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 sub-language 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 languages 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 (Anlauf’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 set 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 $X$;
• $X \rightarrow Y$ denotes the set of all total functions from $X$ to $Y$;
• $X \rightarrow Y \oplus X' \rightarrow Y'$, 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:

• $\text{bool}$ denotes the set $\text{bool} = \{true, false\}$;
• $\forall X \in Y(A)$ 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;
• $\exists X \in Y(A)$ denotes that there is no $X \in Y$ such that the property $A$ is true;
• $\forall[X_1 \in Y_1, ..., X_n \in Y_n](A)$, where $\ast \in \{\forall, \exists, \exists^c\}$, denotes $\ast X_1 \in Y_1$...
• $\neg A, A \land B,$ and $A \lor B$ 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 occurrences 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 $\text{pos}(T)$ be the set of all positions in the tree $T$. The relation $\prec$ is defined on the set $\text{pos}(T)$. The record $q \prec q'$ means that $A_q$ is a subtree of $A_{q'}$. Let $\text{cPos}(T)$ be the set of all positions of childrens of the tree $T$. The relation $\prec_c$ is defined
on the set $c\text{Pos}(T)$. 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

$$tr \in st \times st \rightarrow \text{bool}$$

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

$$tr \in st \times st \rightarrow (l \rightarrow \text{bool})$$

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:

$$st = ob \rightarrow se.$$
is called a content retrieval. The set \( \text{cont}(Co, St) \) is called a content of the concept \( Co \) in the state \( St \). The set \( \text{cont}(Re, St) \) 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:

\[
kW := ? | ! | \#... | = | \sim | : | @ | ] | ] | \{ | \}
\]

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:

\[
se := () | nSe,
\]
\[
nSe := ob | ob nSe.
\]

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

**Concatenation.** The concatenation function \( con \in se \times se \rightarrow se \) is defined as follows:
\( \text{con}(NSe, NSe') = NSe NSe' \land \text{con}(), Se) = \text{con}(Se, ()) = Se. \)

**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:
\[
\text{co} \subseteq \text{ob} \land \text{re} \subseteq \text{ob}.
\]

**Basic concepts.** There are five basic concepts in OTSL: \( o, s, e, ns, \) and \( eo \). Their content does not depend on a state:
\[
\forall St(\text{cont}(o, St) = \text{ob}) \land
\text{cont}(s, St) = \text{se} \land
\text{cont}(e, St) = \{()\} \land
\text{cont}(ns, St) = nSe \land
\text{cont}(eo, St) = \text{ob} \cup \{()\}).
\]

**Actions.** The set \( \text{act} \) 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 \( \text{otsDec} \) and \( \text{otsDecMem} \) of OTS declarations and OTS declaration members, respectively, are built in the following way:
\[
\text{otsDec} ::= \text{otsDecMem} \mid \text{otsDecMem otsDec}.
\]
\[
\text{otsDecMem} ::= \text{coDec} \mid \text{reDec} \mid \text{trDec}.
\]
The sets \( \text{coDec} \), and \( \text{reDec} \) of concept declarations and relation declarations, respectively, are defined in sections 9 and 10 below. The definition of the set \( \text{trDec} \) 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 \( \text{te} \) of terms is built in the following way:
\[
\text{te} ::= eSe \mid \text{ob} \mid \text{obC} \mid \text{teCom}.
\]
The object content \( \text{obC} \) and term composition \( \text{teCom} \) are defined below.

**Term evaluation.** The function \( \text{val} \in \text{st} \rightarrow (\text{te} \rightarrow \text{se}) \) is called a term evaluation. This function defines the semantics of terms. The sequence \( \text{val}(St)(Te) \) 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:
Ontological transition systems

\[ \text{val}(St)((())) = (). \]

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

\[ \text{val}(St)(Ob) = Ob. \]

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

\[ \text{obC} ::= ? \text{ob}. \]

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

\[ \text{val}(St)(?Ob) = St(Ob). \]

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

\[ \text{teCom} ::= \text{te} \text{te}. \]

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

\[ \text{val}(St)(Te \text{Te}') = \text{con}(\text{val}(St)(Te), \text{val}(St)(Te')). \]

6. Concept expressions

This section defines concept expressions and the related entities.

**Concept expressions.** The set \( \text{coExp} \) of concept expressions is built in the following way:

\[ \text{coExp} ::= \text{co} | \text{im} | \text{preIm}, \]

where the image \( \text{im} \) and preimage \( \text{preIm} \) are defined below.

