INTERNATIONAL STANDARD

ISO 18629-41

First edition
2006-08-15

Industrial automation systems and integration — Process specification language —

Part 41: **Definitional extension: Activity extensions**

Systèmes d'automatisation industrielle et intégration — Langage de spécification de procédé —

Partie 41: Extension de définition: Extensions d'activité

Reference number
ISO 18629-41:2006(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Figures

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 18629-41 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

A complete list of parts of ISO 18629 is available from the Internet:

http://www.tc184-sc4.org/titles

Introduction

ISO 18629 is an International Standard for the computer-interpretable exchange of information related to manufacturing processes. Taken together, all the parts contained in ISO 18629 provide a generic language for describing a manufacturing process throughout the entire production process within the same industrial company or across several industrial sectors or companies, independently from any particular representation model. The nature of this language makes it suitable for sharing process specifications and properties related to manufacturing during all the stages of a production process.

This part of ISO 18629 provides a description of the definitional extensions of the language related to activity extensions defined within ISO 18629.

All parts of ISO 18629 are independent of any specific process representation model used in a given application. Collectively, they provide a structural framework for improving the interoperability of these applications.

Industrial automation systems and integration — Process specification language —

Part 41: Definitional extension: Activity extensions

1 Scope

This part of ISO 18629 provides a specification of non-primitive concepts of the language, using a set of definitions written in the language of ISO 18629. These definitions provide an axiomatization of the semantics for terminology in this part of ISO 18629.

The following is within the scope of this part of ISO 18629:

— definitions of concepts specified using concepts of ISO 18629-11 and ISO 18629-12 that are independent of state and time relations, and related to classes of activity such as those listed in Clause 5.

The following is outside the scope of this part of ISO 18629:

— definitions of state and time-related concepts specified using concepts of ISO 18629-11 and ISO 18629-12.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 8824-1, *Information technology - Abstract Syntax Notation One (ASN.1) - Part 1: Specification of basic notation*

ISO 10303-1, *Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles*

ISO 15531-1, *Industrial automation systems and integration - Industrial manufacturing management data - Part 1: General overview*

ISO 18629-1: 2004, *Industrial automation systems and integration – Process specification language – Part 1 : Overview and basic principles*

ISO 18629-11: 2005, *Industrial automation systems and integration – Process specification language – Part 11 : PSL core*

ISO 18629-12: 2005, *Industrial automation systems and integration – Process specification language – Part 12: Outer core*

3 Terms, definitions, and abbreviations

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply:

3.1.1
axiom
well-formed formula in a formal language that provides constraints on the interpretation of symbols in the lexicon of a language

[ISO 18629-1]

3.1.2
data
representation of information in a formal manner suitable for communication, interpretation, or processing by human beings or computers

[ISO 10303-1]

3.1.3
defined lexicon
set of symbols in the non-logical lexicon which denote defined concepts

NOTE Defined lexicon is divided into constant, function and relation symbols.

EXAMPLE terms with conservative definitions.

[ISO 18629-1]

3.1.4
definitional extension
extension of PSL-Core that introduces new linguistic items which can be completely defined in terms of the PSL-Core

NOTE: Definitional extensions add no new expressive power to PSL-Core but are used to specify the semantics and terminology in the domain application.

[ISO 18629-1]

3.1.5
discrete manufacturing
production of discrete items

EXAMPLE Cars, appliances or computer.

[ISO 15531-1]

3.1.6
extension
augmentation of PSL-Core containing additional axioms

NOTE 1 The PSL-Core is a relatively simple set of axioms that is adequate for expressing a wide range of basic processes. However, more complex processes require expressive resources that exceed those of the PSL-Core. Rather than clutter the PSL-Core itself with every conceivable concept that might prove useful in describing one process or another, a variety of separate, modular extensions need to be developed and added to the PSL-Core as necessary. In this way a user can tailor the language precisely to suit his or her expressive needs.

NOTE 2 All extensions are core theories or definitional extensions.

[ISO 18629-1]

3.1.7
grammar
specification of how logical symbols and lexical terms can be combined to make well-formed formulae

[ISO 18629-1]

3.1.8
information
facts, concepts, or instructions

[ISO 10303-1]

3.1.9
language
combination of a lexicon and a grammar

[ISO 18629-1]

3.1.10
manufacturing
function or act of converting or transforming material from raw material or semi-finished state to a state of further completion

[ISO 15531-1]

3.1.11
manufacturing process

structured set of activities or operations performed upon material to convert it from the raw material or a semifinished state to a state of further completion

NOTE Manufacturing processes may be arranged in process layout, product layout, cellular layout or fixed position layout. Manufacturing processes may be planned to support make-to-stock, make-to-order, assemble-to-order, etc., based on strategic use and placements of inventories.

