оформленные в проектную работу.

Работа мастерских проходила довольно ровно, существенных сбоев ни у кого не было. Почти не было необходимости авральной работы в конце Школы, и все мастерские успели выполнить все запланированное. Большинство мастеров выполнили качественную подготовку к отчетам, на что времени обычно не хватает, а учащиеся подготовили презентации выполненных проектов в мастерских.

В ряде случаев содержательная часть работы оказалась сделанной уже к концу Школы, причиной чего было названо завышение изначально запланированного объема работ и уровень продвижения учащихся по изучаемой тематике. При этом время было с меньшей эффективностью направлено на доработку внешнего интерфейса, реализацию дополнительных режимов работы, создание демонстрационных версий и написание документации.

10.4. Отчеты мастерских

Промежуточные результаты работы в мастерских до итоговой конференции позволили примерно оценить ожидаемые результаты, дать предварительные оценки работы всем учащимся, выявить слабые места в работе мастеров и вовремя их подкорректировать. В этом году практически все мастерские успели подготовить демонстрационные версии, некоторые - довольно качественную документацию.

Жюри и заинтересованные слушатели посмотрели выполненные в мастерских работы и оценили высокий уровень реализованных проектов.

Итоговая конференция, прошедшая в виде демонстрации рекомендованных Жюри презентаций всем участникам Школы, вызвала большой интерес и активное обсуждение.

Мастерские представили отчеты, отражающие постановку задачи, распределение работ и полученные результаты. Большую роль здесь сыграла возможность электронных отчетов в виде сайтов и презентаций, подготовленных заблаговременно для всеобщего обозрения: качество и содержательность отчетов, стали лучше. Сводная демонстрация программ на итоговой конференции также имела важное значение для всех участников, продемонстрировав, что можно сделать даже за короткое время, а также уровень лучших работ. На демонстрацию были выдвинуты все работы, имевшие наглядный демонстрационный вариант.

Состоялось подведение итогов работ по мастерским с их полным составом.

10.5. Оценка работы мастерских

Для оценки работы мастерских и подведения итогов Летней школы работало Жюри в составе: А.Г. Марчук (председатель Оргкомитета), Т.И. Тихонова (завуч), Б.Л. Файфель (председатель Жюри), В.А. Сакерин.

Жюри в первую очередь оценивается рост уровня знаний учащихся и их качество, первоочередное значение имеет формулирование цели эксперимента и обоснованность полученного результата. Кроме того, оцениваются потребительские качества программного продукта: удобство пользовательского интерфейса, отладка, предоставление документации, презентабельность.

Другими, общими параметрами оценки были: качество отчетов, понимание учащимися задач и состояния дел в своей мастерской, своей роли в ней. Жюри высоко оценило работу, выполненную всеми мастерскими и, в отличие от прошлых лет, было выдано рекордное количество поощрений. Высшей награды – диплома за успехи в программировании – в этом году были удостоены школьники не всех мастерских. При этом в других мастерских получили высшие награды более половины школьного состава.

Жюри было высказано пожелание о привлечения всех мастеров к обсуждению оценки мастером результатов работы его мастерской.

10.6. Научно-педагогический опыт работы мастерских

В небольших группах под руководством опытных программистов–практиков дети работают над оригинальными проектами, параллельно осваивая новые компьютерные инструменты, технологии и приобретая бесценный опыт работы в команде. Задача мастера не только научить, но и создать обстановку, чтобы каждый участник проекта развивался сообразно своим интересам, возможностям и стартовому уровню. Этот уровень может быть различен, но неизменным требованием для участников ЛШЮП является знание языков программирования и наличие навыков программирования.

Координацию учебной работы и мастерских ведет завуч. В этом году работало 15 мастерских, их тематика, как обычно, по различным направлениям информационных технологий, охватывают возможности как для младших, так и для продвинутых детей.

Год от года состав мастерских меняется в соответствие с подбором как мастеров, так и по велению времени, согласно уровню развития информационных технологий. Создание мастерских для детей 9-11 лет продолжается. Данное направление стало хорошей традицией ЛШЮП. Некоторые студенты, прошедшие обучение в таких мастерских, на нынешних Летних школах с успехом преподают, в том числе и ведут мастерские для младших школьников. В этом году для младших детей работало 2 мастерских: проекты на Лого и Флэш-технологии. Дети младшего школьного возраста успешно справились с задачами, уверенно ориентируются в делах Летней школы. Их проекты вызывают восхищение даже взрослых преподавателей вузов своей техничностью и основательностью.

