УДК 004.8

Formalisms for conceptual design of information systems*

Anureev I.S. (Institute of Informatics Systems)

A class of information systems considered in this paper is defined as follows: a system belongs to the class if its change can be caused by both its environment and factors inside the system, and there is an information transfer from it to its environment and from its environment to it. Two formalisms (information transition systems and conceptual transition systems) for abstract unified modelling of the artifacts (concept sketches and models) of the conceptual design of information systems of the class, early phase of information systems design process, are proposed. Information transition defines the abstract unified information model for the artifacts, based on such general concepts as state, information query, answer and transition. Conceptual transition systems are a formalism for conceptual modelling of information transition systems. They defines the abstract unified conceptual model for the artifacts. The basic definitions of the theory of conceptual transition systems are given. A language of conceptual transition systems is defined.

Keywords: information system, information transition system, conceptual structure, ontology, ontological element, conceptual, conceptual state, conceptual configuration, conceptual transition system, conceptual information transition model, transition system, CTSL

1. Introduction

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

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

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

model.

An information transition system (ITS for short) is an extension of an information query system (IQS for short) characterized additionally by the exogenous and endogenous transition relations specifying transitions on states. The exogenous transition relation models change of an information system caused by its environment. It associates queries with binary relations on states called transition relations and answers returning by state pairs from these transition relations called transitions. The endogenous transition relation models change of an information system caused by factors inside the system. It is defined as a transition relation with answers returning by transitions of the transition relation.

A wide variety of information systems is modelled by ITSs in the information aspect, including database management systems with transitions initiated by queries, expert systems with transitions initiated by operations with facts and rules, social networks with transitions initiated by actions of users in accordance with certain communications protocols, abstract machines specifying operational semantics of programming languages with transitions initiated by instructions of abstract machines, verification condition generators specifying axiomatic semantics of programming languages with transitions initiated by inference rules and so on.

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

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

ITS.

10. It must model the change of the content of the conceptual structure.
11. It must model the transition relations of the ITS.
12. The model of the exogenous transition relation must be extensible.

As is shown in [3], conceptual configuration systems (CCSs for short) meet the seven requirements in relation to IQSs. Comparison of CCSs with the abstract state machines [4, 5] which partially meet these requirements was made in [3]. In this paper we present an extension of CCSs, conceptual transition systems (CTSs for short) as the formalism satisfying the all above requirements.

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 language CTSL of CTSs is described in section 4. Semantics of executable elements in CTSL is defined in 5. We establish that CTSs meet the above requirements in section 6.

2. Preliminaries

2.1. Sets, sequences, multisets

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

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

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

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

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

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

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

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

2.2. Contexts

The terms used in the paper are context-dependent.

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

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

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

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

2.3. Functions

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

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

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

Let $[f_n\ u_p]$ denote the function $f_{n.1}$ such that $[f_{n.1}\ a_{rg}] = [f_n\ a_{rg}]$ if $a_{rg} \neq a_{rg}[\![u_p]\!]$, and

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

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

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

2.4. Attributes and multi-attributes

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

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

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

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

3. Basic definitions of the theory of conceptual transition systems

Conceptual transition systems (CTSs) are transition systems in which states are conceptual configurations, and transition relations are binary relations on conceptual configurations. In this section the basic definitions of the theory of conceptual transition systems are presented. The defined structures of CTSs are constructed from atoms and, thus, defined implicitly in $[\![A_{tm}]\!]$.

3.1. Information transition systems

Let S_{tt} be a set of objects called states. An element t_{rn} of the form $(s_{tt.1}, s_{tt.2})$ is called a transition. Let T_{rn} be a set of transitions. The states $s_{tt.1}$ and $s_{tt.2}$ are called input and output states in $[\![t_{rn}]\!]$.

Let $S_{s.q}$ be a set of query systems. An object $s_{s.t.i}$ of the form $(s_{s.q}, t_{rn.rlt.ex}, t_{rn.rlt.en})$ is an information transition system if $t_{rn.rlt.ex} \in Q_r \times A_{ns} \times S_{tt} \times S_{tt} \to B_l$, $t_{rn.rlt.en} \in A_{ns} \times S_{tt} \times S_{tt} \to B_l$, and for all $q_r \in Q_r$ there exists $s_{tt} \in S_{tt}$ such that $[value\ q_r\ s_{tt}] \neq und$, or there exist $s_{tt.1} \in S_{tt}$, $s_{tt.2} \in S_{tt}$ and $a_{ns} \in A_{ns}$ such that $[t_{rn.rlt.ex}\ q_r\ a_{ns}\ s_{tt.1}\ s_{tt.2}] = true$. Let $S_{s.t.i}$ be a set of information transition systems.

The system $s_{s.q}$ is called a query system in $[\![s_{s.t.i}]\!]$. The function $t_{rn.rlt.ex}$ is called an exogenous transition relation in $[\![s_{s.t.i}]\!]$. The function $t_{rn.rlt.en}$ is called an endogenous transition relation in $[\![s_{s.t.i}]\!]$. Let $s_{tt.1} \to_{q_r, a_{ns}} s_{tt.2}$ and $s_{tt.1} \to_{a_{ns}} s_{tt.2}$ be shortcuts for $[t_{rn.rlt.ex}\ q_r\ a_{ns}\ s_{tt.1}\ s_{tt.2}] = true$ and $[t_{rn.rlt.en}\ a_{ns}\ s_{tt.1}\ s_{tt.2}] = true$, respectively.

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

A query q_r is an information query in $[\![s_{s.t.i}]\!]$ if $[value\ q_r\ s_{tt}] \neq und$ for some s_{tt}. A query q_r is a change query in $[\![s_{s.t.i}]\!]$ if $[t_{rn.rlt.ex}\ q_r\ a_{ns}\ s_{tt.1}\ s_{tt.2}] = true$ for some $s_{tt.1}$, $s_{tt.2}$ and a_{ns}.

A system $s_{s.t.i}$ executes t_{rn} if $s_{tt.1}[\![t_{rn}]\!] \to_{q_r, a_{ns}} s_{tt.2}[\![t_{rn}]\!]$ for some q_r and a_{ns}, or $s_{tt.1}[\![t_{rn}]\!] \to_{a_{ns}} s_{tt.2}[\![t_{rn}]\!]$ for some a_{ns}. A system $s_{s.t.i}$ transits from $s_{tt.1}$ to $s_{tt.2}$ if $s_{s.t.i}$ executes $(s_{tt.1}, s_{tt.2})$.

3.2. Substitutions, patterns, pattern specifications, instances

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

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

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

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

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

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

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

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

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

3.3. The transition relation

Let $S_{s.c.c}$ be a set of conceptual configuration systems. Let C_{nf} be a set of conceptual configurations. An element t_{rn} of the form $(c_{nf.1}, c_{nf.2})$ is called a transition. Let T_{rn} be a set of transitions. The configurations $c_{nf.1}$ and $c_{nf.2}$ are called input and output configurations in $[\![t_{rn}]\!]$.

The transition relations of a IQS is modelled by the transition relation $t_{rn.rlt} \in T_{rn} \to B_l$ based on atomic exogenous transition relations, transition rules, atomic endogenous transition relations, the exogenous transition order and the endogenous transition order. The exogenous transition relation of the IQS is modelled by atomic exogenous transition relations and transition rules. The endogenous transition relation of the IQS is modelled by atomic endogenous transition relations.

Transitions from a configuration c_{nf} in $[\![t_{rn.rlt}]\!]$ are executed by a program in $[\![c_{nf}]\!]$. An element sequence p_{rg} is a program in $[\![c_{nf}]\!]$ if $[c_{nf}\ (0 : ()) :: state :: program] = (p_{rg})$. Let

P_{rg} be a set of programs. Thus, programs in configurations are specified by the conceptual $(0 : ()) :: state :: program$ from the substate *program* of the configurations. A program in $[\![c_{nf}]\!]$ is empty if $[c_{nf}\ (0 : ()) :: state :: program] = ()$. Atomic exogenous transition relations and transition rules define transitions executed by the first element of the program. Atomic endogenous transition relations define transitions executed in the case of the empty program.

Let $c_{nf.1} \to c_{nf.2}$ be a shortcut for $[t_{rn.rlt}\ c_{nf.1}\ c_{nf.2}] = true$. Transitions can return values. An element v_l is a value in $[\![c_{nf}]\!]$ if $v_l = [c_{nf}\ (0 : ()) :: state :: value]$. An element v_l is a value in $[\![t_{rn}]\!]$ if $c_{nf.1}[\![t_{rn}]\!] \to c_{nf.2}[\![t_{rn}]\!]$, and v_l is a value in $[\![c_{nf.2}[\![t_{rn}]\!]]\!]$. Thus, the returned values in transitions are specified by the conceptual $(0 : ()) :: state :: value$ from the substate *value* of output configurations of the transitions. A transition t_{rn} returns a value v_l if v_l is a value in $[\![t_{rn}]\!]$. A transition t_{rn} returns (or generates) an exception e_{xc} if e_{xc} is a value in $[\![t_{rn}]\!]$. A transition t_{rn} is normally executed if t_{rn} returns no exception.

The special variables $conf :: in$ and $val :: in$ reference to the current configuration and the value in the current configuration, respectively, in the definitions below.

An object $t_{rn.rlt.ex}$ of the form $(p_t, (v_{r.*}), (v_{r.s.*}), f_n)$ is an atomic exogenous transition relation if $(p_t, (v_{r.*}), (v_{r.s.*}))$ is a pattern specification, $conf :: in \notin \{v_{r.*}\} \cup \{v_{r.s.*}\}$, $val :: in \notin \{v_{r.*}\} \cup \{v_{r.s.*}\}$, $f_n \in S_b \to (T_{rn} \to B_l)$, $[support\ f_n] = \{s_b : [support\ s_b] = \{v_{r.*}\} \cup \{v_{r.s.*}\} \cup \{conf :: in, val : in\}, [s_b\ v_r] \in E_l$ for $v_r \in \{v_{r.*}\}$ and $[s_b\ v_r] \in E_{l.*}$ for $v_r \in \{v_{r.s.*}\}\}$. Let $T_{rn.rlt.ex}$ be a set of atomic exogenous transition relations. Let $c_{nf.1} \to_{f_n, s_b} c_{nf.2}$ be a shortcut for $[[f_n\ s_b]\ c_{nf.1}\ c_{nf.2}] = true$.

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

A function $t_{rn.rlt.ex.s} \in E_l \to T_{rn.rlt.ex}$ is called an atomic exogenous transition specification if $[support\ t_{rn.rlt.ex.s}]$ is finite. A relation $t_{rn.rlt.ex}$ is an atomic exogenous transition relation in $[\![t_{rn.rlt.ex.s}]\!]$ if $[t_{rn.rlt.ex.s}\ n_m] = t_{rn.rlt.ex}$ for some $n_m \in E_l$. An element n_m is a name in $[\![t_{rn.rlt.ex}, t_{rn.rlt.ex.s}]\!]$ if $[t_{rn.rlt.ex.s}\ n_m] = t_{rn.rlt.ex}$. An element n_m a name in $[\![t_{rn.rlt.ex.s}]\!]$ if n_m is a name in $[\![t_{rn.rlt.ex}, t_{rn.rlt.ex.s}]\!]$ for some $t_{rn.rlt.ex}$. Let $c_{nf.1} \to_{n_m, s_b} c_{nf.2}$ be a shortcut for $c_{nf.1} \to_{f_n[\![t_{rn.rlt.ex.s}\ n_m]\!], s_b} c_{nf.2}$.

An element r_l of the form $(p_t, (v_{r.*}), (v_{r.s.*}), (b_d))$ is a transition rule if $b_d \in E_{l.*}$, $(p_t, (v_{r.*}), (v_{r.s.*}))$ is a pattern specification, $conf :: in \notin \{v_{r.*}\} \cup \{v_{r.s.*}\}$, and $val :: in \notin \{v_{r.*}\} \cup \{v_{r.s.*}\}$. Let R_l be a set of transition rules.

