Bull. Nov. Comp. Center, Comp. Science, 26 (2007), 1–17

Ontological transition systems

Igor S. Anureev

Abstract. A new class of transition systems, ontological transition systems is presented. They enrich transition systems with ontological entities. In the framework of development of a language of ontological transition systems OTSL, a sublanguage of formulas is defined. Formulas are used to specify ontological entities of ontological transition systems.

1. Introduction

The state transition systems are a well-known formalism for description of operational semantics of programming lanquages and program models. A common way to rigorously define the operational semantics, pioneered by Gordon Plotkin in his paper "A Structural Approach to Operational Semantics" [1], is to provide a state transition system for the language of interest.

A state transition system is defined as an abstract machine which consists of a set of states and transitions between states. On the one hand, simplicity of definition of these systems makes them a universal formalism for description of the behaviour of systems of different nature (algorithms, programs, program models, computer systems, and so on). On the other hand, it leads to a loss of the conceptual structure of systems in their descriptions.

A natural question is how to enrich the states or/and transitions of transition systems to make these systems more conceptually capacious, having preserved their generality.

A logical-algebraic approach to solution of this problem was suggested by Yuri Gurevich, based around the concept of an abstract state machine [2, 3]. Abstract state machines (ASMs), formerly known as evolving algebras, are a special kind of transition systems. The states of ASMs can be arbitrary algebras. The choice of an appropriate algebra signature allows us to adapt ASMs to problem domains. The ASM approach has already proven to be suitable for large-scale specifications of realistic programming languages [4, 5, 6, 7, 8, 9]. Other applications of ASMs to various domains can be found in [10].

The ASM theory is the basis for Abstract State Machine Language [11] developed by Microsoft and XASM (Anlauff's eXtensible ASMs) [12], an open source implementation.

We suggest the ontological approach to solution of this problem, based around the concept of an ontological transition system. Ontological transition systems (OTSs) are a special kind of transition systems. An OTS can be regarded as a transition system which has the following properties:

- There is a conceptual structure (a sets of concepts and a set of relations) which is common for all states of the transition system.
- There is a function of retrieving the content of this conceptual structure from the states of the transition system.

Formally, an ontological transition system consists of a set of objects, a transition system, an ontology and a function, called content retrieval, which defines the content of concepts and relations for each state of the transition system.

On the basis of OTSs, the ontological transition system language OTSL has been developed. It includes two sublanguages: a language of actions and a language of formulas. Actions specify the transitions of OTSs. Formulas specify the ontological entities of OTSs. In this paper, the language of formulas is presented. A description of the language of actions can be found in [13].

The paper has the following structure. Section 2 presents preliminary notions and denotations used in this paper. Section 3 defines the ontological transition systems and related entities. Section 4 sketches out the main notions of the OTSL language. The base constructs of the language of formulas such as terms, concept expressions and formulas are presented in Sections 5, 6, and 7, respectively. Section 8 presents additional constructs which can be used in formulas. On the one hand, these constructs are reducible to the base formula constructs. On the other hand, they enlarge a conceptual capacity of the language of formulas. Sections 9 and 10 define concept declarations and relation declarations, respectively. They are used to specify ontological entities of OTSs (concepts, relations, the content of concepts and relations).

This research is partially supported by the grant 06-01-00464a from RFBR, and by the integration grant 14 and the grant for young scientists from SB RAS.

2. Preliminaries

This section presents preliminary notions and denotations used in this paper.
Set-theoretical Denotations. Set-theoretical denotations used in this paper are the following:

- $\emptyset$ denotes the empty set;
- $X \in Y$ denotes that an element X belongs to a set Y;

- $X \cup Y$, $X \cap Y$, and $X \setminus Y$ denote a union, intersection and difference of sets X and Y, respectively;
- $X \times Y$ denotes the Cartesian product of sets X and Y;
- 2^X denotes the set of all subsets of a set X;
- $X \to Y$ denotes the set of all total functions from X to Y;
- $X \to Y \oplus X' \to Y'$, where $X' \cap Y' = \emptyset$, denotes the set of all total functions which act from X to Y and from X' to Y'.