Для выполнения основной задачи осуществляется «погружение» в проблему, но при этом для каждого участника мастерской мастер уделяет достаточное внимание и находит индивидуальные методы и средства работы для наиболее эффективного профессионального роста школьника. Заметный эффект при этом стимулируется поддержкой самостоятельности в работе и упором на коллективное взаимодействие учащихся не только в рамках мастерской.

В целом работа мастерских, по отдельности и в совокупности, была расценена как вполне успешная.

10.7. Учебно-лекционный цикл

Учитывая пожелания прошлых лет, учебная программа этого года предусматривала более систематический и полный обзор проблем, методов, стиля и форм программирования.

Проведение Летней школы на удаленной площадке, малочисленность преподавательского состава и сжатые сроки ее проведения не приспособлены для полного учебного цикла по языкам программирования, операционным системам и типовым технологиям.

Реальный учебный план Школы не предусматривает подробного изучения каких-либо средств, а только краткие курсы, обеспечивающие включение всех учащихся в работу мастерских. В их числе вводные курсы по различным языкам программирования (по мастерским) и системам. Было знакомство с операционной системой Linux для тех, кому это необходимо в работе. Проведены открытые семинары (в которых могли принимать участие не только участники конкретной мастерской, но и те, кому просто интересно или полезно с точки зрения их мастера послушать тему). Значительное внимание было уделено обзорным лекциям по истории информатики, ее современным перспективам и исторически оправдавшимся концепциям. Основная задача таких лекций - пробудить интерес и дать импульс для самостоятельного обучения после окончания Летней школы.

К сожалению, проведение ЛШЮП на отдаленной площадке затрудняет приезд лекторского состава на протяжение работы всей школы. Тем не менее, лекции проводились практически ежедневно.

В этом году удалось организовать замечательный цикл лекций. Очень ценной оказалась подборка материалов Б.Л. Файфеля, оформленная в виде презентаций (эти материалы использует для чтения лекций в Саратовском политехе студентам). Лекция была посвящена кодированию и сжатию информации Лекция А.Г. Марчука по теме «Теория решения изобретательских задач» не только заинтересовала подходом, но и дала возможность расширить кругозор относительно того, что ученые всего мира трудятся в одном направлении. Традиционными для ЛШЮП стали презентации IT-компаний Представители извесных фирм Интел, Excelsior и Яндекс рассказали соответственно о технологиях/ Их темы «Каково быть программистом: хороший программист, крутой девелопер», «Свобода или безопасность: Чем Ява лучше Си», и «Модель распределенных вычислений MapReduce». Представитель фирмф Майкрософт рассказал о «Функциональном программировании для самых маленьких». Кроме специализированных лекций, была лекция «Эхо первой мировой» в целях напомнить об истории Отечества, мировой истории, а также в связи со столетием ее начала.

Участникам ЛШЮП было небезынтересно познакомиться с фирмами, где им, возможно, предстоит работать в недалеком будущем. Само по себе внимание столь солидных

организаций к работе ЛШЮП является показателем признания ее роли в подготовке программистских кадров.

10.8. Контроль уровня знаний и навыков

Жюри провело приемку промежуточных результатов работы мастерских непосредственно на рабочих местах. Затем была проведена конференция по итогам работы всех мастерских и демонстрация работ.

Определенный контроль дает ежедневная олимпиада, активность в которой, в отличие от предыдущих лет, медленно возрастала в первые дни Школы, но повысилась в последующем после нарастания интенсивности работы в мастерских, несмотря на выделения потенциальных чемпионов.

Каждая мастерская представила отчет о проделанной работе в виде презентаций. Мастерам было предложено сделать анализ работы каждого школьника на Летней школе и дать рекомендации для них, также предоставить отчеты мастеров об этапах и итогах работы в мастерской.

10.9. Конференция по результатам работы

Итоговая конференция Летней школы прошла в лучших научных традициях. Докладчики представляли выполненные в мастерских проекты, участники конференции задавали вопросы, содержание которых говорит о искреннем профессиональном интересе к представленным разработкам и о квалификации слушателей.

Работа по повышению культуры публичных выступлений на ЛШЮП ведется постоянно, готовятся инструкции для мастеров, которые с должным вниманием должны помочь подготовить выступление школьников, но не презентовать работу мастерской сами. Общеизвестно, что любителей выступать среди программистов мало, большинство предпочитают безмолвно демонстрировать на компьютере результаты, качество которых зрителям трудно оценить. Но успех в любой профессии существенно зависит от искусства представления результатов. Поэтому участие в итоговой конференции обязательно для всех мастерских.

На конференции рассматриваются полученные результаты и выбранные технические решения. Школьники обмениваются рекомендациями по улучшению их разработок. Важную

роль играет личность председателя конференции, задающего уважительный стиль общения и обсуждения.