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

An attribute element $r_{l.s}$ is called a transition rule specification if $[support\ r_{l.s}] \subseteq E_l$, and $[image\ r_{l.s}] \subseteq E_l$. A rule r_l is a rule in $[\![r_{l.s}]\!]$ if $[r_{l.s}\ n_m] = r_l$ for some $n_m \in E_l$. An element n_m is a name in $[\![r_l, r_{l.s}]\!]$ if $[r_{l.s}\ n_m] = r_l$. An element n_m a name in $[\![r_{l.s}]\!]$ if n_m is a name in $[\![r_l, r_{l.s}]\!]$ for some r_l.

A function $t_{rn.rlt.en} \in \{c_{nf} : [c_{nf}\ (0 : ()) :: state :: program] = ()\} \times C_{nf} \to B_l$ is called an atomic endogenous transition relation. Let $T_{rn.rlt.en}$ be a set of atomic endogenous transition relations.

A function $t_{rn.rlt.en.s} \in E_l \to T_{rn.rlt.en}$ is called an atomic endogenous transition specification if $[support\ t_{rn.rlt.en.s}]$ is finite. A relation $t_{rn.rlt.en}$ is an atomic endogenous transition relation in $[\![t_{rn.rlt.en.s}]\!]$ if $[t_{rn.rlt.en.s}\ n_m] = t_{rn.rlt.en}$ for some $n_m \in E_l$. An element n_m is a name in $[\![t_{rn.rlt.en}, t_{rn.rlt.en.s}]\!]$ if $[t_{rn.rlt.en.s}\ n_m] = t_{rn.rlt.en}$. An element n_m a name in $[\![t_{rn.rlt.en.s}]\!]$ if n_m is a name in $[\![t_{rn.rlt.en}, t_{rn.rlt.en.s}]\!]$ for some $t_{rn.rlt.en}$. Let $c_{nf} \to_{n_m} c_{nf}$ be a shortcut for $[[t_{rn.rlt.en.s}\ n_m]\ c_{nf}\ c_{nf.1}] = true$.

Let $[support\ t_{rn.rlt.ex.s}]$, $[support\ t_{rn.rlt.en.s}]$ and $[support\ r_{l.s}]$ be pairwise disjoint.

An element $o_{rd.trn.ex}$ of the form $(n_{m.*})$ is called an exogenous transition order in $[\![t_{rn.rlt.ex.s}$, $r_{l.s}]\!]$ if $\{n_{m.*}\} \subseteq [support\ t_{rn.rlt.ex.s}] \cup [support\ r_{l.s}]$, and the elements of $n_{m.*}$ are pairwise distinct. It specifies the order of application of atomic exogenous transition relations and transition rules.

An element $o_{rd.trn.en}$ of the form $(n_{m.*})$ is called an endogenous transition order in $[\![t_{rn.rlt.en.s}]\!]$ if $\{n_{m.*}\} \subseteq [support\ t_{rn.rlt.en.s}]$, and the elements of $n_{m.*}$ are pairwise distinct. It specifies the order of application of atomic endogenous transition relations.

The information about the transition rule specification and the transition orders is stored in the substate *transition* of the configurations. The conceptuals $(0 : rules) :: state :: transition$, $(-1 : exogenous, 0 : order) :: state :: transition$ and $(-1 : endogenous, 0 : order) :: state :: transition$ define the transition rule specification, exogenous transition order and endogenous transition order. The conceptual $(0 : history) :: state :: transition$ defines the substates that store the information about transitions preceding the transition to the current configuration.

An element c_{nf} is consistent with $(t_{rn.rlt.ex.s}, r_{l.s}, t_{rn.rlt.en.s}, o_{rd.trn.ex}, o_{rd.trn.en})$ if the following properties hold:

- if $[support\ t_{rn.rlt.ex.s}] \cap [support\ [c_{nf}\ (0 : rules) :: state :: transition]] = \emptyset$;

- if $[support\ o_{rd.trn.en}] \cap [support\ [c_{nf}\ (0 : rules) :: state :: transition]] = \emptyset$;
- if $r_{l.s} \subseteq [c_{nf}\ (0 : rules) :: state :: transition]$;
- if $n_{m.1} \prec_{[\![o_{rd.trn.ex}]\!]} n_{m.2}$, and $n_{m.1},\ n_{m.2} \in [c_{nf}\ (-1 : exogenous, 0 : order) :: state :: transition]$, then $n_{m.1} \prec_{[\![[c_{nf}\ (-1:exogenous,0:order)::state::transition]]\!]} n_{m.2}$;
- if $n_{m.1} \prec_{[\![o_{rd.trn.en}]\!]} n_{m.2}$, and $n_{m.1},\ n_{m.2} \in [c_{nf}\ (-1 : endogenous, 0 : order) :: state :: transition]$, then $n_{m.1} \prec_{[\![[c_{nf}\ (-1:endogenous,0:order)::state::transition]]\!]} n_{m.2}$.

Let $e_{l.*}\ \#\ c_{nf}$ be a shortcut for $[c_{nf}\ program.(0 : ()) : (e_{l.*})]$. Let $e_{l.*}\ \#\ v_l\ \#\ c_{nf}$ be a shortcut for $[c_{nf}\ program.(0 : ()) : (e_{l.*}), value.(0 : ()) : v_l]$.

Let $[add{-}history\ c_{nf.1}\ to\ c_{nf.2}]$ denote $[narrow\ c_{nf.1}\ [support\ c_{nf.1}] \setminus \{[c_{nf.1}\ (0 : history) :: state :: transition]\}] \cup [narrow\ c_{nf.1}\ \{[c_{nf.1}\ (0 : history) :: state :: transition]\}]$. A function $t_{rn.rlt} \in C_{nf.c} \times C_{nf} \to B_l$ is a transition relation in $[\![t_{rn.rlt.ex.s}, r_{l.s}, t_{rn.rlt.en.s},\ o_{rd.trn.ex}, o_{rd.trn.en}]\!]$ if it is defined by the following definition rules (the first proper rule is applied):

- if c_{nf} is not consistent with $(t_{rn.rlt.ex.s}, r_{l.s}, t_{rn.rlt.en.s}, o_{rd.trn.ex}, o_{rd.trn.en})$, then $[t_{rn.rlt}\ c_{nf}\ c_{nf.1}] = false$;
- if $t_{rn.rlt.ex} = [t_{rn.rlt.ex.s}\ n_m]$, e_l is an instance in $[\![p_{t.s}[\![t_{rn.rlt.ex}]\!], m_t, s_b]\!]$, $e_{l.*}\ \#\ c_{nf} \to_{n_m, s_b \cup (conf::in:c_{nf}, val::in:v_l[\![c_{nf}]\!])} e_{l.*.1}\ \#\ v_l\ \#\ c_{nf.1}$, and $v_l \neq und$, then $(execute{-}exogenous{-}transition,\ e_l,\ (n_m\ n_{m.*}))$, $e_{l.*}\ \#\ c_{nf} \to e_{l.*.1}\ \#\ v_l\ \#\ c_{nf.1}$;
- if $t_{rn.rlt.ex} = [t_{rn.rlt.ex.s}\ n_m]$, e_l is an instance in $[\![p_{t.s}[\![t_{rn.rlt.ex}]\!], m_t, s_b]\!]$, $e_{l.*}\ \#\ c_{nf} \to_{n_m, s_b \cup (conf::in:c_{nf}, val::in:v_l[\![c_{nf}]\!])} e_{l.*.1}\ \#\ u_{nd}\ \#\ c_{nf.1}$, then $(execute{-}exogenous{-}transition,\ e_l,\ (n_m\ n_{m.*}))$, $e_{l.*}\ \#\ c_{nf} \to (execute{-}exogenous{-}transition,\ e_l,\ (n_{m.*}))$, $e_{l.*}\ \#\ [add{-}history\ c_{nf.1}\ to\ c_{nf}]$;
- if $t_{rn.rlt.ex} = [t_{rn.rlt.ex.s}\ n_m]$, and e_l is not an instance in $[\![p_{t.s}[\![t_{rn.rlt.ex}]\!], m_t]\!]$, then $(execute{-}exogenous{-}transition,\ e_l,\ (n_m,\ n_{m.*}))$, $e_{l.*}\ \#\ c_{nf} \to (execute{-}exogenous{-}transition,\ e_l,\ (n_{m.*}))$, $e_{l.*}\ \#\ c_{nf}$;
- if $r_l = [[c_{nf}\ (0 : rules) :: state :: transition]\ n_m]$, and e_l is an instance in $[\![p_{t.s}[\![r_l]\!], m_t, s_b]\!]$, then $(execute{-}exogenous{-}transition,\ e_l,\ (n_m\ n_{m.*}))$, $e_{l.*}\ \#\ c_{nf} \to ([subst\ s_b \cup (conf :: in : c_{nf}, val :: in : v_l[\![c_{nf}]\!])\ b_d[\![r_l]\!]],\ (execute{-}exogenous{-}transition,\ e_l,\ (n_m\ n_{m.*}),\ (e_{l.*}),\ c_{nf}),\ e_{l.*}\ \#\ c_{nf}$;
- if $v_l \neq und$, then $(execute{-}exogenous{-}transition,\ e_l,\ (n_m\ n_{m.*}),\ (e_{l.*.1}),\ c_{nf.1}),\ e_{l.*}\ \#\ v_l\ \#\ c_{nf} \to e_{l.*}\ \#\ v_l\ \#\ c_{nf}$;
- $(execute{-}exogenous{-}transition,\ (n_m\ n_{m.*}),\ e_l,\ (e_{l.*.1}),\ c_{nf.1}),\ e_{l.*}\ \#\ und\ \#\ c_{nf} \to (execute{-}exogenous{-}transition,\ e_l,\ (n_{m.*}))$, $e_{l.*.1}\ \#\ [add{-}history\ c_{nf}\ to\ c_{nf.1}]$;

- if $r_l = [[c_{nf}\ (0 : rules) :: state :: transition]\ n_m]$, and e_l is not an instance in $[\![p_{t.s}[\![r_l]\!], m_t]\!]$, then $(execute{-}exogenous{-}transition,\ e_l,\ (n_m\ n_{m.*}))$, $e_{l.*}\ \#\ c_{nf} \rightarrow (execute{-}exogenous{-}transition,\ e_l,\ (n_{m.*}))$, $e_{l.*}\ \#\ c_{nf}$;
- $(execute{-}exogenous{-}transition,\ e_l,\ ())$, $e_{l.*}\ \#\ c_{nf} \rightarrow e_{l.*}\ \#\ und\ \#\ c_{nf}$;
- if $t_{rn.rlt.en} = [t_{rn.rlt.en.s}\ n_m]$, $c_{nf} \rightarrow_{n_m} e_{l.*}\ \#\ v_l\ \#\ c_{nf.1}$, and $v_l \neq und$, then $(execute{-}endogenous{-}transition,\ (n_m\ n_{m.*}))\ \#\ c_{nf} \rightarrow e_{l.*}\ \#\ v_l\ \#\ c_{nf.1}$;
- if $t_{rn.rlt.en} = [t_{rn.rlt.en.s}\ n_m]$, and $c_{nf} \rightarrow_{n_m} e_l\ e_{l.*}\ \#\ u_{nd}\ \#\ c_{nf.1}$, then $(execute{-}endogenous{-}transition,\ (n_m\ n_{m.*}))\ \#\ c_{nf} \rightarrow e_l\ e_{l.*}\ \#\ u_{nd}\ \#\ c_{nf.1}$;
- if $t_{rn.rlt.en} = [t_{rn.rlt.en.s}\ n_m]$, and $c_{nf} \rightarrow_{n_m} \#\ u_{nd}\ \#\ c_{nf.1}$, then $(execute{-}endogenous{-}transition,\ (n_m\ n_{m.*}))\ \#\ c_{nf} \rightarrow (execute{-}endogenous{-}transition,\ (n_{m.*}))\ \#\ [add{-}history\ c_{nf.1}\ to\ c_{nf}]$;
- $e_l,\ e_{l.*}\ \#\ c_{nf} \rightarrow (execute{-}exogenous{-}transition,\ e_l,\ [c_{nf}\ (-1 : exogenous, 0 : order) :: state :: transition]),\ e_{l.*}\ \#\ c_{nf}$;
- $\#\ c_{nf} \rightarrow (execute{-}endogenous{-}transition,\ [c_{nf}\ (-1 : endogenous, 0 : order) :: state :: transition]),\ \#\ c_{nf}$.