Logical Denotations. Logical denotations used in this paper are the following:

- *bool* denotes the set $bool = \{true, false\}$;
- $\forall X \in Y(A)$ denotes that for all $X \in Y$ the property A is true;
- $\exists X \in Y(A)$ denotes that there is $X \in Y$ such that the property A is true;
- $\nexists X \in Y(A)$ denotes that there is no $X \in Y$ such that the property A is true;
- $*[X_1 \in Y_1, ..., X_n \in Y_n](A)$, where $* \in \{\forall, \exists, \nexists\}$, denotes $*X_1 \in Y_1 ... *X_n \in Y_n(A)$;
- $\neg A$, $A \wedge B$, and $A \vee B$ denote negation of the property A and conjunction and disjunction of the properties A and B, respectively.

Entities. In this paper, an entity is often defined by the name of a set which contains all instances of the entity. This name is also used as a nonterminal in grammar rules. Under the agreement, it starts with a small letter. The name of an element of the set coincides with the name of the set (possibly with additional indexes and strokes) except for the first letter which is capitalized. For example, let *ob* be a set of objects (it defines the entity "object"). Then Ob, Ob' and Ob_2 are objects. In grammar rules the following denotations are used: ::= means "has the form", | separates alternatives and means "or".

Positions and Substitutions. A number of entities in this paper are defined by grammar rules. These entities can be represented in the form of labeled trees. The notions of position and substitution are defined for this representation. The record $T[L_1 \leftarrow T_1, ..., L_n \leftarrow T_n]$ denotes a tree which is obtained from the tree T by replacement of all occurences of leaves with the labels L_1, ..., L_n by the trees T_1, ..., T_n, respectively. The record T_q denotes a subtree of the tree T in the position q. Let $pos(T)$ be the set of all positions in the tree T. The relation $\prec$ is defined on the set $pos(T)$. The record $q \prec q'$ means that A_q is a subtree of $A_{q'}$. Let $cPos(T)$ be the set of all positions of childrens of the tree T. The relation $\prec_c$ is defined on the set $cPos(T)$. The record $q \prec_c q'$ means that A_q is to the left from $A_{q'}$. The record $T[T']_q$ denotes a tree which is obtained from the tree T by replacement of a subtree in the position q by the tree T'.

3. Ontological Transition Systems

This section defines ontological transition systems and related entities.
Transition Systems. Transition systems can be labelled or unlabelled.
Unlabelled Transition Systems. An unlabelled transition system tS is defined as a pair (st, tr). The set st is called a set of states. The function

$$tr \in st \times st \to bool$$

is called a transition relation. The property $tr(St, St')$ means that there is a transition from the state St to the state St'.
Labelled Transition Systems. A labelled transition system tS is defined as a triple (l, st, tr). The set l is called a set of labels. The set st is called a set of states. The function

$$tr \in st \times st \to (l \to bool)$$

is called a transition relation. The property $tr(St, St)(L)$ means that there is a transition from the state St to the state St' with the label L.
Ontologies. An ontology of a system describes its conceptual structure. It consists of a set of concepts and a set of relations. Concepts define the kinds of sequences of objects of the system. In particular, they define the kinds of objects of the system. Relations define the kinds of interrelations between objects.

Formally, an ontology ont is defined as a pair (co, re). The set co is called a set of concepts of the ontology ont. The set re is called a set of relations of the ontology ont.
Ontological Transition Systems. An ontological transition system (OTS for short) consists of a set of objects, a transition system, an ontology and a function, called content retrieval, which define the content of concepts and the content of relations for each state of the transition system. The content of a concept is defined as a subset of the set of sequences of objects. The content of a relation is defined as a binary relation on sequences of objects.

Formally, an OTS is defined as a quadruple $(ob, tr, ont, cont)$. The set ob is called a set of objects. Let se be a set of sequences of objects. The set st of states of the OTS is defined as follows:

$$st = ob \to se.$$

The function

$$cont \in co \times st \to 2^{se} \oplus re \times st \to 2^{se \times se}$$

is called a content retrieval. The set $cont(Co, St)$ is called a content of the concept Co in the state St. The set $cont(Re, St)$ is called a content of the relation Re in the state St. The sequence $St(Ob)$ is called a content of the object Ob in the state St. The content of an object defines its structure and objects which interrelate with it. The content of objects is used to retrieve information about the content of concepts and relations.