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

**Concepts.** The value \( \text{val}(St)(Co) \) of the concept \( Co \) in the state \( St \) is defined as the content of the concept \( Co \):

\[ \text{val}(St)(Co) = \text{cont}(Co, St). \]

**Images.** The set \( \text{im} \) of images is defined as follows:

\[ \text{im} ::= \text{re} < \text{te}. \]

The value of an image \( \text{Re} < \text{Te} \) in a state \( St \) is the image of the content of the relation \( \text{Re} \) for the set \( \{\text{val}(St)(Te)\} \):
Preimages. The set \( \text{preIm} \) of preimages is defined as follows:

\[
\text{preIm} ::= re > te.
\]

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

\[
\text{val}(St)(Re > Te) = \{ Se | (Se, \text{val}(St)(Te)) \in \text{cont}(Re) \}.
\]

7. Formulas

This section defines formulas and the related entities.

Formulas. The set \( fo \) of formulas is built in the following way:

\[
fo ::= aFo | pFo | qFo | dFo | bFo.
\]

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 \( val \in st \rightarrow (fo \rightarrow se) \) is called a formula evaluation. This function defines the semantics of formulas. The sequence \( \text{val}(St)(Fo) \) is called the value of the formula \( Fo \) in the state \( St \). A formula \( Fo \) is true in a state \( St \), if \( \text{val}(St)(Fo) \neq () \). Otherwise, the formula \( Fo \) is false in the state \( St \).

Atomic Formulas. The set \( aFo \) of atomic formulas is built in the following way:

\[
aFo ::= te | \text{mem} | eq | wEq.
\]

The membership \( \text{mem} \), equality \( eq \) and weak equality \( wEq \) are defined below.

Memberships. The set \( mem \) of memberships is built in the following way:

\[
\text{mem} ::= te : \text{coExp}.
\]

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 \( CoExp \):

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

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{array}{ll} \text{true}, & \text{if } val(St)(Te) = val(St)(Te'); \\ () , & \text{otherwise.} \end{array} \\

**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:

\[ val(St)(Te \sim Te') = \{ \begin{array}{ll} \text{true}, & \text{if } val(St)(Te) \sim val(St)(Te'); \\ () , & \text{otherwise.} \end{array} \]

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

\[ pFo ::= \text{not } fo | fo \text{ and } fo | fo \text{ or } fo | fo \text{ implies } fo | fo \text{ iff } fo. \]

with their usual semantics:

\[ val(St)(\text{not } Fo) = \{ \begin{array}{ll} \text{true}, & \text{if } val(St)(Fo) = (); \\ () , & \text{otherwise,} \end{array} \]

\[ val(St)(Fo \text{ and } Fo') = \{ \begin{array}{ll} \text{true}, & \text{if } val(St)(Fo) \neq () \land val(St)(Fo') \neq (); \\ () , & \text{otherwise,} \end{array} \]

\[ val(St)(Fo \text{ or } Fo') = \{ \begin{array}{ll} \text{true}, & \text{if } val(St)(Fo) \neq () \lor val(St)(Fo') \neq (); \\ () , & \text{otherwise,} \end{array} \]

\[ val(St)(Fo \text{ implies } Fo') = \{ \begin{array}{ll} \text{true}, & \text{if } val(St)(Fo) = () \lor val(St)(Fo') \neq (); \\ () , & \text{otherwise,} \end{array} \]

\[ val(St)(Fo \text{ iff } Fo') = \{ \begin{array}{ll} \text{true}, & \text{if } val(St)(Fo) = () \land val(St)(Fo') \neq (); \\ () , & \text{otherwise.} \end{array} \]

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

\[ qFo ::= (\text{? (bin ) } Fo ) | (\text{? (bin ) } fo ) \]

\[ bin ::= ob : coExp | bin \text{ bin} \]

with their usual semantics:

\[ val(St)((? (Ob_1 : CoExp_1 ... Ob_n : CoExp_n) Fo)) = \{ \begin{array}{ll} \text{true}, & \exists(A_1 \in val(St)(CoExp_1),...A_n \in val(St)(CoExp_n)) \\ & (val(St)(Fo(Ob_1 \leftarrow A_1,...,Ob_n \leftarrow A_n)) \neq () ); \\ () , & \text{otherwise,} \end{array} \]
\[ \text{val}(\text{St})(\forall(A_1 \in \text{val}(\text{St})(\text{CoExp}_1), \ldots, A_n \in \text{val}(\text{St})(\text{CoExp}_n)) \Rightarrow \text{val}(\text{St})(\text{Fo}(\text{Ob}_1 \leftarrow A_1, \ldots, \text{Ob}_n \leftarrow A_n)) \neq ()); \]

\]

The elements of the set \( \text{bin} \) are called bindings.