[ISO 15531-1]

3.1.12
model
combination of a set of elements and a truth assignment that satisfies all well-formed formulae in a theory

NOTE 1 The word "model" is used, in logic, in a way that differs from the way it is used in most scientific and everyday contexts : if a sentence is true in a certain interpretation, it is possible to say that the interpretation is a model of the sentence. The kind of semantics presented here is often called model-theoretical semantics.

NOTE 2 A model is typically represented as a set with some additional structure (partial ordering, lattice, or vector space). The model then defines meanings for the terminology and a notion of truth for sentences of the language in terms of this model. Given a model, the underlying set of axioms of the mathematical structures used in the set of axioms then becomes available as a basis for reasoning about the concepts intended by the terms of the language and their logical relationships, so that the set of models constitutes the formal semantics of the ontology.

[ISO 18629-1]

3.1.13
ontology
lexicon of specialised terminology along with some specification of the meaning of terms in the lexicon

NOTE 1: structured set of related terms given with a specification of the meaning of the terms in a formal language. The specification of meaning explains why and how the terms are related and conditions how the set is partitioned and structured.

NOTE 2: The primary component of a process specification language such as ISO 18629 is an ontology. The primitive concepts in the ontology according to ISO 18629 are adequate for describing basic manufacturing, engineering, and business processes.

NOTE 3: The focus of an ontology is not only on terms, but also on their meaning. An arbitrary set of terms is included in the ontology, but these terms can only be shared if there is an agreement about their meaning. It is the intended semantics of the terms that is being shared, not simply the terms.

NOTE 4: Any term used without an explicit definition is a possible source of ambiguity and confusion. The challenge for an ontology is that a framework is needed for making explicit the meaning of the terms within it. For the ISO 18629 ontology, it is necessary to provide a rigorous mathematical characterisation of process information as well as a precise expression of the basic logical properties of that information in the ISO 18629 language.

NOTE 5 In practice, extensions incorporate the axioms of the Outer Core.

[ISO 18629-1]

3.1.14
primitive concept
lexical term that has no conservative definition

[ISO 18629-1]

3.1.15
primitive lexicon
set of symbols in the non-logical lexicon which denote primitive concepts

NOTE Primitive lexicon is divided into constant, function and relation symbols.

[ISO 18629-1]

3.1.16
process
structured set of activities involving various enterprise entities, that is designed and organised for a given purpose

NOTE The definition provided here is very close to that given in ISO 10303-49. Nevertheless ISO 15531 needs the notion of structured set of activities, without any predefined reference to the time or steps. In addition, from the point of view of flow management, some empty processes may be needed for a synchronisation purpose although they are not actually doing anything (ghost task).

[ISO 15531-1]

3.1.17
product
a thing or substance produced by a natural or artificial process

[ISO 10303-1]

3.1.18
resource
any device, tool and means, excepted raw material and final product components, at the disposal of the enterprise to produce goods or services

NOTE 1 Resources as they are defined here include human resources considered as specific means with a given capability and a given capacity. Those means are considered as being able to be involved in the manufacturing process through assigned tasks. That does not include any modelling of an individual or common behaviour of human resource excepted in their capability to perform a given task in the manufacturing process (e.g.: transformation of raw material or component, provision of logistic services). That means that human resources are only considered, as the other, from the point of view of their functions, their capabilities and their status (e.g.: idle, busy). That excludes any modelling or representation of any aspect of individual or common «social» behaviour.

NOTE 2 This definition includes ISO 10303-49 definition but is included in the definition that applies for ISO 18629-14 and ISO 18629-44 that includes raw materials and consumables.

[ISO 15531-1]

3.1.19
theory
set of axioms and definitions that pertain to a given concept or set of concepts

NOTE this definition reflects the approach of artificial intelligence in which a theory is the set of assumptions on which the meaning of the related concept is based.

[ISO 18629-1]

3.2 Abbreviations

KIF Knowledge Interchange Language.

4 General organization of parts 41 to 49 of ISO 18629[1]

Parts 41 to 49 of ISO 18629 specify definitional extensions needed to give precise definitions and a set of axioms for non-primitive concepts of ISO 18629. Definitional extensions are extensions of ISO 18629-11 and ISO 18629-12 that introduce new items for the lexicon. The items found in definitional extensions can be completely defined using theories of ISO 18629-11 and ISO 18629-12. The definitional extensions provide precise semantic definitions for elements used in the specification of individual applications or types of applications for the purpose of interoperability. Definitional extensions exist in the following categories:

— Activity Extensions;

— Temporal and State Extensions;

— Activity Ordering and Duration Extensions;

— Resource Roles;

— Resource Sets;

— Processor Activity Extensions.