4. Introduction to OTSL

This section sketches out the main notions of the ontological transition system language OTSL.

Keywords. The set kW of keywords is built in the following way:

$$\begin{array}{l} kW ::= \,?\ |\ !\ |\ \#...\ |\ =\ |\ \sim\ |\ :\ |\ ;\ |\ @\ |\ [\ |\ ]\ |\ \{\ |\ \} \\ \quad |\ (\ |\ )\ |\ ()\ |\ and\ |\ or\ |\ not\ |\ implies\ |\ iff\ |\ :=\ |\ +=, \end{array}$$

where #... is any sequence of letters and digits from the set

$$\{a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$$

starting with # except for $\#i$ and $\#o$. Keywords are used in constructs of OTSL.

Objects. The set ob of objects is used to present the objects of OTSs. It is an arbitrary set such that $ob \cap kW = \emptyset$.

Special Objects. The set $sOb \subseteq ob$ of special objects is built in the following way:

$$sOb ::= true\ |\ new\ |\ val\ |\ \#i\ |\ \#o\ |\ o\ |\ s\ |\ e\ |\ ns\ |\ eo.$$

Special objects represent the specific-purpose objects which are common for all OTSs.

Sequences. The sets se and nSe of sequences and nonempty sequences, respectively, are built in the following way:

$$\begin{array}{l} se ::= ()\ |\ nSe, \\ nSe ::= ob\ |\ ob\ nSe. \end{array}$$

Let us note that, by definition, $ob \subseteq se \wedge ob \subseteq nSe$. The sequence () is called then empty sequence.

Concatenation. The concatenation function $con \in se \times se \to se$ is defined as follows:

$$con(NSe, NSe') = NSe\ NSe' \wedge con((), Se) = con(Se, ()) = Se.$$

Equalities. The equality relation $=$ on sequences is defined as follows: $Se = Se'$ is true, if Se is equal to Se', otherwise, it is false.
Weak Equalities. The weak equality relation $\sim$ on sequences is defined as follows: $Se \sim Se'$ is true, if the sequence Se is a permutation of the sequence Se', otherwise, it is false.
Concepts and Relations. Concepts and relations are objects:

$$co \subseteq ob \wedge re \subseteq ob.$$

Basic concepts. There are five basic concepts in OTSL: o, s, e, ns, and eo. Their content does not depend on a state:

$$\begin{aligned} \forall St(&cont(o, St) = ob \wedge \\ &cont(s, St) = se \wedge \\ &cont(e, St) = \{()\} \wedge \\ &cont(ns, St) = nSe \wedge \\ &cont(eo, St) = ob \cup \{()\}). \end{aligned}$$

Actions. The set act of actions is defined in paper [13] which describes the sublanguage of actions of OTSL. Actions are used as labels in transition relations.
OTS declarations. OTS declarations are used to specify OTSs. The sets $otsDec$ and $otsDecMem$ of OTS declarations and OTS declaration members, respectively, are built in the following way:

$$otsDec ::= otsDecMem \mid otsDecMem\ otsDec.$$

$$otsDecMem ::= coDec \mid reDec \mid trDec.$$

The sets $coDec$, and $reDec$ of concept declarations and relation declarations, respectively, are defined in sections 9 and 10 below. The definition of the set $trDec$ of transition declarations can be found in the description of the language of actions [13].

5. Terms

This section defines terms and the related entities.
Terms. The set te of terms is built in the following way:

$$te ::= eSe \mid ob \mid obC \mid teCom.$$

The object content obC and term composition $teCom$ are defined below.
Term evaluation. The function $val \in st \to (te \to se)$ is called a term evaluation. This function defines the semantics of terms. The sequence $val(St)(Te)$ is called the value of the term Te in the state St.
The Empty Sequence. The value of the empty sequence is the empty sequence itself:

$$val(St)(()) = ().$$

Objects. The value of an object is the object itself:

$$val(St)(Ob) = Ob.$$

The Object Content. The object content is defined as follows:

$$obC ::=?\ ob.$$

The value of the object content $?Ob$ in a state St is the content of the object Ob in the state St:

$$val(St)(?Ob) = St(Ob).$$

Term Composition. The term Composition $teCom$ is defined as follows:

$$teCom ::= te\ te.$$

The value of a composition of terms is concatenation of the values of the terms:

$$val(St)(Te\ Te') = con(val(St)(Te), val(St)(Te')).$$

6. Concept expressions

This section defines concept expressions and the related entities.