Оценивается качество доклада, уровень вопросов и ответов, активность обсуждения. Все это учитывается в итоговых формулировках наградных грамот.

11. Методическая работа

Непосредственно перед проведением Летней школы было проведено специально запланированное собрание мастеров ЛШЮП. Было рассказано об особенностях детского коллектива в разных возрастных группах. Надо отметить своевременность и необходимость такого рода мероприятий с учетом того, что большинство преподавателей Летней школы являются студентами и выпускниками НГУ, не знакомыми с педагогическими аспектами обучения школьников. Беседа по детской психологии и социологии необходима для всех, особенно для студентов, не имеющих не только жизненного опыта, но и педагогического образования.

Для взрослых участников Школы была проведена серия специальных семинаров по проблемам подготовки и проведения Школы, на которых обсуждались вопросы проведения этой ЛШЮП и школ вообще, цели, формы и способы обучения, перспективы проведения Школы в современных условиях.

В целом, привлечение мастеров и преподавателей к методической работе по организации учебного процесса Школы происходило интенсивно, большая часть решений принималась в коллегиальном порядке с учетом прошлого опыта.

В течение всей работы ЛШЮП проводилась ежедневная олимпиада по решению небольших алгоритмических задач.

12. Поощрение участников

По окончании Школы участникам выдавались сертификаты. Высшей награды – диплома за успехи в программировании – в этом году были удостоены более 20 школьников. Почти каждому участнику, включая взрослый состав, была вручена грамота. Большое внимание было уделено наградным формулировкам в дипломах и поощрительных грамотах. Формулировки индивидуально отражают характер достижений каждого участника, тонко подмечены личные свойства, положительные и проблемные.

Помимо этого, каждый школьник получил в подарок футболки с логотипами ЛШЮП и «флэшки» с записанными материалами ЛШЮП, кроме того, были подарки и призы от спонсоров, которые выдавались не только на закрытии, но и во время всей ЛШЮП по итогам различных мероприятий.

13. Культурная и спортивная программы. Организация досуга

Желающие имели возможности для занятий спортом на открытом воздухе (утренняя зарядка, футбол, настольный теннис и др. спортивные игры).

Был организован костер и последний день пребывания на базе. Во время учебных дней проведены игры «Что? Где? Когда?». На третий день организовали представление мастерских. Несмотря на временные трудности, представление было легким, интересным и приятным во всех отношениях. Это дало возможность показать себя участникам с артистическими талантами (гитара, песни, сценки). Ставший традиционным для выходного дня Летней школы КВН, прошел весело, интересно, с большим количеством шуток и прекрасно исполненными музыкальными номерами. На Летней школе во время выходного дня был организована экскурсия в Республиканский музей с выездом в Горно-Алтайск. Программа ЛШЮП была дополнена песнями под гитару.

14. Жилищно-бытовые условия, питание, транспорт

Большое внимание организаторы ЛШЮП уделяют бытовым вопросам: пятиразовое питание, проживание, личная гигиена и медицинское обслуживание находится под неусыпным наблюдением соответствующих специалистов. Такие оздоровительные мероприятия, как утренняя зарядка, прогулки на свежем воздухе – обязательная составляющая ЛШЮП. В комнатах жило от 2 до 4 человек. Приезжавшие на краткий срок гости поселялись в гостинице. Питание участников Летней Школы было организовано в столовой турбазы. Следует отметить хорошее качество питания и хороший уровень обслуживания. Качество питания и состояние здоровья участников было под контролем профессионального медика, неоднократно работавшего на Летних школах.

Хуже всего была обеспеченность Летней школы транспортом. Транспорт оказался дорогим для доставки участников и техники. Организаторы решили часть транспортных затрат возместить за счет спонсорских средств. По-хорошему, надо специально планировать

наличие автомобилей в распоряжении школы, необходимо, по крайней мере, два рейса для заезда-выезда приглашенных участников, привлекаемых на два-три дня.

Рекомендации по подготовке личных вещей и своевременный прогноз погоды участникам был предоставлен заранее.

Место базы ЛШЮП находилось в прекрасном в экологически чистом месте, вблизи водоема река Катунь. Свежий сосновый воздух способствовал оздоровлению. Постоянное пребывание на свежем воздухе позволило минимизировать простудные заболевания.