Individual users or groups of users of ISO 18629 may need to extend ISO 18629 for specifying concepts that are currently absent in parts 41 to 49 of ISO 18629. They shall use the elements presented in ISO 18629 for doing so. User-defined extensions and their definitions constitute definitional extensions but shall not become part of parts 41 to 49 of ISO 18629.

Note: User-defined extensions must conform to ISO 18629 as defined in ISO 18629-1 : 2004, 5.1 and 5.2.

The following are within the scope of parts 41 to 49 of ISO 18629 :

— the semantic definitions, using concepts in ISO 18629-11 and ISO 18629-12, of elements that are specific to the six concepts outlined above;

[1] Certain parts are under development

- a set of axioms for constraining the use of elements in definitional extensions.

The following is outside the scope of parts 41 to 49 of ISO 18629 :

- definitions and axioms for concepts that are part of ISO 18629-11 and ISO 18629-12;
- elements that are not defined using the elements in ISO 18629-11 and ISO 18629-12;
- user-defined extensions.

5 Organization of this part of ISO 18629

The fundamental theories that constitute this part of ISO 18629 are:

- Non-deterministic Activities: Permuting Branch Structure;
- Non-deterministic Activities: Folding Branch Structure;
- Non-deterministic Activities: Branch Structure and Ordering;
- Non-deterministic Activities: Repetitive Branch Structure;
- Spectrum of Activities: Permuting Activity Trees;
- Spectrum of Activities: Compacting Branch Structure;
- Spectrum of Activities: Activity Trees and Re-ordering;
- Spectrum and Sub-tree Containment;
- Embedding Constraints for Activities;
- Skeletal Activity Trees;
- Atomic Activities: Upwards Concurrency;
- Atomic Activities: Downwards Concurrency;
- Spectrum for Atomic Activities;
- Preconditions for Activities.

Figure 1 shows the relationships between these extensions and the core theories of ISO 18629-12 that are useful for specifying definitional extensions in this part of ISO 18629. The arrows show dependencies between extensions in this part of ISO 18629 and ISO 18629-12. All theories in this part of ISO 18629 are extensions of ISO 18629-12, itself an extension of ISO 18629-11.

NOTE A complete diagram of dependencies between the extensions in ISO 18629-12 is found in ISO 18629-12 : 2005, 5.1.