3.4. Conceptual transition systems

An object $s_{s.t.c}$ of the form $(s_{s.c.c}, t_{rn.rlt.ex.s}, r_{l.s}, t_{rn.rlt.en.s}, o_{rd.trn.ex}, o_{rd.trn.en})$ is a conceptual transition system if $s_{s.c.c}$ is a conceptual configuration system, $t_{rn.rlt.ex.s}$, $r_{l.s}$, $t_{rn.rlt.en.s}$, $o_{rd.trn.ex}$ and $o_{rd.trn.en}$ are an atomic exogenous transition specification, transition rule specification, atomic endogenous transition specification, exogenous transition order and endogenous transition order in $[\![A_{tm}[\![s_{s.c.c}]\!]]\!]$, and the sets $[support\ t_{rn.rlt.ex.s}]$, $[support\ t_{rn.rlt.en.s}]$ and $[support\ r_{l.s}]$ are pairwise disjoint. It specifies the transition system $(C_{nf}[\![s_{s.c.c}]\!], t_{rn.rlt}[\![t_{rn.rlt.ex.s}, r_{l.s}, t_{rn.rlt.en.s}, o_{rd.trn.ex}, o_{rd.trn.en}, m_t[\![s_{s.c.c}]\!]]\!])$. Let $S_{s.t.c}$ be a set of conceptual transition systems.

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

The objects $t_{rn.rlt.ex.s}$, $r_{l.s}$, $t_{rn.rlt.en.s}$, $o_{rd.trn.ex}$, $o_{rd.trn.en}$, $i_{ntr.a.s}[\![s_{s.c.c}]\!]$, $d_{f.s}[\![s_{s.c.c}]\!]$, $o_{rd.intr}[\![s_{s.c.c}]\!]$ and $m_t[\![s_{s.c.c}]\!]$ are called an atomic exogenous transition specification, transition rule specification, atomic endogenous transition specification, exogenous transition order, endogenous transition order, atomic element interpretation specification, element definition specification, element intepretation order and match in $[\![s_{s.t.c}]\!]$.

The function $t_{rn.rlt}[\![t_{rn.rlt.ex.s}, r_{l.s}, t_{rn.rlt.en.s}, o_{rd.trn.ex}, o_{rd.trn.en}, m_t]\!]$ is called a transition rela-

tion in $[\![s_{s.t.c}]\!]$. A system $s_{s.t.c}$ executes t_{rn} if $s_{tt.1}[\![t_{rn}]\!] \to s_{tt.2}[\![t_{rn}]\!]$. A system $s_{s.t.c}$ transits from $s_{tt.1}$ to $s_{tt.2}$ if $s_{s.t.c}$ executes $(s_{tt.1}, s_{tt.2})$.

An element e_l is interpretable in $[\![s_{s.t.c}]\!]$ if e_l is interpretable in $[\![s_{s.c.c}[\![s_{s.t.c}]\!]]\!]$.

An element e_l is executable in $[\![s_{s.t.c}]\!]$ if there exist n_m such that e_l is an instance in $[\![p_{t.s}[\![[t_{rn.rlt.ex.s}\ n_m]]\!], m_t]\!]$, or e_l is an instance in $[\![p_{t.s}[\![[r_{l.s}\ n_m]]\!], m_t]\!]$.

3.5. Conceptual information transition models

An object $m_{dl.t.q.c}$ of the form $(s_{s.t.c}, r_{pr.s}, r_{pr.q}, r_{pr.a})$ is a conceptual information transition model in $[\![s_{s.t.i}]\!]$ if $(s_{s.c.c}[\![s_{s.t.c}]\!], r_{pr.s}, r_{pr.q}, r_{pr.a})$ is a conceptual query model in $[\![s_{s.q}[\![s_{s.t.i}]\!]]\!]$, $[t_{rn.rlt.ex}[\![s_{s.t.i}]\!]\ q_r\ a_{ns}\ s_{tt.1}\ s_{tt.2}] = [t_{rn.rlt}[\![s_{s.t.c}]\!]\ [[r_{pr.s}\ s_{tt.1}]\ (0 : ()) :: state :: program : ([r_{pr.q}\ q_r])]\ [[r_{pr.s}\ s_{tt.2}]\ (0 : ()) :: state :: value : [r_{pr.a}\ a_{ns}]]]$, and $[t_{rn.rlt.en}[\![s_{s.t.i}]\!]\ a_{ns}\ s_{tt.1}\ s_{tt.2}] = [t_{rn.rlt}[\![s_{s.t.c}]\!]\ [[r_{pr.s}\ s_{tt.1}]\ (0 : ()) :: state :: program : ()]\ [[r_{pr.s}\ s_{tt.2}]\ (0 : ()) :: state :: value : [r_{pr.a}\ a_{ns}]]]$. Let $M_{dl.t.q.c}$ be a set of conceptual query transition models.

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

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

3.6. Extensions

A system $s_{s.t.i.1}$ is an extension of $s_{s.t.i.2}$ if $s_{s.q}[\![s_{s.t.i.1}]\!]$ is an extension of $s_{s.q}[\![s_{s.t.i.2}]\!]$, and $s_t[\![s_{s.t.i.1}]\!] \subseteq s_t[\![s_{s.t.i.2}]\!]$ for each $s_t \in \{t_{rn.rlt.ex}, t_{rn.rlt.en}\}$.

A system $s_{s.t.c.1}$ is an extension of $s_{s.t.c.2}$ if $s_{s.c.c}[\![s_{s.t.c.1}]\!]$ is an extension of $s_{s.c.c}[\![s_{s.t.c.2}]\!]$, $s_t[\![s_{s.t.c.1}]\!] \subseteq s_t[\![s_{s.t.c.2}]\!]$ for each $s_t \in \{t_{rn.rlt.ex.s}, r_{l.s}, t_{rn.rlt.en.s}\}$, and the following property hold:

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

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

3.7. Programs

A program p_{rg} is executed in $[\![t_{rn}]\!]$ if p_{rg} is a program in $[\![c_{nf.1}[\![t_{rn}]\!]]\!]$, and $c_{nf.1}[\![t_{rn}]\!] \rightarrow c_{nf.2}[\![t_{rn}]\!]$. A program p_{rg} executes (initiates) t_{rn} if p_{rg} is executed in $[\![t_{rn}]\!]$.

An element v_l is a value in $[\![p_{rg}, t_{rn}]\!]$ if p_{rg} executes $[\![t_{rn}]\!]$, and v_l is a value in $[\![t_{rn}]\!]$. A program p_{rg} returns v_l in $[\![t_{rn}]\!]$ if v_l is a value in $[\![p_{rg}, t_{rn}]\!]$. A program p_{rg} returns v_l in $[\![c_{nf}]\!]$ if there exists t_{rn} such that p_{rg} returns v_l in $[\![t_{rn}]\!]$, and $c_{nf} = c_{nf.1}[\![t_{rn}]\!]$.

A program p_{rg} returns (or generates) an exception e_{xc} in $[\![t_{rn}]\!]$ if e_{xc} is a value in $[\![p_{rg}, t_{rn}]\!]$. A program p_{rg} is normally executed in $[\![t_{rn}]\!]$ if p_{rg} is executed in $[\![t_{rn}]\!]$, and t_{rn} is normally executed.

An element e_l is executed in $[\![t_{rn}]\!]$ if there exist p_{rg} such that p_{rg} is executed in $[\![t_{rn}]\!]$, and $e_l = [p_{rg} \ . \ 1]$. An element e_l executes (initiates) t_{rn} if e_l is executed in $[\![t_{rn}]\!]$.

An element v_l is a value in $[\![e_l, t_{rn}]\!]$ if e_l is executed in $[\![t_{rn}]\!]$, and v_l is a value in $[\![t_{rn}]\!]$. An element v_l returns v_l in $[\![t_{rn}]\!]$ if v_l is a value in $[\![v_l, t_{rn}]\!]$. An element v_l returns v_l in $[\![c_{nf}]\!]$ if there exists t_{rn} such that v_l returns v_l in $[\![t_{rn}]\!]$, and $c_{nf} = c_{nf.1}[\![t_{rn}]\!]$.

An element e_l returns (or generates) an exception e_{xc} in $[\![t_{rn}]\!]$ if e_{xc} is a value in $[\![e_l, t_{rn}]\!]$. An element e_l is normally executed in $[\![t_{rn}]\!]$ if e_l is executed in $[\![t_{rn}]\!]$, and t_{rn} is normally executed.

3.8. Safe configurations, transitions, programs and elements

A configuration c_{nf} is locally safe if $v_l[\![c_{nf}]\!] \neq und$.

A transition t_{rn} is safe if $c_{nf.1}[\![t_{rn}]\!]$ and $c_{nf.2}[\![t_{rn}]\!]$ are locally safe.

A configuration c_{nf} is safe if there is no $c_{nf.1}$ such that $c_{nf} \rightarrow^* c_{nf.1}$ and $c_{nf.1}$ is not locally safe.

A program p_{rg} is safe in $[\![c_{nf}]\!]$ if p_{rg} is a program in $[\![c_{nf}]\!]$, and c_{nf} is safe. A program p_{rg} is safe if p_{rg} is safe in $[\![c_{nf}]\!]$ for each c_{nf}.

An element e_l is safe in $[\![c_{nf}]\!]$ if $e_l = [p_{rg}[\![c_{nf}]\!] \ . \ 1]$, and p_{rg} is safe in $[\![c_{nf}]\!]$. An element e_l is safe if e_l is safe in $[\![c_{nf}]\!]$ for each c_{nf}.

4. The CTSL language

The CTSL language (Conceptual Transition System Language) is a basic language of CTSs. The CCSL language is a sublanguage of CTSL. Interpretable and executable elements of CTSL are called basic elements of CTSs.

Let $s_b \subseteq (x : x_0,\ y : y_0,\ z : z_0,\ u : u_0,\ v : v_0,\ w : w_0,\ x_1 : x_{1.0},\ ...,\ x_{n_t} : x_{n_t.0},\ conf :: in : c_{nf},\ val :: in : v_l[\![c_{nf}]\!])$.

4.1. Syntax of CCSL

CTSL is an extension of CCSL. Therefore, atoms, elements, conceptual states, conceptual configurations, pattern specifications and element definitions are represented in CTSL as in CCSL.

The element $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ b_d) :: name :: n_m$ in CCSL represents the transition rule $(p_t, (v_{r.*}), (v_{r.s.*}), b_d)$ with the name n_m.

For simplicity, we omit the names of atomic transition relations and transition rules.

4.2. The special forms for atomic exogenous transition relations, transition rules and atomic endogenous transition relations

In this section we define the special forms for atomic exogenous transition relations, transition rules and atomic endogenous transition relations used below.

The form $(transition\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ f_n) :: name :: n_m$ denotes the atomic exogenous transition relation $(p_t, (v_{r.*}), (v_{r.s.*}), f_n)$ with the name n_m.

The objects $var\ (v_{r.*})$ and $seq\ (v_{r.s.*})$ in the form $(transition\ ...)$ can be omitted. The omitted objects correspond to $var\ ()$ and $seq\ ()$, respectively.

The form $(endogenous{-}transition\ f_n) :: name :: n_m$ denotes the atomic endogenous transition relation f_n with the name n_m.