15. Формы управления, самоуправление

Летние Школы юных программистов традиционно ведут поиск удобных форм самоуправления. Работу по обеспечению работы компьютерной сети вел А. Салмин, К. Лихтер осуществлял техническую поддержку системы регистации участников ЛШЮП, за технические работы по подготовке техники отвечал Д. Тумайкин,, с помощью Д. Горбунова и И. Дульцева. Постановку учебной работы осуществлял завуч Т.И. Тихонова. Работали культорг А. Анкудинова, за состоянием здоровья следила Т.Н. Смоляк, за быт отвечала И.В. Занина. В течение года вел сайт А. Лысцов, подготовку вступительных заданий осуществила Е. Дмитриева.

Для оценки работы мастерских и подведения итогов Летней школы работало жюри в составе: Б.Л. Файфель (председатель Жюри), А.Г. Марчук (председатель Оргкомитета), Т.И. Тихонова (завуч), В.А. Сакерин. Координацию работы мастерских вел завуч, действуя по опыту аналогичной работы прошлых лет.

Вопросы подготовки и ведения Школы, а также ее итоги обсуждались на организационных собраниях и планерках соответственно. Постоянно велась работа по переписке и обмену мнениями как по электронной почте, так и на сайте ЛШЮП.

Значительный обьем как организационно-технической, так и преподавательской работы ведут студенты и магистранты, показавшие себя эффективным и трудоспособным звеном Летней школы. Целесообразно при подготовке следующих школ уделить особое внимание привлечению к работе большего числа студентов. Отдельно следует отметить, что среди студентов и школьников удавалось найти специалистов с весьма глубокими познаниями.

16. Сравнение с предыдущими Школами

В этом году надо отметить снижение количественного состава старшеклассников. Тем не менее, ведущаяся многолетняя работа по снижению возрастного барьера школьников, занимающихся программирование, может быть признана успешной. Ребята, начавшие участвовать в процессах ЛШЮП с 4-5 класса, начинают понимать и осваивать премудрости профессии в более раннем возрасте – в этом году основной когортой были школьники 7-9 классов.

Выездной вариант дает ограничение численности, но зато все участники вместе, нет проблемы, что кто-то неожиданно уехал домой или, наоборот, привез друзей.

Отрицательно сказывается на Школе снижение доли иногородних участников, особенно взрослых. Этот факт сказывается на разнообразии Школы, полезных внешних связях Оргкомитета, способствует замыканию Школы в себе. Надо отметить, что в этом направлении намечены некоторые подвижки – участие иногородних желающих снижается в пользу новосибирских участников. В этом году был продолжен опыт распространения опыта ЛШЮП на площадке республики Хакасия (В.А. Сакерин). Из Саратовского политеха приехал для участия в проведении ЛШЮП Б.Л. Файфель, из Миасса – выпускник ЛШЮП 1992 года М.В. Братусь. Кстати, двое из них еще и являются неформальными руководителями делегаций Хакасии и Челябинской области, искренне заинтересованные в том, чтобы ребята повышали свою квалификацию в ЛШЮП. Иногородние студенты, продолжившие обучение в престижных вузах столиц, с энтузиазмом приезжают в Новосибирск для проведения ЛШЮП, также помогают с проведением на местах с собеседованием участников.

К достоинствам следует отнести возможность общения участников в любое время, не только на занятиях. Это повышает нагрузку на преподавателей, но существенно повышает эффективность Школы.

Надо отметить, что все школьники, принимавшие участие в Летней школе им. А.П. Ершова, являются отобранными по критерию подготовленности и готовности заниматься серьезными вопросами в области предварительной подготовки учащихся к профессии.

17. Планирование следующей Летней школы

В следующем году Летняя школа будет юбилейной – 40-й. Хотелось бы, безусловно, это важное событие подготовить соответствующим образом. Например, провести в рамках ЛШЮП симпозиум людей, в разные годы принимавших участие в ней как в качестве руководителей.

Формирование рабочей группы оргкомитета из числа студентов приводит к естественной смене состава. Поиск мастеров, привлечение активных производственников, владеющих современными технологиями, согласование времени проведения Летней школы, поиск удобной площадки для ее проведения, учет специфики работы Летней школы на дальней площадке – все эти вопросы решались заблаговременно. Тем не менее, некоторое напряжение по поводу вопроса участия того или иного мастера в некоторых случаях остается неподтвержденным вплоть до начала ЛШЮП.

Очень эффективна работа мастеров, прошедших ЛШЮП в качестве школьников и подмастерьев.

Безусловно, напряженным остается момент по поводу утверждения финансовой поддержки ЛШЮП, особенно это сказывается в условиях проведения конкурсов и котировок.