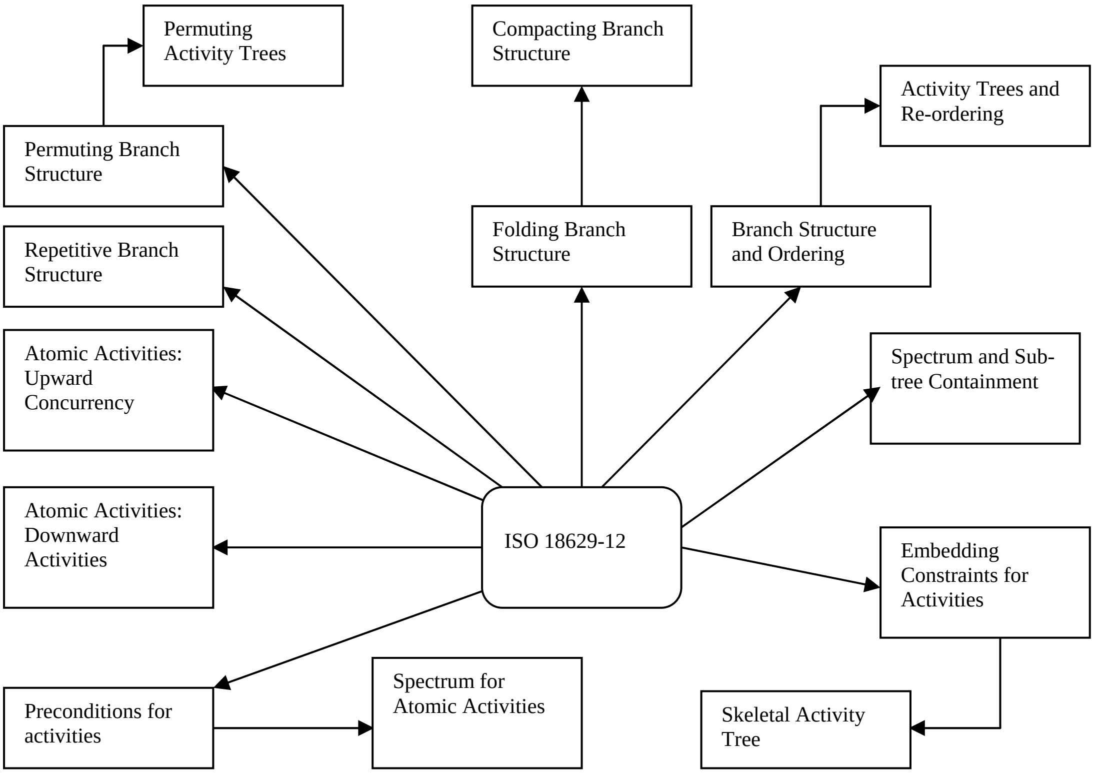

Figure 1: Definitional Extensions of ISO 18629-41

6 Non-deterministic activities: Permuting Branch Structure

This clause characterizes all definitions pertaining to non deterministic activities: Permuting Branch Structure.

6.1 Primitive lexicon of the Permuting Branch Structure

No primitive relations are required by the lexicon of Permuting Branch Structure.

6.2 Defined lexicon for concepts of Permuting Branch Structure

The following relations are defined in this clause:

(branch_monomorphic ?a ?occ1 ?occ2);

(branch_automorphic ?a ?occ1 ?occ2);

(permuted ?a);

(nondet_permuted ?a);

(partial_permuted ?a);

(simple ?a).

Each concept is described by informal semantics and a KIF axiom.

6.3 Core Theories required by Permuting Branch Structure

This extension requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

6.4 Definitional extensions required by Permuting Branch Structure

No definitional extensions are required by the Permuting Branch Structure.

6.5 Definitions of concepts for Permuting Branch Structure

The following concepts are defined for Permuting Branch Structure.

6.5.1 Branch_monomorphic

One branch of a minimal activity tree is branch-monomorphic to another if and only if the set of atomic activity occurrences in the one branch can be embedded into the set of atomic activity occurrences on the other branch.

```
(forall (?occ1 ?occ2) (iff (branch_monomorphic ?occ1 ?occ2)
(forall (?s1 ?a)
  (implies (and     (occurrence_of ?occ1 ?a)
              (subactivity_occurrence ?s1 ?occ1))
          (exists (?s2)
              (and(subactivity_occurrence ?s2 ?occ2)
```

```
(mono ?s1 ?s2 ?a))))))))
```

6.5.2 Branch_automorphic

Two branches of a minimal activity tree are branch-automorphic if and only if the set of atomic activity occurrences on one branch is a permutation of the set of atomic activity occurrences on the other branch.

```
(forall (?occ1 ?occ2) (iff (branch_automorphic ?occ1 ?occ2)
(and    (branch_monomorphic ?occ1 ?occ2)
  (branch_monomorphic ?occ2 ?occ1))))
```

6.5.3 Permuted

An activity occurrence is permuted if and only if the set of atomic subactivity occurrences on each branch of the minimal activity tree is a permutation of the atomic subactivity occurrences on every other branch.

```
(forall (?occ) (iff (permuted ?occ)
(forall (?occ1 ?occ2)
  (implies  (and      (same_grove ?occ1 ?occ)
            (same_grove ?occ2 ?occ))
        (branch_automorphic ?occ1 ?occ2))))))
```

6.5.4 Nondet_permuted

An activity occurrence is nondeterministically permuted if and only if the set of atomic subactivity occurrences on each branch of the minimal activity tree is a permutation of the atomic subactivity occurrences on some other branch.

```
(forall (?occ) (iff (nondet_permuted ?occ)
(forall (?occ1)
  (implies  (same_grove ?occ1 ?occ)
        (exists (?occ2)
          (and(same_grove ?occ1 ?occ2)
              (not (= ?occ1 ?occ2))
              (branch_automorphic ?occ1 ?occ2)))))))
```

6.5.5 Partial_permuted

An activity occurrence is partially permuted if and only if there exists a branch such that the set of atomic subactivity occurrences is a permutation of the atomic subactivity occurrences on some other branch, and there exists a branch that is not a permutation of any other branch.

```
(forall (?occ) (iff (partial_permuted ?occ)
(exist (?occ1 ?occ2 ?occ3)
  (and(same_grove ?occ1 ?occ)
      (same_grove ?occ2 ?occ)
      (not (= ?occ1 ?occ2))
      (branch_automorphic ?occ1 ?occ2)
      (same_grove ?occ3 ?occ)
      (forall (?occ4)
           (implies  (and      (same_grove ?occ3 ?occ4 ?a)
                      (branch_automorphic ?occ3 ?occ4))
                 (= ?occ3 ?occ4))))))))
```

6.5.6 Simple

An activity occurrence is simple if and only if none of the branches in an activity tree are branch automorphic.

```
(forall (?occ) (iff (simple ?occ)
(forall (?occ1 ?occ2)
  (implies  (and      (same_grove ?occ1 ?occ)
            (same_grove ?occ2 ?occ)
            (branch_automorphic ?occ1 ?occ2))
       (= ?occ1 ?occ2)))))
```


6.6 Grammar for relations of Permuting Branch Structure

The following grammar sentences describe process descriptions and auxiliary rules specified in KIF for Permuting Branch Structure.