Let $\{v_{r.*}\}$, $\{v_{r.s.*}\}$, $\{v_{r.*.1}\}$ and $\{v_{r.*.2}\}$ are pairwise disjoint, $\{v_{r.*.3}\} \subseteq \{v_{r.*}\} \cup \{v_{r.*.1}\} \cup \{v_{r.*.2}\}$, and $(e_{l.*}) \in \{(), und, abn\}$. The form $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.1})\ und\ (v_{r.*.2})\ val\ (v_{r.*.3})\ e_{l.*}\ where\ c_{nd}\ then\ b_d)$ called a rule form is defined as follows:

- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ e_{l.*}\ where\ c_{nd}\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.1})\ und\ (v_{r.*.2})\ val\ (v_{r.*.3})\ e_{l.*}\ then\ (if\ c_{nd}\ then\ b_d\ else\ und))$;
- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3},\ v_r)\ e_{l.*}\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ e_{l.*}\ then\ (let\ w\ be\ v_r\ in\ [subst\ (v_r :: * : w)\ b_d]))$, where w is a new element that does not occur in this definition;
- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ ()\ e_{l.*}\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ e_{l.*}\ then\ b_d)$;
- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1},\ v_r)\ abn\ (v_{r.*.2})\ e_{l.*}\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ e_{l.*}\ then\ (if\ (v_r\ is\ undefined)\ then\ und\ else\ b_d))$;

- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ ()\ abn\ (v_{r.*.2})\ e_{l.*}\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2})\ e_{l.*}\ then\ b_d)$;
- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2},\ v_r)\ e_{l.*}\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2})\ e_{l.*}\ then\ (if\ (v_r\ is\ abnormal)\ then\ v_r\ else\ b_d))$;
- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ ()\ e_{l.*}\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ e_{l.*}\ then\ b_d)$;
- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ (if\ (val :: in\ is\ undefined)\ then\ skip\ else\ b_d)$;
- $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ then\ b_d)$ is a shortcut for $(rule\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ (if\ (val :: in\ is\ abnormal)\ then\ skip\ else\ b_d)$.

The element c_{nd} specifies the restriction on the values of the pattern variables. The undefined value is propagated through the variables of $v_{r.*.1}$. Abnormal values are propagated through the variables of $v_{r.*.2}$. The sequence $e_{l.*}$ specifies propagation of abnormal values depending on the value of $val :: in$. The undefined value is propagated when $e_{l.*} = und$. Abnormal values are propagated when $e_{l.*} = abn$. The special element $v_r :: *$ references to the value of element associated with the pattern variable v_r. A pattern variable is evaluated if the element associated with it is evaluated. Thus, the sequence $v_{r.*.3}$ contains evaluated pattern variables. A pattern variable is quoted if the element associated with it is not evaluated. Let $F_{rm.r}$ be a set of rule forms.

The objects $var\ (v_{r.*})$, $seq\ (v_{r.s.*})$, $und\ (v_{r.*.1})$, $abn\ (v_{r.*.2})$, $val\ (v_{r.*.3})$ and $where\ c_{nd}$ in the form $(rule\ ...)$ can be omitted. The omitted objects correspond to $var\ ()$, $seq\ ()$, $und\ ()$, $abn\ ()$, $val\ ()$ and $where\ true$, respectively.

5. Semantics of executable elements in CTSL

5.1. Element interpretation

The element $x :: value$ returning the interpretation of x is defined by the rule

$(rule\ \ x :: value\ \ var\ \ (x)\ \ abn\ \ then\ \ x :: value :: atm)$;

$(transition\ \ x :: value :: atm\ \ var\ \ (x)\ \ then\ \ f_n)$,

where $x_0 :: value :: atm\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [value\ x_0\ c_{nf}]\ \#\ c_{nf}$.

5.2. Abnormal elements operations

The element *und* is defined by the rule

$(rule\ und\ abn\ then\ und :: q)$.

The element e_{xc} is defined by the rule

$(rule\ x\ var\ (x)\ abn\ where\ (x\ is\ exception)\ then\ x :: q) :: name :: (\text{"@"},\ exception)$.

The rule satisfies the property: $n_m \prec_{[\![o_{rd.trn.ex}]\!]} (\text{"@"}, exception)$ for each n_m such that n_m is a name of an atomic exogenous transition relation or transition rule with the pattern distinct from v_r, where v_r is a variable of this pattern.

The element $e_l :: q$ is defined by the rule

$(rule\ x :: q\ var\ (x)\ abn\ then\ x :: q :: value)$.

The element e_l of the form $(catch :: u\ x\ y)$ called an undefined value handler is defined as follows:

$(transition\ (catch :: u\ x\ y)\ var\ (x)\ seq\ (y)\ then\ f_n)$,

where $(catch :: u\ x_0\ y_0),\ e_{l.*}\ \#\ v_l\ \#\ c_{nf} \to_{f_n,s_b} [subst\ (x_0 : v_l)\ y_0],\ e_{l.*}\ \#\ true\ \#\ c_{nf}$. The elements x and y are called a variable and body in $[\![e_l]\!]$. The element e_l replaces all occurences of x in y by the current value, resets the current value to *true* and executes the modified body.

The element e_l of the form $(catch\ x\ y)$ called an exception handler is defined as follows:

$(rule\ (catch\ x\ y)\ var\ (x)\ seq\ (y)\ und\ then\ (catch :: u\ x\ y))$.

The elements x and y are called a variable and body in $[\![e_l]\!]$.

The element e_l of the form $(throw\ x)$ is defined by the rule

$(rule\ (throw\ x)\ var\ (x)\ val\ (x)\ abn\ then\ (throw\ x :: *) :: atm)$;

$(transition\ (throw\ x) :: atm\ var\ (x)\ then\ f_n)$,

where $(throw\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \to_{f_n,s_b} e_{l.*}\ \#\ x_0\ \#\ c_{nf}$. The element x is called a body in $[\![e_l]\!]$.

The deletion $(delete{-}exception\ x)$ of the exception of the type x is defined by the rule

$(rule\ (delete{-}exception\ x)\ var\ (x)\ und\ then\ (catch\ w$
$(if\ ((w\ is\ exception)\ and\ (((element\ in\ w)\ ..\ type)\ =\ x :: q))$
$then\ (throw\ true)\ else\ (throw\ w :: q))))$.

5.3. Statements

The element *skip* is defined as follows:

$(rule\ skip\ abn\ then\ skip :: atm)$;

$(transition\ skip :: atm\ then\ f_n)$,

where $skip :: atm, e_{l.*}\ \#\ c_{nf} \to_{f_n,s_b} e_{l.*}\ \#\ c_{nf}$.

The sequential composition e_l of the form $(seq\ e_{l.*})$ is defined by the rule

$(rule\ \ (seq\ \ x)\ \ var\ \ (x)\ \ seq\ \ (x)\ \ then\ \ x)$

The elements of $e_{l.*}$ are called elements in $[\![e_l]\!]$ and $e_{l.*}$ is called a body in $[\![e_l]\!]$. The element e_l executes its elements sequentially from left to right.

The conditional element $(if\ x\ then\ y\ else\ z)$ is defined as follows:

$(rule\ \ (if\ \ x\ \ then\ \ y\ \ else\ \ z)\ \ var\ \ (x)\ \ seq\ \ (y,\ \ z)\ \ val\ \ (x)\ \ abn$

$then\ \ (if\ \ x :: *\ \ then\ \ y\ \ else\ \ z) :: atm);$

$(transition\ \ (if\ \ x\ \ then\ \ y\ \ else\ \ z) :: atm\ \ var\ \ (x)\ \ seq\ \ (y,\ \ z)\ \ then\ \ f_n),$

where $(if\ x_0\ then\ y_0\ else\ z_0) :: atm, e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} [if\ [x_0 \neq und]\ then\ y_0\ else\ z_0], e_{l.*}\ \#\ c_{nf}.$

The element $(if\ x\ then\ y)$ is a shortcut for $(if\ x\ then\ y\ else\ skip)$.

The conditional element $(if\ x\ then\ y\ elseif\ z\ then\ u\ ...\ else\ v)$ is defined as follows:

$(definition\ \ (if\ \ x\ \ then\ \ y\ \ elseif\ \ z)\ \ var\ \ (x)\ \ seq\ \ (y,\ \ z)\ \ abn$

$then\ \ (if\ \ x\ \ then\ \ y\ \ else\ \ (if\ \ z))).$

The element e_l of the form $(let\ x\ be\ y\ in\ z)$ is defined as follows:

$(rule\ \ (let\ \ x\ \ be\ \ y\ \ in\ \ z)\ \ var\ \ (x)\ \ seq\ \ (y,\ \ z)\ \ abn\ \ then\ \ (let\ \ x\ \ be\ \ y\ \ in\ \ z) :: atm);$

$(transition\ \ (let\ \ x\ \ be\ \ y\ \ in\ \ z) :: atm\ \ var\ \ (x)\ \ seq\ \ (y,\ \ z)\ \ then\ \ f_n),$

where $(let\ x_0\ be\ y_0\ in\ z_0) :: atm, e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} y_0,\ (let\ x_0\ be{-}val{-}in\ z_0), e_{l.*}\ \#\ c_{nf}.$ The elements x, y and z are called a substitution variable, substitution value and substitution body in $[\![e_l]\!]$.

The auxiliary element $(let\ x\ be{-}val{-}in\ y)$ is defined as follows:

$(transition\ \ (let\ \ x\ \ be{-}val{-}in\ \ y)\ \ var\ \ (x)\ \ seq\ \ (y)\ \ abn\ \ then\ \ f_n),$

where $(let\ x_0\ be{-}val{-}in\ y_0), e_{l.*}\ \#\ v_l\ \#\ c_{nf} \rightarrow_{f_n,s_b} [subst\ (x_0 : v_l)\ y_0], e_{l.*}\ \#\ c_{nf}.$

The element e_l of the form $(let :: seq\ x\ be\ y\ in\ z)$, where $x \in E_{l.(*)}$, $y \in E_{l.(*)}$, and $[len\ x] = [len\ y]$, is defined by the rule

$(rule\ \ (let :: seq\ \ x,\ \ y\ \ be\ \ (z),\ \ u\ \ in\ \ v)\ \ var\ \ (x)\ \ seq\ \ (y,\ \ z,\ \ u,\ \ v)\ \ abn$

$then\ \ (let\ \ x\ \ be\ \ z\ \ in\ \ (let :: seq\ \ y\ \ be\ \ u\ \ in\ \ v)));$

$(rule\ \ (let :: seq\ \ be\ \ in\ \ v)\ \ seq\ \ (v)\ \ abn\ \ then\ \ v).$

The elements x, y and z are called a substitution variables specification, substitution values specification and substitution body in $[\![e_l]\!]$. The elements of x and y are called substitution variables and substitution values in $[\![e_l]\!]$.

The iterator e_l of the form $(while\ x\ do\ y)$ is defined by the rule

$(if\ \ (while\ \ x\ \ do\ \ y)\ \ var\ \ (x)\ \ seq\ \ (y)\ \ abn\ \ then\ \ (if\ \ x\ \ then\ \ y\ \ (while\ \ x\ \ do\ \ y))).$

The elements x and y are called a condition and body in $[\![e_l]\!]$.

The iterator e_l of the form $(foreach\ x\ in\ y\ do\ z)$ is defined as follows:

$(rule\ (foreach\ x\ in\ y\ do\ z)\ var\ (x,\ y)\ seq\ (z)\ val\ (y)\ abn\ where\ (y :: *\ is\ sequence)$
$then\ (foreach1\ x\ in\ y :: *\ do\ z))$.

The objects x, y and z are called an iteration variable, iteration structure specifier and body in $[\![e_l]\!]$. The element e_l executes sequentially z for values of x from $e_{l.1}$, where $e_{l.1}$ is the value of y.

The element $(foreach1\ x\ in\ y\ do\ z)$ is defined by the rules

$(rule\ (foreach1\ x\ in\ ()\ do\ y)\ var\ (x)\ seq\ (y)\ abn\ then)$;

$(rule\ (foreach1\ x\ in\ (y\ z)\ do\ v)\ var\ (x,\ y)\ seq\ (z,\ v)\ abn$
$then\ (let\ x\ be\ y\ in\ v),\ (foreach1\ x\ in\ (z)\ do\ v))$.

5.4. Characteristic functions for defined concepts

An object $d_{f.c}$ is a concept definition if $d_{f.c}$ is an atomic transition relation of the form $(transition\ n_m\ if\ (e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ f_n)$, or $d_{f.c}$ is a transition rule of the form $(rule\ n_m\ if\ (e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ b_d)$. Concept definitions specify concepts and their instances. Concepts specified by them are called defined concepts. The elements $e_{l.1}$ and $e_{l.2}$ are called an instance pattern and concept pattern in $[\![d_{f.c}]\!]$. The element $(e_{l.1}\ is\ e_{l.2})$ is called a characteristic function in $[\![d_{f.c}]\!]$. Let $D_{f.c}$ be a set of concept definitions.