18. Организации, поддержавшие проведение Летней школы юных программистов

Организация Школы стала реальной благодаря целевому финансированию со стороны Президиума СО РАН, ГАОУ ДОД НСО «Центр развития творчества детей и юношества» Областного центра работы с одаренными детьми «Дио-Ген» Министерства науки, образования и инновационной политики Новосибирской области, усилиям Института Систем Информатики СО РАН в организации Школы и сборе для нее дополнительных средств. Сбор финансов был организован своевременно. В этом году было дополнительное поступление средств от спонсоров компании Эксельсиор, Софтлаб, личных пожертвований частных лиц.

Дополнительные средства и призы были выделены Софтлаб, Ledas, Яндекс, ВМК МГУ, Информационная поддержка осуществлялась газетами «Навигатор», «Наука в Сибири», другим СМИ.

Всем организаторам, а также спонсорам, содействовавшим организациям и физическим лицам Оргкомитет выражает глубокую признательность и благодарность.

19. Заключение

Новосибирск – огромная индустриальная держава, которая очень нуждается в высококвалифицированных кадрах в области информационных технологий. Общепризнанно, что участники Летних школ юных программистов значительно более выигрышно выглядят

на фоне других студентов в области информационных технологий и как программисты представляют когорту высококвалифицированных специалистов. Для СО РАН проведение Школы является важным механизмом привлечения талантливой молодежи в сферу влияния науки и развития отечественной информационной индустрии. Для родителей ЛШЮП дает удачную форму сочетания летнего отдыха школьников с получением интересных знаний и востребованных навыков. Для информационной индустрии механизм ЛШЮП дает полигон для ранней профориентации школьников, а также для сочетания смены деятельности специалистов с вольным экспериментированием и поиском будущих помощников.

Для НГУ и, в частности, для кафедр "Программирование" и "Вычислительные системы" НГУ, а также ВКИ и СУНЦ НГУ проведение Школы – это обкатка методик раннего обучения современной информатике; привлечение в НГУ абитуриентов, интересующихся программированием, способных в будущем участвовать в конкурсах и научных проектах; рост профессионального уровня студентов ВКИ и СУНЦ НГУ; привлечение к преподавательской деятельности студентов, приобретение студентами навыков работы в качестве руководителей проектов и постановщиков задач;

Для администрации Новосибирской области представляет интерес, что разрабатывается механизм выездной работы со школьниками, изучающими информатику. Этот механизм может быть распространен на сельские районы, способствует повышению уровня подготовки сельских участников в вузы. Также в Новосибирск привлекаются иногородние школьники, которые получают возможность приехать в последующем в качестве абитуриентов в новосибирские вузы

Для ИСИ СО РАН существенно, что сотрудники, аспиранты и студенты ИСИ приняли участие в работе Школы, выполнено несколько проектов по темам, разрабатываемым сотрудниками ИСИ, развит эксперимент по обучению в форме мастерских, идея которых сформулирована и внедрена сотрудниками ИСИ, выявлена заинтересованность молодежи в новых формах экспериментальной работы в области систем учебной информатики, а именно, предлагается эксперимент по организации Школы программирования (воскресной и вечерней, дистанционной) для наиболее подготовленных мастерских в течение учебного года, продолжает развиваться традиционный механизм, потеря которого может повредить интересам ИСИ в отношении плановой темы "Исследование основ информатики и методов преподавания информатики и программирования", привлечено внимание к методическим наработкам.

Иногородние участники имели возможность общения по интересам и повышения квалификации.

Показателем стабильности и эффективности работы Школы является то, что школьники и мастера общаются после Летней школы лично и на ее сайте, не один год участвуют в работе ЛШЮП. Не только ради учебы и коллективной работы школьники хотят вернуться в ЛШЮП. Привлекает демократичный доброжелательный микроклимат, который делает неуместными даже простые детские шалости. Каждый, от директора до самого юного участника, осознает свою причастность к созданию атмосферы, где приветствуется активно-сознательное отношение к делу и досугу. Школа оставляет яркий след в жизни учащихся, и потом они возвращаются в нее в качестве мастеров, их помощников, организаторов. Школа помогла с определением будущей специальности многим старшеклассникам; они, теперь уже выпускники ВУЗов, составляют цвет программистского сообщества в нашей стране и за ее пределами.

Познакомиться с материалами Летних школ юных программистов можно на сайте http://school.iis.nsk.su.