Note: The function and importance of grammar sentences in ISO 18629 is explained in ISO 18629-1: 2004, 3.3.8, 4.2.4, and 5.1.

```
< permuted_spec > ::=   (forall (< variable >)
                        (implies  (same_tree < variable > ?occ)
                             < occurrence_sentence >))
< nondet_permuted_spec > ::=   (forall (< variable >*)
                        (implies  (same_tree < variable > ?occ)
                             (or < occurrence_sentence >*)))
< partial_permuted_spec > ::=     (or  < nondet_permuted_spec >
                          < simple_spec >)
< simple_spec > ::=       (exists (< variable >*)
                        (and< occurrence_formula >
                            < branch_spec >))
< occurrence_literal > ::=     (occurrence < variable > < term >)
                        (subactivity_occurrence < variable > < variable >)
< occurrence_formula > ::=  < occurrence_literal > |
                        (and < occurrence_literal >*)
< occurrence_sentence > :=  (exists (< variable >*)
                            < occurrence_formula >)
< branch_spec > ::=       (and(same_tree < variable > ?occ)+
                        (not (= < variable > ?occ))+)
```

7 Non-deterministic activities: Folding Branch Structure

This clause characterizes all definitions pertaining to Non-determininistic activities: Folding Branch Structure.

7.1 Primitive lexicon of Folding Branch Structure

No primitive relations are required by the lexicon of Folding Branch Structure.

7.2 Defined lexicon for concepts of Folding Branch Structure

The following relations are defined in this clause:

— (branch_homomorphic ?occ1 ?occ2);

— (folded ?a);

— (nondet_folded ?a);

— (partial_folded ?a);

— (rigid ?a);

Each concept is described by informal semantics and a KIF axiom.

7.3 Theories required by Folding Branch Structure

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

7.4 Definitional extensions required by Folding Branch Structure

No Definitional Extensions are required by Folding Branch Structure.

7.5 Definitions of Folding Branch Structure

The following concepts are defined for Folding Branch Structure.

7.5.1 Branch_homomorphic

One branch of a minimal activity tree is branch-homomorphic to another if and only if there exists a mapping of the set of atomic activity occurrences in the one branch into the set of atomic activity occurrences on the other branch.

```
(forall (?occ1 ?occ2) (iff branch_homomorphic ?occ1 ?occ2)
(and    (same_grove ?occ1 ?occ2)
     (forall (?s1 ?a)
          (implies  (and (occurrence_of ?occ1 ?a)
               (subactivity_occurrence ?s1 ?occ1))
            (exists (?s2)
                    (subactivity_occurrence ?s2 ?occ2)
                    (hom ?s1 ?s2 ?a)))))))
```

7.5.2 Folded

An activity is folded if and only if there exists a single branch such that the set of atomic subactivity occurrences on each branch of the minimal activity tree is a permutation of the atomic subactivity occurrences on this single branch.

```
(forall (?occ) (iff (folded ?occ)
(exists (?occ1)
  (and(same_grove ?occ1 ?occ)
      (forall (?occ2)
          (implies  (same_grove ?occ2 ?occ)
                (branch_homomorphic ?occ1 ?occ2)))))))
```

7.5.3 Nondet_folded

An activity is nondeterministically folded if and only if there exists a branch such that the set of atomic subactivity occurrences on each branch of the minimal activity tree is a permutation of the atomic subactivity occurrences on the first branch.

```
(forall (?occ) (iff (nondet_folded ?occ)
(forall (?occ1)
  (implies  (same_grove ?occ1 ?occ)
```

```
(exists (?occ2)
    (and(same_grove ?occ2 ?occ)
        (not (= ?occ1 ?occ2))
        (branch_homomorphic ?occ1 ?occ2)))))))
```