An element $c_{ncp.d}$ is a defined concept in $[\![d_{f.c}, s_b]\!]$ if c_{ncp} is an instance in $[\![(e_{l.2}, var\ (v_{r.*})\ seq\ (v_{r.s.*})), m_t, s_b]\!]$. An element $c_{ncp.d}$ is a defined concept in $[\![d_{f.c}]\!]$ if there exists s_b such that $c_{ncp.d}$ is a concept in $[\![d_{f.c}, s_b]\!]$. An element $c_{ncp.d}$ is a defined concept in $[\![c_{nf}]\!]$ if there exists $d_{f.c}[\![c_{nf}]\!]$ such that $c_{ncp.d}$ is a concept in $[\![d_{f.c}]\!]$. Let $C_{ncp.d}$ be a set of defined concepts.

An element i_{nstn} is an instance in $[\![d_{f.c}, s_b]\!]$ if i_{nstn} is an instance in $[\![(e_{l.1}, var\ (v_{r.*})\ seq\ (v_{r.s.*})), m_t, s_b]\!]$. An element i_{nstn} is an instance in $[\![d_{f.c}]\!]$ if there exists s_b such that $c_{ncp.d}$ is an instance in $[\![d_{f.c}, s_b]\!]$.

An element i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}, d_{f.c}]\!]$ if i_{nstn} is an instance in $[\![d_{f.c}]\!]$, $c_{ncp.d}$ is a defined concept in $[\![d_{f.c}]\!]$, and there exist $c_{nf.1}$ and v_l such that $(execute{-}exogenous{-}transition, (i_{nstn}\ is\ c_{ncp.d}),\ (n_m))\ \#\ c_{nf} \rightarrow^*\ \#\ v_l\ \#\ c_{nf.1}$, and $v_l \neq und$. An element i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}]\!]$ if there exists $d_{f.c}$ such that i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}, d_{f.c}]\!]$. An element $c_{ncp.d}$ is an instance in $[\![c_{nf}]\!]$ if there exists $c_{ncp.d}$ such that i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}]\!]$. Let I_{nstn} be a set of instances.

A set s_t is called a content in $[\![c_{ncp.d}, c_{nf}]\!]$ if s_t is a set of all i_{nstn} such that i_{nstn} is an instance in $[\![c_{ncp.d}, c_{nf}]\!]$. Let $[content\ c_{ncp.d}\ c_{nf}]$ denote the content in $[\![c_{ncp.d}, c_{nf}]\!]$.

The notion of defined concepts is extended to the rules of the form $(rule\ (e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ val\ (v_{r.*.3})\ where\ c_{nd}\ then\ b_d)$. Let r_l have this form. An element $c_{ncp.d}$ is a defined concept in $[\![r_l, s_b]\!]$ if $c_{ncp.d}$ is a defined concept in $[\![r_{l.1}, s_b]\!]$, where $r_{l.1}$ is a rule of the form $(rule\ (e_{l.1}\ is\ e_{l.2})\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ b_{d.1})$ such that r_l is reduced to $r_{l.1}$.

The element $(x\ is\ atom)$ specifying that x is an atom is defined by the rule
$(rule\ (x\ is\ atom)\ var\ (x)\ abn\ then\ (x\ is\ atom) :: value)$.

The element $(x\ is\ update)$ specifying that x is an element update is defined by the rule
$(rule\ (x\ is\ update)\ var\ (x)\ abn\ then\ (x\ is\ update) :: value)$.

The element $(x\ is\ multi{-}attribute)$ specifying that x is a multi-attribute element is defined by the rule
$(rule\ (x\ is\ multi{-}attribute)\ var\ (x)\ abn\ then\ (x\ is\ multi{-}attribute) :: value)$.

The element $(x\ is\ attribute)$ specifying that x is an attribute element is defined by the rule
$(rule\ (x\ is\ attribute)\ var\ (x)\ abn\ then\ (x\ is\ attribute) :: value)$.

The element $(x\ is\ sorted)$ specifying that x is a sorted element is defined by the rule
$(rule\ (x\ is\ sorted)\ var\ (x)\ abn\ then\ (x\ is\ sorted) :: value)$.

The element $(x\ is\ undefined)$ specifying that x equals und is defined by the rule
$(rule\ (x\ is\ undefined)\ var\ (x)\ abn\ then\ (x\ is\ undefined) :: value)$.

The element $(x\ is\ defined)$ specifying that x does not equal und is defined by the rule
$(rule\ (x\ is\ defined)\ var\ (x)\ abn\ then\ (x\ is\ defined) :: value)$.

The element $(x\ is\ exception)$ specifying that x is an exception is defined by the rule
$(rule\ (x\ is\ exception)\ var\ (x)\ abn\ then\ (x\ is\ exception) :: value)$.

The element $(x\ is\ normal)$ specifying that x is a normal element is defined by the rule
$(rule\ (x\ is\ normal)\ var\ (x)\ abn\ then\ (x\ is\ normal) :: value$.

The element $(x\ is\ normal)$ specifying that x is an abnormal element is defined by the rule
$(rule\ (x\ is\ abnormal)\ var\ (x)\ abn\ then\ (x\ is\ abnormal) :: value$.

The element $(x\ is\ sequence)$ specifying that x is a sequence element is defined by the rule
$(rule\ (x\ is\ sequence)\ var\ (x)\ abn\ then\ (x\ is\ sequence) :: value)$.

The element $(x\ is\ set)$ specifying that the elements of the sequence element x are pairwise distinct is defined as follows:
$(rule\ (x\ is\ set)\ var\ (x)\ abn\ then\ (x\ is\ set) :: value)$.

The element $(x\ is\ empty)$ specifying that x is an empty element is defined by the rule
$(rule\ \ (x\ \ is\ \ empty)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ empty) :: value)$.

The element $(x\ is\ nonempty)$ specifying that x is not an empty element is defined by the rule
$(rule\ \ (x\ \ is\ \ nonempty)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ nonempty) :: value)$.

The element $(x\ is\ conceptual)$ specifying that x is a conceptual is defined by the rule
$(rule\ \ (x\ \ is\ \ conceptual)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ conceptual) :: value)$.

The element $(x\ is\ (conceptual\ in\ y))$ specifying that x is a conceptual in the context of the state y is defined by the rule
$(rule\ \ (x\ \ is\ \ (conceptual\ \ in\ \ y))\ \ var\ \ (x,\ \ y)\ \ abn\ \ then\ \ (x\ \ is\ \ (conceptual\ \ in\ \ y)) :: value$.

The element $(x\ is\ state)$ specifying that x is a conceptual state is defined by the rule
$(rule\ \ (x\ \ is\ \ state)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ state) :: value)$.

The element $(x\ is\ configuration)$ specifying that x is a conceptual configuration is defined by the rule
$(rule\ \ (x\ \ is\ \ configuration)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ configuration) :: value)$.

The element $(x\ is\ nat)$ specifying that x is a natural number is defined by the rule
$(rule\ \ (x\ \ is\ \ nat)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ nat) :: value)$.

The element $(x\ is\ nat0)$ specifying that x is either a natural number, or a zero is defined by the rule
$(rule\ \ (x\ \ is\ \ nat0)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ nat0) :: value)$.

The element $(x\ is\ int)$ specifying that x is an integer is defined by the rule
$(rule\ \ (x\ \ is\ \ int)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ int) :: value)$.

The element $(x\ is\ (satisfiable\ in\ y))$ specifying that x is satisfiable in the context of variables y is defined by the rule
$(rule\ \ (x\ \ is\ \ (satisfiable\ \ in\ \ y))\ \ var\ \ (x)\ \ seq\ \ (y)\ \ abn$
$then\ \ (x\ \ is\ \ (satisfiable\ \ in\ \ (y))) :: value)$.

The element $(x\ is\ (valid\ in\ y))$ specifying that x is valid in the context of variables y is defined by the rule
$(rule\ \ (x\ \ is\ \ (valid\ \ in\ \ y))\ \ var\ \ (x)\ \ seq\ \ (y)\ \ abn\ \ then\ \ (x\ \ is\ \ (valid\ \ in\ \ (y))) :: value)$.

The element $(x\ is\ (sequence\ y))$ specifying that x is a sequence element such that the value in $[\![(e_l\ is\ y)]\!]$ does not equal und for each element e_l of x is defined by the rule
$(rule\ \ ((x\ \ y)\ \ is\ \ (sequence\ \ z))\ \ var\ \ (x,\ \ z)\ \ seq\ \ (y)\ \ abn$

$then\ ((x\ is\ z)\ and\ ((y)\ is\ (sequence\ z)))$;

$(rule\ (()\ is\ (sequence\ x))\ var\ (x)\ abn\ then\ true)$.

The element $(x\ is\ rule)$ specifying that x is a rule is defined as follows:

$(rule\ (x\ is\ rule)\ var\ (x)\ abn\ then\ (x\ is\ rule) :: value)$;

$(interpretation\ (x\ is\ rule)\ var\ (x)\ then\ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in R_l]\ then\ true\ else\ und]$.

The element $(x\ is\ (rule\ in\ y))$ specifying that x is a rule in the context of the state y is defined as follows:

$(rule\ (x\ is\ (rule\ in\ y))\ var\ (x,\ y)\ abn\ then\ (x\ is\ (rule\ in\ y)) :: value)$;

$(definition\ (x\ is\ (rule\ in\ y))\ var\ (x,\ y)\ where\ ((x\ is\ rule)\ and\ (y\ is\ state))$

$then\ (x\ is\ conceptual\ in\ y) :: atm)$;

$(interpretation\ (x\ is\ (conceptual\ in\ y)) :: atm\ var\ (x,\ y)\ then\ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in R_l[\![y_0]\!]]\ then\ true\ else\ und]$.

The element $(x\ is\ transition)$ specifying that x is a transition is defined as follows:

$(rule\ (x\ is\ transition)\ var\ (x)\ abn\ then\ (x\ is\ transition) :: value)$;

$(interpretation\ (x\ is\ transition)\ var\ (x)\ then\ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in T_{rn}]\ then\ true\ else\ und]$.

5.5. Elements operations

The element () is defined by the rule

$(rule\ ()\ abn\ then\ () :: q)$.

The element $(len\ x)$ specifying the length of the element x is defined by the rule

$(rule\ (len\ x)\ var\ (x)\ val\ (x)\ abn\ then\ (len\ x :: * :: q) :: value)$.

The element $(x\ =\ y)$ specifying the equality of the elements x and y is defined by the rule

$(rule\ (x\ =\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x :: * :: q\ =\ y :: * :: q) :: value)$.

The element $(x\ !=\ y)$ specifying the inequality of the elements x and y is defined in the similar way.

The element $(x\ .\ y)$ specifying the y-th element of the sequence element x is defined by the rule

$(rule\ (x\ .\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x :: * :: q\ .\ y :: * :: q) :: value)$.

The element $(x\ ..\ y)$ specifying the value of the attribute element x for the attribute y is defined by the rule

$(rule\ (x\ ..\ y)\ var\ (x,\ y)\ val\ (x)\ abn\ then\ (x :: * :: q\ ..\ y) :: value)$.

The element $(x\ +\ y)$ specifying the concatenation of the sequence elements x and y is defined by the rule

$(rule\ (x\ +\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x :: * :: q\ +\ y :: * :: q) :: value)$.

The element $(x\ .+\ y)$ specifying the addition of the element x to the head of the sequence element y is defined by the rule

$(rule\ (x\ .+\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x :: * :: q\ .+\ y :: * :: q) :: value)$.

The element $(x\ .+ :: set\ y)$ specifying the addition of the element x to the head of the sequence element y representing a set is defined as follows:

$(rule\ (x\ .+ :: set\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ where\ (y :: *\ is\ set)$
$then\ (x :: * :: q\ .+ :: set\ y :: * :: q) :: value)$.

The element $(x\ +.\ y)$ specifying the addition of the element y to the tail of the sequence element x is defined by the rule

$(rule\ (x\ +.\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x :: * :: q\ +.\ y :: * :: q) :: value)$.

The element $(x\ +. :: set\ y)$ specifying the addition of the element y to the tail of the sequence element x representing a set is defined by the rule

$(rule\ (x\ +. :: set\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ where\ (x :: *\ is\ set)$
$then\ (x :: * :: q\ +. :: set\ y :: * :: q) :: abn)$.