Список литературы

1. Г.А. Звенигородский. Первые уроки программирования. // Под ред. А.П. Ершова. М.: Наука. Главная редакция физико-математической литературы, 1985. 208 с. (Библиотечка «Квант». Вып. 41.).
2. Марчук А.Г., Тихонова Т.И., Городняя Л.В. Новосибирская школа юных программистов. //Материалы международной конференции «Развитие вычислительной техники в России и странах бывшего СССР: история и перспективы». Петрозаводск, 2006. Часть 2. С. 117-124.
3. Тихонова Т.И. Работа с одаренными детьми в области программирования и информационных технологий // Материалы VI международной конференции «Перспективы систем информатики», Новосибирск, 2006. С. 86-90.
4. Марчук А.Г., Тихонова Т.И. Традиции в системе подготовки творческой молодежи. // Компьютерные инструменты в школе. Санкт-Петербург, 2008. № 2. С. 3-11.
5. Перкова В.Г., Сапрыкин Э.Э., Сапрыкина Г.А., Тихонова Т.И. Дистанционное обучение в контексте развития творческих способностей школьников. //Материалы VI международной конференции «Перспективы систем информатики». Новосибирск, 2006. С. 69-73.
6. Тихонова Т.И. От алгоритмов – до проектной деятельности. // Педагогические заметки. Новосибирск: ИПИО РАО, 2012. Т5, Вып. 3. С. 40-47.

7. Тихонова Т.И. В какую информатику будем играть? // Вест. Новосибирского гос. ун. Сер.: Информационные технологии. Новосибирск: НГУ, 2012. Т.10, Вып. 2. С. 100-105.

Приложение А. Аннотации мастерских 2014 года

1. Мастерская "Программируем с Черепашкой"

Мастер: Дмитрий Горбунов

Подмастерье: Сергей Кузькоков

В мастерской планируется обучение младших школьников началам программирования на примере языка Лого. Некогда популярный язык программирования Лого с простым синтаксисом и интуитивно ясной графической составляющей (так называемой черепашьей графикой) хорошо зарекомендовал себя как первый язык для изучения основ процедурного программирования.

В ходе работы мастерской будут освоены основные конструкции и принципы программирования (условия, циклы, вложенные циклы). Также планируется освоение и использование рекурсии. По возможности будет разобрана концепция структур данных. За время Летней Школы будут подробно разобраны и изучены задачи олимпиад по Лого для 5-7 классов.

Инструментарий: FMSLogo, возможно LaTeX.

2. Мастерская "Система online-соревнований"

Мастер: Владислав Ретивых

Подмастерье: Анастасия Голованова

Многим наверняка случалось участвовать в различных online-соревнованиях (писать олимпиады/сдавать тесты/проходить квесты/...). Устройство некоторых таких соревнований было терпимым, некоторых - нет. Мы напишем систему, в которой будет удобно как проходить соревнования, так и создавать их.

Нашим проектом станет WSGI-приложение, написанное на языке python, предоставляющее web-интерфейс для создания произвольных соревнований с любым количеством задач нужного вида, сдачи ответов к этим соревнованиям, а также просмотра результатов каждого участника.

Результатом работы мастерской будет система, которая:

- позволит любому желающему создать и провести своё соревнование/тест;
- будет различать варианты ответов (текстовые/выбор одного/выбор нескольких);
- предоставит удобный интерфейс для прохождения соревнований;
- будет иметь авторизацию.

В рамках мастерской мы:

- изучим язык python;

- научимся работать с базами данных;

- познакомимся с HTML и JavaScript .

Требования к участникам: >8 класс, знание любого процедурного языка программирования (желательно C или Pascal)

Инструментарий: ubuntu, python, vim/sublime text, git.

3. Мастерская "Учи меня везде"

Мастера: Шумаков Алексей, Смиренко Кирилл

Сегодня все школьники озабочены подготовкой к экзаменам (ОГЭ и ЕГЭ). При этом, помимо услуг репетиторов, многие пользуются системами онлайн-обучения. В нашей мастерской мы сами создадим систему удалённого обучения, позволяющую учителям создавать и модерировать курсы, а ученикам - дистанционно обучаться (это будет что-то похожее на 100ege.ru). В рамках мастерской мы:

- изучим язык C#;
- познакомимся с основами ООП (объектно-ориентированного программирования);
- познакомимся с основами проектирования ПО;
- научимся работать с сетью;
- возможно, научимся использовать мультимедиа в программах (аудио, видео).

Требования к участникам: 7-9 класс, знание любого процедурного языка программирования (pascal, C и т.д.)

Инструментарий: Visual Studio 10 и выше, Git.

4. Мастерская «WOW Исполнители на JavaScript»

Мастер: Братусь Михаил

Подмастерье: Василий Колобов.

Мастерская разработает несколько исполнителей, которые будут доступны на любых устройствах с поддержкой HTML5 (ПК, планшет, смартфон). WOW исполнители будут разительно отличаться от привычных школьных системами команд и своими средами.