7.5.4 Partial_folded

An activity is partially folded if and only if there exists a branch such that the set of atomic subactivity occurrences on each branch of the minimal activity tree is a permutation of the atomic subactivity occurrences on the first branch.

```
(forall (?occ) (iff (partial_folded ?occ)
(exists (?occ1 ?occ2 ?occ3)
  (and(same_grove ?occ1 ?occ)
      (same_grove ?occ2 ?occ)
      (not (= ?occ1 ?occ2))
      (branch_homomorphic ?occ1 ?occ2)
      (same_grove ?occ3 ?occ)
      (forall (?occ4)
          (implies  (and      (same_grove ?occ4 ?occ)
                    (not (= ?occ3 ?occ4)))
                (not (branch_homomorphic ?occ3 ?occ4))))))))
```

7.5.5 Rigid

An activity is rigid if and only if none of the branches in an activity tree are branch homomorphic.

```
(forall (?occ) (iff (rigid ?occ)
(forall (?occ1 ?occ2)
  (implies  (and      (same_grove ?occ1 ?occ)
            (same_grove ?occ2 ?occ)
            (not (= ?occ1 ?occ2)))
```


(not (branch_homomorphic ?occ1 ?occ2))))))

7.6 Grammar for process descriptions of Folding Branch Structure

The following grammar sentences describe process descriptions and auxiliary rules specified in KIF for Folding Branch Structure.

< folded_spec > ::= (exists (< variable >)

(and(same_tree < variable > ?occ)

< occurrence_axiom >))

< nondet_folded_spec > ::= (exists (< variable >)

(and (same_tree < variable > ?occ)

(or < occurrence_axiom >*)))

< partial_folded_spec > ::= (or < nondet_folded_spec >

< rigid_spec >)

< rigid_spec > ::= (exists (< variable >*)

(and< occurrence_formula >

< branch_spec >))

< occurrence_disjunct > ::= < occurrence_literal > |

(or < occurrence_literal >*)

< occurrence_axiom > ::= (exists (< variable >*)

< occurrence_disjunct >)

8 Non-deterministic activities: Branch Structure and Ordering

This clause characterizes all definitions pertaining to Non-determininistic activities: Branch Structure and Ordering.

8.1 Primitive lexicon of Branch Structure and Ordering

No primitive relations are required by the lexicon of Branch Structure and Ordering.

8.2 Defined lexicon of Branch Structure and Ordering

The following relations are defined in this clause:

— (mono_tree ?s1 ?s2 ?a)

— (order_tree ?s1 ?s2 ?a)

— (root_automorphic ?occ1 ?occ2)

— (ordered ?occ)

— (nondet_ordered ?occ)

— (broken_ordered ?occ)

— (unordered ?occ)

Each concept is described by informal semantics and a KIF axiom.

8.3 Theories required by Branch Structure and Ordering

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

8.4 Definitional extensions required by Branch Structure and Ordering

No Definitional Extensions are required by Branch Structure and Ordering.

8.5 Definitions of Branch Structure and Ordering

The following concepts are defined for Branch Structure and Ordering.

8.5.1 Mono_tree

The subtree rooted in ?s1 can be monomorphically embedded into the subtree rooted in ?s2.

(forall (?s1 ?s2 ?a) (iff (mono_tree ?s1 ?s2 ?a)

(forall (?s3 ?s4 ?s5)

```
(implies (and (min_precedes ?s1 ?s3 ?a)
     (min_precedes ?s2 ?s4 ?a)
     (mono ?s3 ?s4 ?a)
     (min_precedes ?s3 ?s5 ?a))
   (exists (?s6)
     (and(mono ?s5 ?s6 ?a)
        (min_precedes ?s4 ?s6 ?a)))))))
```

8.5.2 Order_tree

This relation holds iff there exists an order automorphism that maps the subtree rooted in ?s1 to the subtree rooted in ?s2.

```
(forall (?s1 ?s2 ?a) (iff (order_tree ?s1 ?s2 ?a)
(and (mono_tree ?s1 ?s2 ?a)
  (forall (?s3 ?s4 ?s5 ?s6)
     (implies (and (min_precedes ?s1 ?s3 ?a)
          (min_precedes ?s2 ?s4 ?a)
          (cousin ?s3 ?s4 ?a)
          (min_precedes ?s3 ?s5 ?a)
          (cousin ?s5 ?s6 ?a))
        (iff(iso_occ ?s3 ?s5)
          (iso_occ ?s4 ?s6)))))))
```

8.5.3 Root_automorphic

Two activity occurrences in the same tree are root automorphic if and only if there exist atomic subactivity occurrences of each activity occurrence that are the roots of isomorphic subtrees.

```
(forall (?occ1 ?occ2) (iff (root_automorphic ?occ1 ?occ2)
(exists (?a ?s1 ?s2)
  (and(occurrence_of ?occ1 ?a)
     (occurrence_of ?occ2 ?a)
```



```
(subactivity_occurrence ?s1 ?occ1)
(subactivity_occurrence ?s2 ?occ2)
(order_tree ?s1 ?s2 ?a)
(order_tree ?s2 ?s1 ?a)))))
```