The element $(x\ -. :: set\ y)$ specifying the deletion of the element y from the sequence element x representing a set is defined by the rule

$(rule\ (x\ -. :: set\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ where\ (x :: *\ is\ set)$
$then\ (x :: * :: q\ -. :: set\ y :: * :: q) :: value)$.

The element $(upd\ x\ y_1 : z_1, ..., y_{n_t} : z_{n_t})$ specifying the sequential updates of the attribute element x at the points y_1, ..., y_{n_t} by z_1, ..., z_{n_t} is defined by the rules

$(rule\ (upd\ x\ y)\ var\ (x)\ seq\ (y)\ val\ (x)\ abn$
$where\ ((x :: *\ is\ attribute)\ and\ ((y)\ is\ (sequence\ update)))\ then\ (upd :: att\ x :: *\ y))$;
$(rule\ (upd :: att\ x\ y\ z)\ var\ (y)\ seq\ (z)\ und\ (x)\ abn$
$then\ (upd :: att\ (upd1 :: att\ x\ y)\ z))$;
$(rule\ (upd :: att\ x)\ var\ (x)\ then\ x)$;
$(rule\ (upd1 :: att\ x\ y : z)\ var\ (x,\ y,\ z)\ val\ (z)\ abn$
$then\ (upd1 :: att\ x\ y : z :: * :: q) :: value)$.

The element $(upd\ x\ y : z)$ specifying the update of the sequence element x at the index y by z is defined by the rule

$(rule\ (upd\ x\ y\ z)\ var\ (x,\ y,\ z)\ val\ (x,\ y,\ z)\ abn$
$then\ (upd :: seq\ x :: * :: q\ y :: * :: q\ z :: * :: q) :: value)$.

The element $(x\ in :: set\ y)$ specifying that x is an element of the sequence element y is defined as follows:

$(rule\ (x\ in :: set\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x\ in :: set\ y) :: value)$.

The element $(x\ includes :: set\ y)$ specifying that the sequence element x includes the elements of the sequence element y is defined as follows:

$(rule\ (x\ includes :: set\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x\ includes :: set\ y) :: value)$.

The element $(attributes\ in\ x)$ specifying the sequence of attributes of the attribute element x is defined by the rule

$(rule\ (attributes\ in\ x)\ var\ (x)\ abn\ then\ (attributes\ in\ x) :: value)$.

The element $(values\ in\ x)$ specifying the sequence of attribute values of the attribute element x is defined by the rule

$(rule\ (values\ in\ x)\ var\ (x)\ abn\ then\ (values\ in\ x) :: value)$.

The element $(element\ in\ x)$ specifying the element of the sorted element x is defined by the rule

$(rule\ (element\ in\ x)\ var\ (x)\ abn\ then\ (element\ in\ x) :: value)$.

The element $(sort\ in\ x)$ specifying the sort of the sorted element x is defined by the rule

$(rule\ (sort\ in\ x)\ var\ (x)\ abn\ then\ (sort\ in\ x) :: value)$.

The element $(attribute\ in\ x)$ specifying the attribute of the element update x is defined by the rule

$(rule\ (attribute\ in\ x)\ var\ (x)\ abn\ then\ (attribute\ in\ x) :: value)$.

The element $(value\ in\ x)$ specifying the value of the element update x is defined by the rule

$(rule\ (value\ in\ x)\ var\ (x)\ abn\ then\ (value\ in\ x) :: value)$.

The element $(unbracket\ (x))$ is defined by the rule

$(rule\ (unbracket\ (x))\ seq\ (x)\ abn\ then\ x)$.

5.6. Boolean operations

The element *true* is defined by the rule:

$(rule\ true\ abn\ then\ true :: value)$.

The element $(x\ and\ y)$ specifying the conjunction of x and y is defined by the rule:
$(rule\ (x\ and\ y)\ var\ (x,\ y)\ abn\ then\ (if\ x\ then\ y\ else\ und))$.

The elements $(x\ o_p\ y)$, where $o_p \in \{or, =>, <=>\}$ specifying the disjunction, implication and equivalence of x and y are defined in the similar way.

The element $(x_1\ and\ x_2\ and\ ...\ and\ x_{n_t})$ specifying the conjunction of x_1, x_2, ..., x_{n_t} is defined by the rule
$(rule\ (x\ and\ y\ and\ z)\ var\ (x,\ y)\ seq\ (z)\ abn\ then\ ((x\ and\ y)\ and\ z)$.

The element $(x_1\ or\ x_2\ or\ ...\ or\ x_{n_t})$ specifying the disjunction of x_1, x_2, ..., x_{n_t} is defined in the similar way.

The element $(not\ x)$ specifying the negation of x is defined by the rule
$(rule\ (not\ x)\ var\ (x)\ abn\ then\ (if\ x\ then\ und\ else\ true))$.

5.7. Integers

The element i_{nt} is defined by the rule
$(rule\ x\ var\ (x)\ abn\ where\ (x\ is\ int)\ then\ x :: q) :: name :: (\text{"@"},\ int)$.
The rule satisfies the property: $(\text{"@"}, exception) \prec_{[\![o_{rd.trn.ex}]\!]} (\text{"@"}, int)$.

The element $(x\ +\ y)$ specifying the sum of x and y is defined by the rule
$(rule\ (x\ +\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x :: * :: q\ +\ y :: * :: q) :: value)$.

The elements $(x\ o_p\ y)$, where $o_p \in \{-, *, div, mod\}$, specifying the integer operations $-$, $*$, div and mod, are defined in the similar way.

The element $(x\ <\ y)$ specifying that x is less than y is defined by the rule
$(rule\ (x\ <\ y)\ var\ (x,\ y)\ val\ (x,\ y)\ abn\ then\ (x :: * :: q\ <\ y :: * :: q) :: value)$.

The elements $(x\ o_p\ y)$, where $o_p \in \{<=, >, >=\}$, specifying the integer relations $\leq$, $>$ and $\geq$, are defined in the similar way.

5.8. Conceptuals operations

The element $(x\ in\ y)$ specifying the value of the conceptual x in the state y is defined by the rule
$(rule\ (x\ in\ y)\ var\ (x,\ y)\ abn\ then\ (x\ in\ y) :: value)$.

The element $x :: state :: y$ specifying the value of the conceptual x in the substate with the name y of the current configuration is defined by the rule
$(rule\ x :: state :: y\ var\ (x,\ y)\ abn\ then\ (x :: state :: y) :: value)$.

The element c_{ncpl} is a shortcut for $c_{ncpl} :: ()$.

The assignment $(c_{ncpl} :: state :: n_m \ ::= \ e_l)$ of e_l to $c_{ncpl} :: state :: n_m$ is defined as follows:

$(rule \ (x :: state :: z \ ::= \ y) \ var \ (x, \ y, \ z) \ val \ (y) \ abn \ where \ (x \ is \ conceptual)$

$then \ (x :: state :: z \ ::= \ y :: *) :: atm)$;

$(transition \ (x :: state :: z \ ::= \ y) :: atm \ var \ (x, \ y, \ z) \ then \ f_n)$,

where $(x_0 :: state :: z_0 ::= y_0) :: atm, e_{l.*} \ \# \ c_{nf} \rightarrow_{f_n, s_b} e_{l.*} \ \# \ [[c_{nf} \ z_0] \ x_0 : y_0]$.

The element $(c_{ncpl} \ ::= \ e_l)$ is a shortcut for $(c_{ncpl} :: () \ ::= \ e_l)$. The elements $(c_{ncpl} :: state :: n_m \ ::=)$ and $(c_{ncpl} \ ::=)$ are shortcuts for $(c_{ncpl} :: state :: n_m \ ::= \ und)$ and $(c_{ncpl} \ ::= \ und)$.

5.9. Countable concepts operations

A normal element $c_{ncp.c}$ is a countable concept in $[\![c_{nf}]\!]$ if $[[c_{nf} \ countable{-}concept] \ (0 : c_{ncp.c})] \in N_t$. Thus, the substate $countable{-}concept$ specifies countable concepts. Let $C_{ncp.c}$ be a set of countable concepts. The element $[[c_{nf} \ countable{-}concept] \ (0 : c_{ncp.c})]$ is called an order in $[\![c_{ncp.c}, c_{nf}]\!]$. Let $O_{rd.cncp.c}$ be a set of orders of countable concepts. An element $n_t :: cc :: c_{ncp.c}$ is called an instance in $[\![c_{ncp.c}]\!]$. An element $n_t :: cc :: c_{ncp.c}$ is an instance in $[\![c_{ncp.c}, c_{nf}]\!]$ if $n_t \leq o_{rd.cncp.c}[\![c_{ncp.c}, c_{nf}]\!]$.

The element $(x \ is \ countable{-}concept)$ specifying that x is a countable concept is defined as follows:

$(rule \ (x \ is \ countable{-}concept) \ var \ (x) \ abn \ then \ (x \ is \ countable{-}concept) :: value)$.

The element $n_t :: cc :: c_{ncp.c}$ is defined by the rule:

$(rule \ x :: cc :: y \ var \ (x, \ y) \ abn \ then \ x :: cc :: y :: value)$.

Let c_{ncpl} denote $(0 : x) :: countable{-}concept$. The element $(new \ x)$ called an instance generator generates a new instance of the countable concept x and adds this concept if it was not. It is defined as follows:

$(rule \ (new \ x) \ var \ (x) \ abn \ then \ (new \ x) :: atm)$;

$(transition \ (new \ x) :: atm \ var \ (x) \ then \ f_n)$,

where $(new \ x_0) :: atm, e_{l.*} \ \# \ c_{nf} \rightarrow_{f_n, s_b} (let \ w \ be \ c_{ncpl} \ in \ (if \ (w \ is \ int) \ then \ (seq \ (c_{ncpl} \ ::= \ (w + 1)), \ (let \ w1 \ be \ (w + 1) \ in \ w1 :: x :: cc)) \ else \ (seq \ (c_{ncpl} \ ::= \ 1), \ 1 :: x :: cc))), \ e_{l.*} \ \# \ c_{nf}$.

5.10. Matching operations

The conditional pattern matching element e_l of the form $(if \ x \ matches \ y \ var \ z \ seq \ u \ then \ v \ else \ w)$, where (y, z, u) is a pattern specification, is defined as follows:

$(rule\ (if\ x\ matches\ y\ var\ z\ seq\ u\ then\ v\ else\ w)\ var\ (x,\ y,\ z,\ u)\ seq\ (v,\ w)\ abn$
$where\ ((z\ is\ sequence)\ and\ (u\ is\ sequence)\ and\ (z\ includes :: set\ u))$
$then\ (if\ x\ matches\ y\ var\ z\ seq\ u\ then\ v\ else\ w) :: atm);$
$(transition\ (if\ x\ matches\ y\ var\ z\ seq\ u\ then\ v\ else\ w) :: atm$
$var\ (x,\ y,\ z,\ u,\ v,\ w)\ then\ f_n),$

where $(if\ x_0\ matches\ y_0\ var\ z_0\ seq\ u_0\ then\ v_0\ else\ w_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} [if\ [x_0$ is an instance in $[\![(y_0, z_0, u_0), m_t, s_{b.1}]\!]$ for some $s_{b.1}]\ then\ [subst\ s_{b.1} \cup (conf :: in : c_{nf}, val :: in : v_l[\![c_{nf}]\!])\ v_0]\ else\ [subst\ (conf :: in : c_{nf}, val :: in : v_l[\![c_{nf}]\!])\ w_0],\ e_{l.*}\ \#\ c_{nf}$. The objects x, y, z, u, v and w are called a matched element, pattern, variable specification, sequence variable specification, *then*-branch and *else*-branch in $[\![e_l]\!]$. The elements of z are called pattern variables in $[\![e_l]\!]$. The element e_l executes the instance of the *then*-branch v in $[\![s_{b.1}]\!]$ if x is an instance in $[\![y, s_{b.1}]\!]$. Otherwise, the element e_l executes the *else*-branch w.