- изучим элегантный JavaScript «strict mode»
- освоим фреймворк Cocos2D HTML5 и др.более простые

- основы HTML, CSS, DOM
- создадим несколько исполнителей, которых можно будет использовать на олимпиадах
- если постараемся, то создадим рабочее место преподавателя для мониторинга ученических ПК с исполнителями (выдача заданий и проверка решения)

Требования к участникам: 7-9 класс (важнее знания, а не класс), опыт программирования на процедурном языке программирования, знакомство с HTML, CSS (вы можете начать это знакомство прямо сейчас)

Инструментарий: Блокнот, FireFox, Firebug, Chrome, Paint.Net, Git.

(возможно для IDE, что-то вроде Sublime Text и WebStorm).

5. Мастерская «Summer School of Tanks»

Мастера: Соловьев Владимир Валерьевич, Тумайкин Данил Михайлович

Подмастерье: Рина Качар

Суть такова: вы любите танчики, мы любим физику. Давайте сделаем это вместе! Мы объединим все лучшее от TeeWorlds и WorldOfTanks! Мы напишем сетевую 2D игру, в которой можно будет ездить на танках, стрелять (пиу-пиу) и грабить "корованы". Мы прикрутим к игре настоящие физические формулы (E = mc2), так что снаряды будут лететь куда надо, а танки падать в пропасти и взлетать в небеса. У нас даже будет редактор карт (ну, если его напишете ВЫ)! Как Warcraft 2, только на танках.

Плюшки: будет резиновая бомба, порталы и сарай с гусями (гуси с AI)!

Цели мастерской: освоить технологии рендернига и сетевого взаимодействия, реализовать физическую модель, укрепить знание C и, возможно, познать C++.

Требуемый уровень школьников: 8 класс и выше, знание C.

Было бы неплохо: знание ООП-языков программирования, умение и желание рисовать танчики и вспоминать уроки физики.

Инструментарий: Visual Studio, FAR, один из популярных движков рендеринга (например, SDL), git.

6. Мастерская "Искусственный интеллект"

Мастер: Борис Леонидович Файфель

В далекие теперь 60-е годы прошлого века исследователям казалось, что искусственный (компьютерный) интеллект вот-вот будет создан. И для этого были веские основания:

придуман замечательный язык Лисп (на котором можно писать самообучаемые программы), и была написана программа, общаясь с которой, человек мог бы долго не догадываться о том, что его собеседник не одушевлен. Автор программы Дж. Вейценбаум назвал эту программу “Элиза”. Участникам мастерской предлагается создать такую программу общими усилиями; научить ее русскому языку, сделать приличный интерфейс пользователя. Все это мы выполним на языке Лисп. Вы не знаете Лиспа? Ничего страшного – работу мы начнем с изучения языка. Возможные участники мастерской: школьники от 6-7 класса и выше. Предварительных знаний Лиспа не требуется. Желательна любовь к математике и хороший кругозор.

7. Мастерская "Игры разума"

Мастер: Анна Анкудинова

Подмастерье: Елена Титиевская

Целью мастерской является изучение основ языка C/C++ и основных алгоритмов, таких как сортировки, алгоритмы на графы и на списки. Для применения полученных знаний мы напишем “искусственный интеллект” для логической игры калах.

Мастерская расчитана на учеников 5-7 класса (примерно), знание какого-нибудь языка программирования приветствуется.

Используемые языки: C/C++.

Инструментарий: Windows, Visual Studio, может быть SVN.

8. Мастерская "Поисковая машина"

Мастер: Климов Николай

Подмастерье: Арсений Цыпушкин

Сегодня сложно представить как бы мы жили в сети без таких сайтов как google и яндекс. Они настолько прочно вошли в нашу жизнь, что название одного их них даже стало глаголом. И при этом только 4 страны в мире (по словам яндекса) имеют свои качественные поисковые системы. Мы попытаемся разобраться почему так происходит, в чем тут сложность и как написать свою поисковую машину. Да так, чтобы она ещё и работала.

В ходе работы мастерской мы изучим простейшие алгоритмы и структуры данных, используемые в задачах информационного поиска (на одном компьютере, в оперативной памяти): напишем обратные индексы, реализуем булев поиск (boolean retrieval), узнаем, как

исправлять опечатки в запросах. Довольно большую часть времени мы посвятим изучению языка C, системе Linux и манерам хорошего программирования.

Требуемый уровень: знание Pascal, C или другого языка этого класса, >= 8 класс.

Инструментарий: gcc, vim, make, svn.