8.5.4 Ordered

An activity occurrence is ordered if and only if it is root automorphic with every other root subactivity occurrence of the same tree.

```
(forall (?occ1 ?occ2) (iff (root_automorphic ?occ1 ?occ2)
(exists (?a ?s1 ?s2)
  (and(occurrence_of ?occ1 ?a)
      (occurrence_of ?occ2 ?a)
      (subactivity_occurrence ?s1 ?occ1)
      (subactivity_occurrence ?s2 ?occ2)
      (order_tree ?s1 ?s2 ?a)
      (order_tree ?s2 ?s1 ?a)))))
```

8.5.5 Nondet_ordered

An activity occurrence is nondeterministically ordered if and only if each activity occurrence in the tree is root automorphic to some other root subactivity occurrence in the tree.

```
(forall (?occ1) (iff (nondet_ordered ?occ1)
(forall (?occ2)
  (implies  (same_grove ?occ1 ?occ2)
        (exists (?occ3)
            (and(same_grove ?occ1 ?occ3)
                  (not (= ?occ3 ?occ2))
                  (root_automorphic ?occ2 ?occ3)))))))
```


8.5.6 Broken_ordered

An activity is broken ordered if and only if there exist activity occurrences in the same tree that are root automorphic to some other root subactivity occurrences in the tree, and there also exist root subactivity occurrences that are not root automorphic to any other root subactivity occurrence.

```
(forall (?occ1) (iff (broken_ordered ?occ1)
(exists (?occ2)
  (and(same_grove ?occ1 ?occ2)
       (not (= ?occ1 ?occ2))
       (root_automorphic ?occ1 ?occ2)
       (forall (?occ3)
            (implies  (and      (same_grove ?occ3 ?occ1)
                      (not (= ?occ3 ?occ2))
                 (not (root_automorphic ?occ3 ?occ2))))))))
```

8.5.7 Unordered

An activity occurrence is unordered if and only if none of the activity occurrences in the same tree are root automorphic.

```
(forall (?occ1) (iff (unordered ?occ1)
(forall (?occ2)
  (implies  (and      (same_grove ?occ1 ?occ2)
            (not (= ?occ1 ?occ2)))
        (not (root_automorphic ?occ1 ?occ2))))))
```

8.6 Grammar for Branch Structure and Ordering

The following grammar sentences describe process descriptions and auxiliary rules specified in KIF for Branch Structure and Ordering.

```
< ordered_spec > ::=   (forall (< variable >)
                            (implies  (same_tree < variable > ?occ)
                                      < ordered_sentence >)) |
                       (forall (< variable >)
```

(implies (same_tree < variable > ?occ)

< ordered_formula >))

< nondet_ordered_spec > ::= (forall (< variable >)

(implies (same_tree < variable > ?occ)

(or < ordered_sentence >*)))

< broken_ordered_spec > ::= (forall (< variable >)

(implies (same_tree < variable > ?occ)

(or {< ordered_sentence >* |

< ordered_list >*})))

< unordered_spec > ::= (forall (< variable >)

(implies (same_tree < variable > ?occ)

(or < ordered_list >*)))

< ordered_literal > ::= (min_precedes < variable > < variable > < term >) |

(next_subocc < variable > < variable > < term >)

< ordered_list > ::= < ordered_literal > |

(and < ordered_literal >*)

< conditional_occurrence > ::= (implies < occurrence_formula >

< ordered_list >)

```
< ordered_sentence > ::=      (exists (< variable >)
                          (and(root_occ < variable > ?occ)
                              < occurrence_disjunct >
                              < conditional_occurrence >))))

< ordered_conjunct > ::=      < ordered_literal > |
                          (and < ordered_formula >*)

< ordered_formula > ::=           (exists (< variable >)
                              (and    (root_occ < variable > ?occ)
                                  < occurrence_disjunct >
                              < ordered_conjunct >)) |
                      (or < ordered_formula >*)
```

9 Non-deterministic activities: Repetitive Branch Structure

This clause characterizes all definitions pertaining to Non-deterministic activities: Repetive Branch Structure.

9.1 Primitive lexicon of Repetitive Branch Structure

No primitive relations are required by the lexicon of Repetitive Branch Structure.