Concept expressions. The set $coExp$ of concept expressions is built in the following way:

$$coExp ::= co \mid im \mid preIm,$$

where the image im and preimage $preIm$ are defined below.

Concept Expression evaluation. The function $val \in st \to (coExp \to 2^{se})$ is called a concept expression evaluation. This function defines the semantics of concept expressions. The sequence $val(St)(CoExp)$ is called the value of the concept expression $CoExp$ in the state St.

Concepts. The value $val(St)(Co)$ of the concept Co in the state St is defined as the content of the concept Co:

$$val(St)(Co) = cont(Co, St).$$

Images. The set im of images is defined as follows:

$$im ::= re < te.$$

The value of an image $Re < Te$ in a state St is the image of the content of the relation Re for the set $\{val(St)(Te)\}$:

$$val(St)(Re < Te) = \{Se \mid (val(St)(Te), Se) \in cont(Re)\}.$$

Preimages. The set $preIm$ of preimages is defined as follows:

$$preIm ::= re > te.$$

The value of a preimage $Re > Te$ in a state St is the preimage of the content of the relation Re for the set $\{val(St)(Te)\}$:

$$val(St)(Re > Te) = \{Se \mid (Se, val(St)(Te)) \in cont(Re)\}.$$

7. Formulas

This section defines formulas and the related entities.
Formulas. The set fo of formulas is built in the following way:

$$fo ::= aFo \mid pFo \mid qFo \mid dFo \mid bFo.$$

The sets aFo, pFo, qFo, dFo, and bFo of atomic formulas, propositional formulas, quantified formulas, dynamic formulas, and bracketed formulas, respectively, are defined below.
Formula evaluation. The function $val \in st \to (fo \to se)$ is called a formula evaluation. This function defines the semantics of formulas. The sequence $val(St)(Fo)$ is called the value of the formula Fo in the state St. A formula Fo is true in a state St, if $val(St)(Fo) \neq ()$. Otherwise, the formula Fo is false in the state St.
Atomic Formulas. The set aFo of atomic formulas is built in the following way:

$$aFo ::= te \mid mem \mid eq \mid wEq.$$

The membership mem, equality eq and weak equality wEq are defined below.
Memberships. The set mem of memberships is built in the following way:

$$mem ::= te : coExp.$$

A membership $Te : CoExp$ is true in a state St, if the value of the term Te belongs to the value of the concept expression $CoExp$:

$$val(St)(Te : CoExp) = \begin{cases} true, & \text{if } val(St)(Te) \in val(St)(CoExp); \\ (), & \text{otherwise.} \end{cases}$$

Equalities. The set eq of equalities is built in the following way:

$$eq ::= te = te.$$

An equality $Te = Te'$ is true in a state St, if the values of terms Te and Te' in the state St are equal:

$$val(St)(Te = Te') = \begin{cases} true, & \text{if } val(St)(Te) = val(St)(Te'); \\ (), & \text{otherwise.} \end{cases}$$

Weak Equalities. The set wEq of weak equalities is built in the following way:

$$eq ::= te \sim te.$$

An equality $Te \sim Te'$ is true in a state St, if the values of terms Te and Te' in the state St are weakly equal:

$$val(St)(Te \sim Te') = \begin{cases} true, & \text{if } val(St)(Te) \sim val(St)(Te'); \\ (), & \text{otherwise.} \end{cases}$$