Let $\{v_{r.*}\}$, $\{v_{r.s.*}\}$, $\{v_{r.*.1}\}$ and $\{v_{r.*.2}\}$ are pairwise disjoint, and $\{v_{r.*.3}\} \subseteq \{v_{r.*}\} \cup \{v_{r.*.1}\} \cup \{v_{r.*.2}\}$. The form $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.1})\ und\ (v_{r.*.2})\ val\ (v_{r.*.3})\ where\ c_{nd}\ then\ e_{l.1}\ else\ e_{l.2})$ is defined as follows:

- $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ where\ c_{nd}\ then\ e_{l.1}\ else\ e_{l.2})$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.1})\ und\ (v_{r.*.2})\ val\ (v_{r.*.3})\ then\ (if\ c_{nd}\ then\ e_{l.1}\ else\ e_{l.2} :: (nosubstexcept\ conf :: in, val :: in))\ else\ e_{l.2})$;
- $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3},\ v_r)\ then\ e_{l.1}\ else\ e_{l.2})$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ then\ (let\ w\ be\ v_r\ in\ [subst\ (v_r :: * : w)\ e_{l.1}])\ else\ e_{l.2})$, where w is a new element that does not occur in this definition;
- $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ ()\ then\ e_{l.1}\ else\ e_{l.2})$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ then\ e_{l.1}\ else\ e_{l.2})$;
- $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1},\ v_r)\ abn\ (v_{r.*.2})\ then\ b_d)$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ then\ (if\ (v_r\ is\ undefined)\ then\ und\ else\ e_{l.1})\ else\ e_{l.2})$;
- $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ ()\ abn\ (v_{r.*.2})\ then\ e_{l.1}\ else\ e_{l.2})$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2})\ then\ e_{l.1}\ else\ e_{l.2})$;
- $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2},\ v_r)\ then\ e_{l.1}\ else\ e_{l.2})$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ (v_{r.*.2})\ then\ (if\ (v_r\ is\ abnormal)\ then\ v_r\ else$

$e_{l.1})$ *else* $e_{l.2}$);

- $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ abn\ ()\ then\ e_{l.1}\ else\ e_{l.2})$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ then\ e_{l.1}\ else\ e_{l.2})$.

The element c_{nd} specifies the restriction on the values of the pattern variables. The undefined value is propagated through the variables of $v_{r.*.1}$. Abnormal values are propagated through the variables of $v_{r.*.2}$. The special element $v_r :: *$ references to the value of element associated with the pattern variable v_r. A pattern variable is evaluated if the element associated with it is evaluated. Thus, the sequence $v_{r.*.3}$ contains evaluated pattern variables. A pattern variable is quoted if the element associated with it is not evaluated.

The objects $var\ (v_{r.*})$, $seq\ (v_{r.s.*})$, $und\ (v_{r.*.1})$, $abn\ (v_{r.*.2})$, $val\ (v_{r.*.3})$, $where\ c_{nd}$ and $else\ e_{l.2}$ in this form can be omitted. The omitted objects correspond to $var\ ()$, $seq\ ()$, $und\ ()$, $abn\ ()$, $val\ ()$, $where\ true$ and $else\ skip$, respectively.

The form $(e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ where\ c_{nd})$ is a shortcut for $(if\ e_l\ matches\ p_t\ var\ (v_{r.*})\ seq\ (v_{r.s.*})\ und\ (v_{r.*.1})\ abn\ (v_{r.*.2})\ val\ (v_{r.*.3})\ where\ c_{nd}\ then\ true\ else\ und)$. The objects $var\ (v_{r.*})$, $seq\ (v_{r.s.*})$, $und\ (v_{r.*.1})$, $abn\ (v_{r.*.2})$, $val\ (v_{r.*.3})$ and $where\ c_{nd}$ in this form can be omitted. The omitted objects correspond to $var\ ()$, $seq\ ()$, $und\ ()$, $abn\ ()$, $val\ ()$ and $where\ true$, respectively.

5.11. Interpretations operations

The element $(x\ is\ definition{-}form)$ specifying that x is a definition form is defined as follows:

$(rule\ (x\ is\ definition{-}form)\ var\ (x)\ abn\ then\ (x\ is\ definition{-}form) :: value)$;

$(transition\ (x\ is\ definition{-}form)\ var\ (x)\ then\ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in F_{rm.d}]\ then\ true\ else\ und]$.

The element $f_{rm.d} :: name :: n_m$ specifying a definition with the name n_m is defined as follows:

$(rule\ x :: name :: y\ var\ (x,\ y)\ abn\ where\ (x\ is\ definition{-}form)$

$then\ x :: name :: y :: atm :: definition)$;

$(transition\ x :: name :: y :: atm :: definition\ var\ (x,\ y)\ then\ f_n)$,

where

- if $y_0 \in [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, then $x_0 :: name :: y_0 :: atm :: definition,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$;

- if $y_0 \notin [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, and x_0 is reduced to d_f, then $x_0 :: name :: y_0 :: atm :: definition,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [c_{nf}\ interpretation.(0 : definitions).y_0 : d_f]$.

The element $(add{-}interpretation\ x)$ adding the interpretation with the name x is defined as follows:

$(rule\ (add{-}interpretation\ x)\ var\ (x)\ abn\ then\ (add{-}interpretation\ x) :: atm);$

$(transition\ (add{-}interpretation\ x) :: atm\ var\ (x)\ then\ f_n),$

where

- if $x_0 \in [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, then $(add{-}interpretation\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [c_{nf}\ interpretation.(0 : order) : [value\ [[c_{nf}\ (0 : order) :: state :: interpretation] :: q\ +. :: set\ x_0 :: q]\ c_{nf}]]$;
- if $x_0 \notin [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, then $(add{-}interpretation\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$.

The element $(add{-}interpretation\ x\ after\ y)$ adding the interpretation with the name x after the interpretation with the name y is defined as follows:

$(rule\ (add{-}interpretation\ x\ after\ y)\ var\ (x,\ y)\ abn$

$then\ (add{-}interpretation\ x\ after\ y) :: atm);$

$(transition\ (add{-}interpretation\ x\ after\ y) :: atm\ var\ (x,\ y)\ then\ f_n),$

where

- if $x_0 \in [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, and $y_0 \notin [c_{nf}\ (0 : order) :: state :: interpretation] :: q\ -. :: set\ x_0]$, then $(add{-}interpretation\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$;
- if $x_0 \in [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, and $[value\ [c_{nf}\ (0 : order) :: state :: interpretation] :: q\ -. :: set\ x_0] = n_{m.*.1}\ y_0\ n_{m.*.2}$, then $(add{-}interpretation\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [c_{nf}\ interpretation.(0 : order) : n_{m.*.1}\ y_0\ x_0\ n_{m.*.2}]$;
- if $x_0 \notin [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, then $(add{-}interpretation\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$.

The element $(delete{-}interpretation\ x)$ deleting the interpretation with the name x is defined as follows:

$(rule\ (delete{-}interpretation\ x)\ var\ (x)\ abn\ then\ (delete{-}interpretation\ x) :: atm);$

$(transition\ (delete{-}interpretation\ x) :: atm\ var\ (x)\ then\ f_n),$

where

- if $x_0 \in [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, then $(delete{-}interpretation\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [c_{nf}\ interpretation.(0 : order) : [value\ [c_{nf}\ (0 : order) :: state :: transition] :: q\ -\ . :: set\ x_0 :: q\ c_{nf}]]$;
- if $x_0 \notin [support\ [c_{nf}\ (0 : definitions) :: state :: interpretation]] \cup [support\ i_{ntr.a.s}]$, then $(delete{-}interpretation\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$.

5.12. Configurations operations

The element $conf :: cur$ specifying the current configuration is defined as follows:

$(rule\ \ conf :: cur\ \ abn\ \ then\ \ conf :: cur :: value)$.

The element $val :: cur$ specifying the value in the current configuration is defined as follows:

$(rule\ \ val :: cur\ \ abn\ \ then\ \ val :: cur :: value)$;

$(definition\ \ val :: cur\ \ then\ \ val :: cur :: value)$;

$(interpretation\ \ val :: cur\ \ then\ \ f_n)$,

where $[f_n\ s_b] = v_l[\![c_{nf}]\!]$.

5.13. Transitions operations

The element $(x\ is\ rule{-}form)$ specifying that x is a rule form is defined as follows:

$(rule\ \ (x\ \ is\ \ rule{-}form)\ \ var\ \ (x)\ \ abn\ \ then\ \ (x\ \ is\ \ rule{-}form) :: value)$;

$(transition\ \ (x\ \ is\ \ rule{-}form)\ \ var\ \ (x)\ \ then\ \ f_n)$,

where $[f_n\ s_b] = [if\ [x_0 \in F_{rm.r}]\ then\ true\ else\ und]$.

The element $f_{rm.r} :: name :: n_m$ specifying a rule with the name n_m is defined as follows:

$(rule\ \ x :: name :: y\ \ var\ \ (x,\ \ y)\ \ abn\ \ where\ \ (x\ \ is\ \ rule{-}form)$

$then\ \ x :: name :: y :: atm :: rule)$;

$(transition\ \ x :: name :: y :: atm :: rule\ \ var\ \ (x,\ \ y)\ \ then\ \ f_n)$,

where

- if $y_0 \in [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}] \cup [support\ t_{rn.rlt.en.s}]$, then $x_0 :: name :: y_0 :: atm :: rule,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$;
- if $y_0 \notin [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}] \cup [support\ t_{rn.rlt.en.s}]$, and x_0 is reduced to r_l, then $x_0 :: name :: y_0 :: atm :: rule,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [c_{nf}\ transition.(0 : rules).y_0 : r_l]$.

The element $(add{-}transition\ x)$ adding the transition with the name x is defined as follows:

$(rule\ (add{-}transition\ x)\ var\ (x)\ abn\ then\ (add{-}transition\ x) :: atm);$
$(transition\ (add{-}transition\ x) :: atm\ var\ (x)\ then\ f_n),$

where

- if $x_0 \in [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}]$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*} \# c_{nf} \to_{f_n,s_b} e_{l.*} \# [c_{nf}\ transition.(-1 : exogenous, 0 : order) : [value\ [[c_{nf}\ (-1 : exogenous, 0 : order) :: state :: transition] :: q\ +. :: set\ x_0 :: q]\ c_{nf}]]$;
- if $x_0 \in [support\ t_{rn.rlt.en.s}]$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*} \# c_{nf} \to_{f_n,s_b} e_{l.*} \# [c_{nf}\ transition.(-1 : endogenous, 0 : order) : [value\ [[c_{nf}\ (-1 : endogenous, 0 : order) :: state :: transition] :: q\ +. :: set\ x_0 :: q]\ c_{nf}]]$;
- if $x_0 \notin [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}] \cup [support\ t_{rn.rlt.en.s}]$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*} \# c_{nf} \to_{f_n,s_b} e_{l.*} \# und \# c_{nf}$.

The element $(add{-}transition\ x\ after\ y)$ adding the transition with the name x after the transition with the name y is defined as follows:

$(rule\ (add{-}transition\ x\ after\ y)\ var\ (x,\ y)\ abn$
$then\ (add{-}transition\ x\ after\ y) :: atm);$
$(transition\ (add{-}transition\ x\ after\ y) :: atm\ var\ (x,\ y)\ then\ f_n),$

where

- if $x_0 \in [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}]$, and $y_0 \notin [c_{nf}\ (-1 : exogenous, 0 : order) :: state :: transition] :: q\ -. :: set\ x_0]$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*} \# c_{nf} \to_{f_n,s_b} e_{l.*} \# und \# c_{nf}$;
- if $x_0 \in [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}]$, and $[value\ [c_{nf}\ (-1 : exogenous, 0 : order) :: state :: transition] :: q\ -. :: set\ x_0] = n_{m.*.1}\ y_0\ n_{m.*.2}$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*} \# c_{nf} \to_{f_n,s_b} e_{l.*} \# [c_{nf}\ transition.(-1 : exogenous, 0 : order) : n_{m.*.1}\ y_0\ x_0\ n_{m.*.2}]$;
- if $x_0 \in [support\ t_{rn.rlt.en.s}]$, and $y_0 \notin [c_{nf}\ (-1 : endogenous, 0 : order) :: state :: transition] :: q\ -. :: set\ x_0]$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*} \# c_{nf} \to_{f_n,s_b} e_{l.*} \# und \# c_{nf}$;
- if $x_0 \in [support\ t_{rn.rlt.en.s}]$, and $[value\ [c_{nf}\ (-1 : endogenous, 0 : order) :: state :: transition] :: q\ -. :: set\ x_0] = n_{m.*.1}\ y_0\ n_{m.*.2}$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*} \# c_{nf} \to_{f_n,s_b} e_{l.*} \# [c_{nf}\ transition.(-1 : endogenous, 0 : order) : n_{m.*.1}\ y_0\ x_0\ n_{m.*.2}]$;
- if $x_0 \notin [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}] \cup [support$

$t_{rn.rlt.en.s}]$, then $(add{-}transition\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$.