9. Мастерская "Приключения в лабиринте"

Мастер: Илья Насибулов

Подмастерье: Глеб Ряскин

Будем изучать C, линукс, основы OpenGL. Как много узнаем - зависит от участников мастерской. В итоге напишем лабиринт, используя OpenGL, в котором и будет гулять наш главный герой. Будет он искать что-либо в нем, убегать от монстров или даже бороться с ними - зависит от нашей фантазии и возможностей. Ждем всех, но без желания работать не берем :)

Инструментарий: vim, gcc.

10. Мастерская "Конструктор интерьера"

Мастер: Виктор Алексеевич Сакерин

Программа позволяет создавать план комнаты (вид сверху), размещать в ней мебель. Предметы мебели выполнены векторно, чтобы их было возможно масштабировать и поворачивать (как и всё помещение в целом). Когда минимальный уровень будет достигнут, начнут добавляться надстройки: 1) отдельная программа для создания мебели; 2) возможность редактирования не одной комнаты, а всей квартиры; 3) расчёт площади, занимаемой мебелью и свободного пространства. Может и ещё придумаем, хватило бы сил и времени.

Участники не моложе 6 класса, не боятся математики. Будет использоваться Pascal. Если большая часть участников его не знает, то учебная цель мастерской будет его изучение. Если все участники знают Pascal, будем изучать Delphi и реализовывать проект на нём.

11. Мастерская "Вероятностные Графические Модели"

Мастер: Карасюк Павел

Подмастерье: Колобов Фёдор

Случайность и неопределенность - неотъемлемые части жизни любого человека. Сколько времени займет поездка? Какое лечение лучше подействует на пациента? Каким был урожай в Зимбабве в прошлом году? Для описания систем случайных взаимосвязанных величин существует аппарат: вероятностные графические модели. В нашей мастерской мы изучим его и реализуем связанные с ним алгоритмы. На этой основе мы напишем ряд демострационных задач.

Требования: 8-9 класс

Инструментарий: Octave

12. Мастерская «Изучаем C#»

Мастер: Александр Гурьевич Марчук

C# является одним из лучших языков объектно-ориентированного программирования. В принципе, начать работать на C# просто и доступно для всех, имеющих хотя бы начальные познания в программировании. Более подготовленные ученики смогут увидеть и ощутить изящество и эффективность разных конструкций и слоев как языка, так и библиотеки . NET Framework . Параллельно с учебным циклом, учащиеся смогут сосредоточиться на творческом проекте, предположительно Web-приложении интерфейса к базе данных и документов ЛШЮП.

Работа будет вестись в ОС Windows, платформа .NET , инструментальная система Visual Studio . Будут также даны элементы знаний по HTML , XML , CSS , JavaScript , клиент-серверным технологиям.

Требования к ученикам: опыт работы на Паскале или Си или хотя бы Лого.

13. Мастерская "Онлайн-хранилище"

Мастер: Игорь Дульцев

Мы напишем онлайн-хранилище картинок с каталогизацией на основе меток. Пользователь сможет добавлять картинки, обозначать их тегами по различным критериям (разрешение, тематика, персоналии), объединять картинки в серии,оценивать и комментировать их.

По возможности организуем ряд прочих полезных дополнений, например, дедупликацию.

В рамках работы в мастерской учащиеся познакомятся с основами веб-разработки.

Мы будем использовать node.js, jQuery, bootstrap, git, bash, imagemagick.

Для разработки мы будем использовать разные редакторы, в том числе vim.

14. Мастерская «Симулятор эволюции»

Мастер: Никита Дрёмов

Вы бы хотели проследить, как развивается жизнь? Влиять на ход ее течения? Именно этим мы и будем заниматься: создавать микромир. Нами будут прописаны сами существа, среда обитания, учитывая разнообразные условия (которые по возможности будут добавляться). А также мы пропишем модель их изменчивости и посмотрим на их поведения при различных ситуациях. Мы сможем помочь им развиться или же наоборот поставить им сложные условия выживания.

Симулятор будет написан на C#.

Ребята от 7 класса или очень жаждущие.

Инструментарий: Visual Studio 2012

15. Мастерская "Flash-студия"

Мастер: Светлана Николаевна Коваль

Создать мультфильм, анимационную игру, деловую презентацию или привлекательный сайт? Нет ничего проще, если вы владеете технологией Flash. Все эти возможности мы попытаемся освоить в ходе работы мастерской. Приглашаются все, кто интересуется web-дизайном и анимацией, пробовал рисовать с помощью карандаша на бумаге или любого графического редактора на компьютере и это занятие не вызывает отвращение. Обязательно будем программировать!

Познакомимся с основами ActionScript