Propositional Formulas. The set pFo of propositional formulas is built with the help of logical connectives: negation (not), conjunction (and), disjunction (or), implication ($implies$), and equivalence (iff)

$$pFo ::= not\ fo \mid fo\ and\ fo \mid fo\ or\ fo \mid fo\ implies\ fo \mid fo\ iff\ fo.$$

with their usual semantics:

$$val(St)(not\ Fo) = \begin{cases} true, & \text{if } val(St)(Fo) = (); \\ (), & \text{otherwise,} \end{cases}$$

$$val(St)(Fo\ and\ Fo') = \begin{cases} true, & \text{if } val(St)(Fo) \neq () \wedge val(St)(Fo') \neq (); \\ (), & \text{otherwise,} \end{cases}$$

$$val(St)(Fo\ or\ Fo') = \begin{cases} true, & \text{if } val(St)(Fo) \neq () \vee val(St)(Fo') \neq (); \\ (), & \text{otherwise,} \end{cases}$$

$$val(St)(Fo\ implies\ Fo') = \begin{cases} true, & \text{if } val(St)(Fo) = () \vee val(St)(Fo') \neq (); \\ (), & \text{otherwise,} \end{cases}$$

$$val(St)(Fo\ iff\ Fo') = \begin{cases} true, & \text{if } \begin{array}{l} val(St)(Fo) = () \wedge val(St)(Fo') = () \vee \\ val(St)(Fo) \neq () \wedge val(St)(Fo') \neq (); \end{array} \\ (), & \text{otherwise.} \end{cases}$$

Quantified Formulas The set qFo of quantified formulas is built with the help of existential (?) and universal (!) quantifiers

$$qFo ::= (\ ?\ (\ bin\ )\ fo\ ) \mid (\ ?\ (\ bin\ )\ fo\ )$$

$$bin ::= ob\ :\ coExp \mid bin\ bin$$

with their usual semantics:

$$val(St)((?(Ob_1 : CoExp_1\ ...\ Ob_n : CoExp_n)Fo)) =$$
$$\begin{cases} true, & \text{if } \begin{array}{l} \exists (A_1 \in val(St)(CoExp_1), ..., A_n \in val(St)(CoExp_n)) \\ (val(St)(Fo(Ob_1 \leftarrow A_1, ..., Ob_n \leftarrow A_n)) \neq ()); \end{array} \\ (), & \text{otherwise,} \end{cases}$$

$$val(St)((!(Ob_1 : CoExp_1 \ ... \ Ob_n : CoExp_n)Fo)) =$$
$$\begin{cases} true, & \text{if } \begin{array}{l}\forall(A_1 \in val(St)(CoExp_1), ..., A_n \in val(St)(CoExp_n)) \\ \quad (val(St)(Fo(Ob_1 \leftarrow A_1, ..., Ob_n \leftarrow A_n)) \neq ());\end{array} \\ (), & \text{otherwise.} \end{cases}$$

The elements of the set bin are called bindings.

Dynamic Formulas. Dynamic formulas are built with the help of dynamic logic modalities

$$dFo ::= (\ ? \ \{ \ act \ \} \ fo \ ) \mid (\ ! \ \{ \ act \ \} \ fo \ )$$

with the usual semantics:

$$val(St)((?\{Act\}Fo)) =$$
$$\begin{cases} true, & \text{if } \exists St'(tr(St, St')(Act) \wedge val(St')(Fo) \neq ()); \\ (), & \text{otherwise,} \end{cases}$$

$$val(St)((!\{Act\}Fo)) =$$
$$\begin{cases} true, & \text{if } \forall St'(tr(St, St')(Act) \Rightarrow val(St')(Fo) \neq ()); \\ (), & \text{otherwise.} \end{cases}$$

Bracketed Formulas. The set bFo of bracketed formulas is built in the following way:

$$bFo ::= (\ fo \ ).$$

Brackets are used to define the order of computation of subformulas in formulas:

$$val(St)((Fo)) = val(St)(Fo).$$

Operations. The order of computation is specified by priority and associativity of operations. Operations are listed below in the descending order:

$$= \ \sim \ \ not \ and \ or \ implies \ iff.$$

Example. The formula $A \ or \ B \ and \ C = D$ is equivalent to the formula

$$A \ or \ (B \ and \ (C = D)).$$

In addition, the operations *and* and *or* are left associative.