The element $(delete{-}transition\ x)$ deleting the transition with the name x is defined as follows:

$(rule\ (delete{-}transition\ x)\ var\ (x)\ abn\ then\ (delete{-}transition\ x) :: atm)$;
$(transition\ (delete{-}transition\ x) :: atm\ var\ (x)\ then\ f_n)$,

where

- if $x_0 \in [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}]$, then $(delete{-}transition\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [c_{nf}\ transition.(-1 : exogenous, 0 : order) : [value\ [c_{nf}\ (-1 : exogenous, 0 : order) :: state :: transition] :: q\ -. :: set\ x_0 :: q\ c_{nf}]]$;
- if $x_0 \in [support\ t_{rn.rlt.en.s}]$, then $(delete{-}transition\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [c_{nf}\ transition.(-1 : endogenous, 0 : order) : [value\ [c_{nf}\ (-1 : endogenous, 0 : order) :: state :: transition] :: q\ -. :: set\ x_0 :: q\ c_{nf}]]$;
- if $x_0 \notin [support\ [c_{nf}\ (0 : rules) :: state :: transition]] \cup [support\ t_{rn.rlt.ex.s}] \cup [support\ t_{rn.rlt.en.s}]$, then $(delete{-}transition\ x_0) :: atm,\ e_{l.*}\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ und\ \#\ c_{nf}$.

The element e_l of the form $(modify\ x)$ or $(modify :: n\ x)$ is defined as follows:

$(rule\ (modify\ x)\ var\ (x)\ then\ (modify\ x) :: atm)$;
$(rule\ (modify :: n\ x)\ var\ (x)\ abn\ then\ (modify\ x) :: atm)$;
$(transition\ (modify\ x) :: atm\ var\ (x)\ then\ f_n)$,

where $(modify\ x_0) :: atm, e_{l.*}\ \#\ v_l\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [if\ [$there exists $c_{nf.1}$ such that $[value\ [subst\ (conf :: in : c_{nf}, val :: in : v_l[\![c_{nf}]\!], conf :: out : c_{nf.1}, val :: out : v_l[\![c_{nf.1}]\!])\ x_0]\ c_{nf}] \neq und]\ then\ v_l\ \#\ c_{nf.1}\ else\ und\ \#\ c_{nf}]$. The element x is called a transition condition in $[\![e_l]\!]$. It specifies the set of configurations reachable from c_{nf} for one transition. The elements $conf :: in$ and $conf :: out$ reference to the input state and the output state, and the elements $val :: in$ and $val :: out$ reference to values in these states.

⊕ The execution of the element $(modify\ (((-1 : value,\ 0 : x,\ 1 : variable)\ inconf :: out) = 0))$ initiates the transition to a state in which the value of the variable x equals to 0.

⊕ The execution of the element $(modify\ (((-1 : value,\ 0 : x,\ 1 : variable) = "green")\ and\ (((-1 : value,\ 0 : x,\ 1 : variable)\ in\ conf :: out) = "red")))$ initiates the transition from a state in which the value of the variable x equals to "green" to a state in which the variable x equals to "red".

The element e_l of the form $(modify{-}exist\ (x)\ y)$ or $(modify{-}exist :: n\ (x)\ y)$ is defined as follows:

$(rule\ (modify{-}exist\ (x)\ y)\ var\ (y)\ seq\ (x)\ then\ (modify{-}exist\ (x)\ y) :: atm)$;

$(rule\ (modify{-}exist :: n\ (x)\ y)\ var\ (y)\ seq\ (x)\ abn\ then\ (modify{-}exist\ (x)\ y) :: atm)$;

$(transition\ (modify{-}exist\ (x)\ y) :: atm\ var\ (y)\ seq\ (x)\ then\ f_n)$,

where $(modify{-}exist\ (x_0)\ y_0) :: atm, e_{l.*}\ \#\ v_l\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [if$ [there exists $c_{nf.1}$ such that $[[subst\ (conf :: in : c_{nf}, val :: in : v_l[\![c_{nf}]\!], conf :: out : c_{nf.1}, val :: out : v_l[\![c_{nf.1}]\!])\ y_0]$ is satisfiable in $((x_0), c_{nf})]\ then\ v_l\ \#\ c_{nf.1}\ else\ und\ \#\ c_{nf}]$. The element y is called a transition condition in $[\![e_l]\!]$. The elements of x are called existential variables in $[\![e_l]\!]$.

5.14. Safety operations

The element e_l of the form $(assert\ x)$ or $(assert :: n\ x)$ is defined as follows:

$(rule\ (assert\ x)\ var\ (x)\ then\ (assert\ x) :: atm)$;

$(rule\ (assert\ x :: n)\ var\ (x)\ abn\ then\ (assert\ x) :: atm)$;

$(transition\ (assert\ x) :: atm\ var\ (x)\ then\ f_n)$,

where $(assert\ x_0) :: atm, e_{l.*}\ \#\ v_l\ \#\ c_{nf} \rightarrow_{f_n,s_b} e_{l.*}\ \#\ [if\ [[value\ [subst\ (conf :: inc_{nf}, val :: in : v_l)\ x_0]\ c_{nf}]\ \neq\ und]\ then\ v_l\ else\ und]\ \#\ c_{nf}$. The element x is called a safety condition in $[\![e_l]\!]$.

5.15. Branching operations

The element e_l of the form $(branching\ x)$ is defined as follows:

$(rule\ (branching\ x)\ seq\ (x)\ abn\ then\ (branching\ x) :: atm)$;

$(transition\ (branching\ x) :: atm\ var\ (x)\ then\ f_n)$,

where $(branching\ x_0) :: atm, e_{l.*}\ \#\ v_l\ \#\ c_{nf} \rightarrow_{f_n,s_b} \#\ (type : assume) :: exc\ \#\ [c_{nf}\ branching.(0 : ()) : [((x_0),\ c_{nf},\ (e_{l.*}))\ .+\ [[c_{nf}\ branching]\ (0 : ())]]]$. The elements of x are called branches in $[\![e_l]\!]$. The element e_l generates the branchpoint with the branches x. The exception $(type : assume) :: exc$ specifies the failure of the execution of the current branch. The substate $branching$ contains information about branching. The conceptual $(0 : ()) :: state :: branching$ specifies the current sequence of branchpoints.

The endogenous transition relation specifying branching is defined as follows:

$(endogenous{-}transition\ f_n) :: name :: branching$

where

- if $[[c_{nf}\ branching]\ (0 : ())] = (((e_{l.*.1},\ e_l,\ e_{l.*.2}),\ c_{nf.1},\ (e_{l.*.3})), e_{l.*})$, then $\#\ (type :$

$assume) :: exc \ \# \ c_{nf} \rightarrow_{branching} e_{l.*.3} \ \# \ [c_{nf.1} \ branching.(0 : ()) : (((e_{l.*.1}, \ e_{l.*.2}), \ c_{nf.1}, \ (e_{l.*.3})), e_{l.*})]$;

- if $[[c_{nf} \ branching] \ (0 : ())] = (((), \ c_{nf.1}, \ (e_{l.*.3})), e_{l.*})$, then $\# \ (type : assume) :: exc \ \# \ c_{nf} \rightarrow_{branching} \# \ (type : assume) :: exc \ \# \ [c_{nf.1} \ branching.(0 : ()) : (e_{l.*})]$.

The element e_l of the form $(assume \ x)$ or $(assume :: n \ x)$ is defined as follows:

$(rule \ (assume \ x) \ var \ (x) \ then \ (assume \ x) :: atm)$;

$(rule \ (assume :: n \ x) \ var \ (x) \ abn \ then \ (assume \ x) :: atm)$;

$(transition \ (assume \ x) :: atm \ var \ (x) \ then \ f_n)$,

where $(assume \ x_0) :: atm, e_{l.*} \ \# \ v_l \ \# \ c_{nf} \rightarrow_{f_n, s_b} [if \ [[value \ [subst \ (conf :: in : c_{nf}, val :: in : v_l[\![c_{nf}]\!]) \ x_0] \ c_{nf}] \neq und] \ then \ e_{l.*} \ \# \ v_l \ else \ \# \ (type : assume) :: exc] \ \# \ c_{nf}$. The element x is called a continuation condition in $[\![e_l]\!]$. The violation of this condition initiates the failure of the execution of the current branch.

The element e_l of the form $(assume{-}exist \ (x) \ y)$ or $(assume{-}exist :: n \ (x) \ y)$ is defined as follows:

$(rule \ (assume{-}exist \ (x) \ y) \ var \ (y) \ seq \ (x) \ then \ (assume{-}exist \ x) :: atm)$;

$(rule \ (assume{-}exist :: n \ (x) \ y) \ var \ (y) \ seq \ (x) \ abn \ then \ (assume{-}exist \ x) :: atm)$;

$(transition \ (assume{-}exist \ (x) \ y) :: atm \ var \ (y) \ seq \ (x) \ then \ f_n)$,

where $(assume \ (x_0) \ y_0) :: atm, e_{l.*} \ \# \ v_l \ \# \ c_{nf} \rightarrow_{f_n, s_b} [if \ [[subst \ (conf :: in : c_{nf}, val :: in : v_l[\![c_{nf}]\!]) \ y_0]$ is satisfiable in $[\![(x_0), c_{nf}]\!]] \ then \ e_{l.*} \ \# \ v_l \ else \ \# \ (type : assume) :: exc] \ \# \ c_{nf}$. The element y is called a continuation condition in $[\![e_l]\!]$. The elements of x are called existential variables in $[\![e_l]\!]$.

6. Justification of requirements for conceptual transition systems

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

8. *The formalism must have language support. The language associated with the formalism must define syntactic representations of models of states, state objects, queries, query objects, answers and answer objects and includes the set of predefined basic query models.* The CTSL language associated with CTSs defines syntactic representations of models of states, state objects, queries, query objects, answers and answer objects and includes the set of predefined basic query models.
9. *It must model the change of the conceptual structure of states and state objects of the ITS.* The change of the conceptual structure of the ITS is described by the transition relation

on conceptual configurations specifying conceptual structures of the ITS with different sets of ontological elements.

10. *It must model the change of the content of the conceptual structure.* The change of the content of the conceptual structure of the ITS is described by the transition relation on conceptual states specifying the same conceptual structure of the ITS. In fact, the distinction between requirements 9 and 10 is relative, for conceptuals allow to define classifications of ontological elements with different granularity.
11. *It must model the transition relations of the ITS.* The transition relations of the ITS are modelled by the transition relation $t_{rn.rlt}$ of the CTS.
12. *The model of the exogenous transition relation must be extensible.* The model of the exogenous transition relation of the IQS is extended by addition of trnasition rules.

Thus, the additional requirements are met for CTSs.

7. Conclusion

In the paper two formalisms (ITSs and CTSs) for abstract unified modelling of the artifacts of the conceptual design of information systems have been proposed by ontological elements with arbitrary conceptual granularity. The basic definitions of the theory of CTSs have been given. The language of CTSs has been defined.

We plan to use CTSs to design and prototype software systems as well as to specify operational and axiomatic semantics of programming languages. In the case of operational semantics of a programming language, CTSs model an abstract machine of the language. In the case of axiomatic semantics of a programming language, CTSs model a verification conditions generator for programs in the language.

References

1. Sokolowski J., Banks C. Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains. Wiley, 2010.
2. Chen P. Entity-relationship modeling: historical events, future trends, and lessons learned // Software pioneers. Springer-Verlag New York, 2002. P. 296-310.
3. Anureev I.S. Formalisms for conceptual design of closed information systems // System Informatics. 2016. N 7. P. 69-148.
4. 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.

5. Gurevich Y. Evolving Algebras. Lipari Guide // Specification and Validation Methods. Oxford University Press, 1995. P. 9-36.