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

\[
dFo ::= ( \mathbin{?} \{ \text{act} \} f o ) \mid ( \mathbin{!} \{ \text{act} \} f o )
\]

with the usual semantics:

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

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

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

\[
bFo ::= ( f o ).
\]

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

\[
\text{val}(\text{St})(\text{Fo}) = \text{val}(\text{St})(\text{Fo}).
\]

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

\[
= \sim \quad \text{not} \quad \text{and} \quad \text{or} \quad \text{implies} \quad \text{iff}.
\]

**Example.** The formula \( A \lor B \land C = D \) is equivalent to the formula

\[
A \lor (B \land (C = D)).
\]

In addition, the operations \text{and} and \text{or} are left associative.

**Example.** The formula \( A \land B \land C \) is equivalent to the formula

\[
(A \land B) \land C.
\]
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:

$$te := eS|ob|obC|teCom|anOb|anSe.$$  

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:

$$anOb := (=te) | (te) | (∼te).$$

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

$$St(Ob) ∼ val(St)(Te).$$

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 $∗ ∈ \{=,∼\}$. 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

$$Se ∈ val(St)(CoExp).$$

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 $Se ∈ val(St)(CoExp)$ 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 $red \in aFo \rightarrow qFo$. This function normalizes atomic formulas, eliminating anonymous objects and anonymous sequences:

$$red(Fo) =
\begin{cases}
\langle ?(Ob:o) \ red(\langle Ob = Te \rangle) \ \land \ red(\langle Ob[\ q] \rangle) \rangle, & \text{if } F_{oq} = (= Te); \\
\langle ?(Ob:o) \ red(\langle Ob = Te \rangle) \ \land \ red(\langle Ob[\ q] \rangle) \rangle, & \text{if } F_{oq} = (Te); \\
\langle ?(Ob:o) \ red(\langle Ob \sim Te \rangle) \ \land \ red(\langle Ob[\ q] \rangle) \rangle, & \text{if } F_{oq} = (\sim Te); \\
\langle ?(Ob:CoExp) \ red(Fo(Ob[\ q])) \rangle, & \text{if } F_{oq} = * : CoExp; \\
F_{o}, & \text{otherwise}.
\end{cases}$$

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

$$( * \ : \ s \ \text{surname} \ Ivanov \ * \ : \ s )$$

is equivalent to the formula

$$\langle ?(X : o \ Y : s \ Z : s) \ ?X = Y \ \text{surname} \ Ivanov \ Z \rangle.$$ 

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

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

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

Thus, \( \text{co}(OtsDec) \) is the set of concepts declared in concept declarations which are members of \( OtsDec \).

**The Content of Concepts in OTS declarations.** The function

\[
\text{cont} \in \text{co}(OtsDec) \times St \to 2^{\text{ob}}
\]

is called a concept content retrieval declared by the OTS declaration \( OtsDec \), if

\[
\text{cont}(Co, St) = \\{Ob \mid \exists [CoDec \in OtsDec, Fo] (CoDec = \#c Co \{Fo\} \wedge val(St)(Fo(#i \leftarrow Ob)) \neq ())\}.
\]

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

\[
\#c \text{ emptyConcept} \{()\}
\]

defines the concept \emph{emptyConcept}. Its content is the empty set in any state.

**Example.** The declaration

\[
\#c \text{ object} \{\# : o\}
\]

defines the concept \emph{object}. The content of the concept \emph{object} is the set of all objects in any state.

**Example.** The declaration

\[
\#c \text{ document} \{\#i : o \text{ and } ?documents \sim \#i * : s\}
\]

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

**Table 1**

<table>
<thead>
<tr>
<th>Object</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>documents</td>
<td>A B</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>
Example. The declaration

\[ \#c \text{document} \{\#i:o \text{and } ?\#i \sim \text{document}* :s\} \]

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 `\text{cont}(\text{document}, St) = \{A, B\}`.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td><code>document B</code></td>
</tr>
<tr>
<td>B</td>
<td><code>C document A</code></td>
</tr>
<tr>
<td>C</td>
<td><code>B</code></td>
</tr>
</tbody>
</table>

Example. The declaration

\[ \#c \text{document} \{\#i:o \text{and } ?\#i = \text{document} * :s\} \]

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 `\text{cont}(\text{document}, St) = \{A\}`.

Example. The declaration

\[ \#c \text{makeReport} \{\#i = \text{makeReport}\} \]

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:

\[ reDec := \#r \text{ob}\{ fo \}. \]

Let `ReDec` be a relation declaration of the form `\#r \text{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(\text{OtsDec})` is called a set of relations declared in the OTS declaration `\text{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

\[cont \in re(OtsDec) \times St \rightarrow 2^{ob} \]

is called a relation content retrieval declared by the OTS declaration \(OtsDec\), if

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

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

\[#r synonym \{#i:word and #o:word and synonymGroup ~ #i #o :s\} \]

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

\[#r title \{#i:document and #o:text and ?#i ~ (= title #o) :s\} \]

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