Example. The formula $A \ and \ B \ and \ C$ is equivalent to the formula

$$(A \ and \ B) \ and \ C.$$

8. Additional formula constructs

This section presents additional constructs which can be used in formulas. On the one hand, these constructs are reducible to the basic formula constructs. On the other hand, they enlarge the conceptual capacity of the OTSL language. These constructs include anonymous objects and anonymous sequences. They are used in place of terms. To introduce them, the set te of terms is redefined.

Redefining Terms. The set te of terms is built in the following way:

$$te ::= eSe \mid ob \mid obC \mid teCom \mid anOb \mid anSe.$$

The sets $anOb$ and $anSe$ of anonymous objects and anonymous sequences, respectively, are defined below.

Anonymous Objects. The set $anOb$ of anonymous objects is built in the following way:

$$anOb ::= (\ =\ \ te\ ) \mid (\ te\ ) \mid (\ \sim\ \ te\ ).$$

An anonymous object $(= Te)$ represents any object Ob with the content equal to the value of the term Te in the state St, i.e. $St(Ob) = val(St)(Te)$. An anonymous object (Te) is a synonym for $(= Te)$.

An anonymous object $(\sim Te)$ represents any object Ob with the content weakly equal to the value of the term Te in the state St, i.e.

$$St(Ob) \sim val(St)(Te).$$

An anonymous object $(* \, Te)$ is implicitly bound by the existential quantifier ?, i.e. any formula Fo such that $Fo_q = (* \, Te)$ is equivalent to the formula

$$(?(Ob : o)(Fo[Ob]_q \ and \ ?Ob \ * \ Te)),$$

where $* \in \{=, \sim\}$. This requirement guarantees existence of at least one object which satisfies the formula Fo and has the content defined by the term Te.

Anonymous Sequences. The set $anSe$ of anonymous sequences is built in the following way:

$$anSe ::= * : coExp.$$

An anonymous sequence $*{:}CoExp$ represents any sequence Se such that

$$Se \in val(St)(CoExp).$$

An anonymous sequence Se is implicitly bound by the existential quantifier ?, i.e. any formula Fo such that $Fo_q = *{:}CoExp$ is equivalent to the formula $(?(Ob{:}CoExp)Fo[Ob]_q)$. This requirement guarantees existence of at least one sequence $Se \in val(St)(CoExp)$ which satisfies the formula Fo.

Elimination of anonymous objects and sequences. Anonymous objects and anonymous sequences can be reducible to other constructs. Their semantics is defined by the reduction function $red \in aFo \to qFo$. This function normalizes atomic formulas, eliminating anonymous objects and anonymous sequences:

$$red(Fo) = \begin{cases} (?(Ob\!:\!o)\ red(?Ob = Te) \wedge red(Fo[Ob]_q)), & \text{if } Fo_q = (= Te); \\ (?(Ob\!:\!o)\ red(?Ob = Te) \wedge red(Fo[Ob]_q)), & \text{if } Fo_q = (Te); \\ (?(Ob\!:\!o)\ red(?Ob \sim Te) \wedge red(Fo[Ob]_q)), & \text{if } Fo_q = (\sim Te); \\ (?(Ob\!:\!CoExp)\ red(Fo[Ob]_q)), & \text{if } Fo_q = *\!:\!CoExp; \\ Fo, & \text{otherwise.} \end{cases}$$

Example. Let an OTS specify a referral database. The formula

$$(* : s\ surname\ Ivanov\ * : s)$$

is equivalent to the formula

$$(?(X : o\ Y : s\ Z : s)\ ?X = Y\ surname\ Ivanov\ Z).$$

It means that there exists at least one object with the surname *Ivanov* in the state of the referral database.
Example. Let an OTS specify a database of vacancies. The formula

$$* : employer$$

is equivalent to the formula

$$(?(X : o)\ X : employer).$$

It means that there exists at least one employer in the state of the database of vacancies.

9. Concept Declarations

This section presents concept declarations which are used to define concepts of OTSs, as well as the content of concepts.
Concept Declarations. The set *coDec* of concept declarations is built in the following way:

$$coDec ::= \#c\ ob\ \{\ fo\ \}.$$

Let *CoDec* be a concept declaration of the form $\#c\ Ob\ \{Fo\}$. The object *Ob* is called a concept declared in the concept declaration *CoDec*. The formula *Fo* is called a content declarator of the concept *Ob*.
Concepts in OTS declarations. The set $co(OtsDec)$ is called a set of concepts declared in the OTS declaration *OtsDec*, if

$co(OtsDec) =$
$\{Co \in ob \mid \exists[CoDec \in OtsDec, Fo](CoDec = \#c\ Co\ \{Fo\})\}.$

Thus, $co(OtsDec)$ is the set of concepts declared in concept declarations which are members of $OtsDec$.
The Content of Concepts in OTS declarations. The function

$$cont \in co(OtsDec) \times St \to 2^{ob}$$

is called a concept content retrieval declared by the OTS declaration $OtsDec$, if

$$cont(Co, St) = \left\{ Ob \mid \begin{array}{c} \exists[CoDec \in OtsDec, Fo] \\ (CoDec = \#c\ Co\ \{Fo\} \wedge \\ val(St)(Fo(\#i \leftarrow Ob)) \neq ()) \end{array} \right\}.$$

The special object $\#i$ is used in the content declarator Fo of the concept Co to refer to the sequences from the content of the concept Co which is declared in the concept declaration $CoDec$.
Example. The declaration

$$\#c\ emptyConcept\ \{()\}$$

defines the concept $emptyConcept$. Its content is the empty set in any state.
Example. The declaration

$$\#c\ object\ \{\# : o\}$$

defines the concept $object$. The content of the concept $object$ is the set of all objects in any state.
Example. The declaration

$$\#c\ document\ \{\#i{:}o\ and\ ?documents \sim \#i\ *{:}s\}$$

defines the concept $document$. Instances of the concept $document$ are defined as objects from the content of the object $documents$. For example, if the state St is defined by Table 1, then $cont(document, St) = \{A, B\}$.

Table 1

Object	**Content**
documents	*A B*
A	*B*
B	*A*

Example. The declaration

$$\#c\ document\ \{\#i\!:\!o\ and\ ?\#i \sim document{*}\!:\!s\}$$

defines the concept *document*. Instances of the concept *document* are defined as objects such that their content includes the object *document*. For example, if the state *St* is defined by Table 2, then $cont(document, St) = \{A, B\}$.

Table 2

Instance	**Content**
A	*document B*
B	*C document A*
C	*B*

Example. The declaration

$$\#c\ document\ \{\#i\!:\!o\ and\ ?\#i = document\ {*}\!:\!s\}$$

defines the concept *document*. Instances of the concept *document* are defined as objects such that their content includes the object *document* as the first element. For example, if the state *St* is defined by Table 2, then $cont(document, St) = \{A\}$.
Example. The declaration

$$\#c\ makeReport\ \{\#i = makeReport\}$$

defines the concept *makeReport*. The content of the concept *makeReport* consists of one object *makeReport* for any state. Concepts of this form are used to represent procedures. Arguments of these procedures are specified by the content of the concepts for each state.

10. Relation declarations

This section presents relation declarations which are used to define relations of OTSs, as well as the content of relations.
Relation Declarations. The set *reDec* of concept declarations is built in the following way:

$$reDec ::= \#r\ ob\ \{\ fo\ \}.$$

Let *ReDec* be a relation declaration of the form $\#r\ Ob\ \{Fo\}$. The object *Ob* is called a relation declared in the relation declaration *ReDec*. The formula *Fo* is called a content declarator of the relation *Ob*.
Relations in OTS declarations. The set $re(OtsDec)$ is called a set of relations declared in the OTS declaration *OtsDec*, if

$$re(OtsDec) =$$
$$\{Re \in ob \mid \exists[ReDec \in OtsDec, Fo](ReDec = \#c\ Re\ \{Fo\})\}.$$

Thus, $re(OtsDec)$ is the set of relations declared in the relation declarations which are members of $OtsDec$.

The Content of Relations in OTS declarations. The function

$$cont \in re(OtsDec) \times St \to 2^{ob}$$

is called a relation content retrieval declared by the OTS declaration $OtsDec$, if

$$cont(Re, St) = \left\{ (Ob, Ob') \mid \begin{array}{c} \exists[ReDec \in OtsDec, Fo] \\ (ReDec = \#c\ Re\ \{Fo\} \wedge \\ val(St)(Fo(\#i \leftarrow Ob, \#o \leftarrow Ob')) \neq ()) \end{array} \right\}.$$

The special objects $\#i$ and $\#o$ are used in the content declarator Fo of the relation Re to refer to the first and the second components of pairs from the content of the relation Ro which is declared in the relation declaration $ReDec$.

Example. The declaration

$$\#r\ synonym\ \{\#i{:}word\ and\ \#o{:}word\ and\ synonymGroup \sim \#i\ \#o\ *{:}s\}$$

defines the relation *synonym* on words. According to this declaration, two words are synonyms, if they are both included in the content of the concept *synonymGroup*.

Example. The declaration

$$\#r\ title\ \{\#i{:}document\ and\ \#o{:}text\ and\ ?\#i = source\ *:o\ title\ \#o\ *:\ s\}$$

defines the relation *title*. According to this declaration, the text $\#o$ is the title of the document $\#i$ in the state St, if *source B title* $\#o$ is a prefix of the content of $\#i$ for some object B.

This form of representation of the relation content is used for concepts with a fixed order of attributes. In this example, *source* is the first attribute with the value B, *title* is the second attribute with the value $\#i$, the rest of attributes represented by C follows *title*.

Example. The declaration

$$\#r\ title\ \{\#i{:}document\ and\ \#o{:}text\ and\ ?\#i \sim (=title\ \#o)\ *:\ s\}$$

defines the relation *title*. According to this declaration, the text $\#o$ is the title of the document $\#i$ in the state St, if there is an object B such that $B \in St(\#i)$ and $St(B) = title\ \#o$.

Example. The declaration

$\#r$ *expression* $\{\#i\!:\!expressionStatement$ *and* $\#o\!:\!expression$ *and* $?\#i = \#o\ ;\}$

defines the relation *expression*. According to this declaration, the expression $\#o$ is an expression of the expression statement $\#i$ in the state St, if the content of $\#i$ in the state St has the form $\#o$;.

11. Conclusion

A new general-purpose method of formal specifications of computer systems is presented. Based on a new notion of ontological transition systems, it includes:

- definitions of OTSs and related notions;
- a language of formulas which are used to specify ontological entities in OTSs;
- a formal operational semantics of the language of formulas.

The advantages of the method are as follows:

- use of natural intuitive terminology in specifications;
- a language support of description of OTSs;
- integration of the ontological and operational approaches to computer systems specification.

References

[1] Gordon D. Plotkin. A Structural Approach to Operational Semantics. — Aarhus, Denmark, 1981. — (Tech. Rep. / Computer Science Department, Aarhus University; DAIMI FN-19).

[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] Stärk R., Börger E. Java and the Java Virtual Machine: Definition, Verification, Validation. — Springer-Verlag, 2001.

[5] Börger E., Fruja N., Gervasi V., Stärk R. A High-Level Modular Definition of C# // Theor. Comput. Sci. — 2005. — Vol. 336, N 2-3. — P. 235–284.

[6] ITU-T Recommendation Z.100 Annex F: SDL Formal Semantics Definition. — Geneva: International Telecommunications Union (ITU), 2000.

[7] Gurevich Y., Huggins J. The Semantics of the C Programming Language // Lect. Notes Comput. Sci. — 1993. — Vol. 702. — P. 274–309.

[8] Börger E., Rosenzweig D. A Mathematical Definition of Full Prolog // Science of Computer Programming. — 1994. — Vol. 24. — P. 249–286.

[9] Kutter P., Pierantonio A. The Formal Specification of Oberon // J. of Universal Computer Science. — 1997. — N 3 (5). — P. 443–503.

[10] Huggins J. Abstract State Machines Web Page. — http://www.eecs.umich.edu/gasm.

[11] AsmL: The Abstract State Machine Language. Reference Manual. 2002. — http://research.microsoft.com/fse/asml/doc/AsmL2_Reference.doc.

[12] XasM — An Extensible, Component-Based Abstract State Machines Language. — http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html.

[13] Anureev I.S. A Language of Actions in Ontological Transition Systems // Bull. Novosibirsk Comp. Center. Ser. Computer Science. — 2007. — IIS Special Iss. 26. — P. 19–37.