9.2 Defined relations of Repetitive Branch Structure

The following relations are defined in this clause:

- (branch_mono ?s1 ?s2 ?s3 ?s4 ?a)
- (reptree ?s ?occ)
- (repetitive ?occ)
- (nondet_repetitive ?occ)
- (partial_repetitive ?occ)

— (amorphous ?occ)

Each concept is described by informal semantics and a KIF axiom.

9.3 Theories required by Repetitive Branch Structure

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

9.4 Definitional extensions required by Repetitive Branch Structure

No Definitional Extensions are required by the Repetitive Branch Structure.

9.5 Definitions of Repetitive Branch Structure

The following concepts are defined for Repetitive Branch Structure.

9.5.1 Branch_mono

The sub-branch of a minimal activity with initial atomic subactivity occurrence ?s1 and final atomic subactivity occurrence ?s2 is occurrence-isomorphic to the sub-branch with initial atomic subactivity occurrence ?s3 and final atomic subactivity occurrence ?s4.

```
(forall (?s1 ?s2 ?s3 ?s4 ?a) (iff (branch_mono ?s1 ?s2 ?s3 ?s4 ?a)
(forall (?s5 ?s6)
    (implies  (and  (min_precedes ?s1 ?s5 ?a)
                    (min_precedes ?s6 ?s2 ?a)
                    (next_subocc ?s5 ?s6 ?a))
          (exists (?s7 ?s8)
                    (and    (min_precedes ?s3 ?s7 ?a)
```

```
                    (min_precedes ?s8 ?s4 ?a)

                    (next_subocc ?s7 ?s8 ?a)

                    (iso_occ ?s5 ?s7)

                    (iso_occ ?s6 ?s8)))))))
```

9.5.2 Reptree

Every atomic subactivity occurrence in ?occ can be embedded in a branch of the subtree whose root occurrence is ?s.

```
(forall (?s ?occ) (iff (reptree ?s ?occ)

(forall (?s1 ?a)

  (implies  (and      (occurrence ?occ ?a)

              (subactivity_occurrence ?s1 ?occ))

         (exists (?s2 ?s3 ?s4 ?occ1)

              (and(same_tree ?o ?o1)

                   (leaf_occ ?s2 ?occ1)

                   (<_min ?s3 ?s1 ?a)

                   (<_min ?s1 ?s4 ?a)

                   (branch_mono ?s3 ?s4 ?s ?s2 ?a)))))))
```

9.5.3 Repetitive

An activity occurrence is repetitive if and only if every branch in the same activity tree can be embedded in a unique subtree. Intuitively, this is equivalent to the existence of a unique repeating subtree.

```
(forall (?occ) (iff (repetitive ?occ)

(exists (?s1 ?occ2)

  (and(same_tree ?occ ?occ2)

       (subactivity_occurrence ?s ?occ2)

       (forall (?occ1)
```

```
                (implies  (same_tree ?occ ?occ1)
                      (and     (reptree ?s ?occ1)
                          (not (root_occ ?s ?occ1))))))))
```

9.5.4 nondet_repetitive

An activity is nondeterministically repetitive if and only if every branch in the same activity tree can be embedded in some subtree. Intuitively, this is equivalent to the existence of multiple nonisomorphic repeating subtrees.

```
(forall (?occ) (iff (nondet_repetitive ?occ)
(forall (?occ1)
  (implies  (same_tree ?occ ?occ1)
        (exists (?s1 ?occ2)
            (and(same_tree ?occ ?occ2)
                 (subactivity_occurrence ?s ?occ2)
                 (reptree ?s ?occ1)
                 (not (root_occ ?s ?occ1))))))))
```

9.5.5 partial_repetitive

An activity occurrence is partially repetitive if and only if there exist repeating subtrees, but there also exist branches in the same tree that cannot be mapped to any repeating subtree.

```
(forall (?occ) (iff (partial_repetitive ?occ)
(and   (exists (?occ1 ?occ2 ?s1)
       (and       (same_tree ?occ ?occ1)
            (same_tree ?occ ?occ2)
            (subactivity_occurrence ?s ?occ2)
            (rep_tree ?s ?occ1)
            (not (root_occ ?s ?occ1))))
  (exists (?occ2)
       (forall (?s2)
```

```
(implies  (reptree ?s1 ?occ1)
          (root_occ ?s1 ?occ1)))))))
```

9.5.6 Amorphous

An activity occurrence is amorphous if and only if there are no repeating subtrees in the same activity tree.

```
(forall (?occ) (iff amorphous ?occ)
(forall (?occ1 ?s)
  (implies  (and     (same_tree ?occ1 ?occ)
                     (reptree ?s ?occ1))
                (root_occ ?s ?occ1)))))
```

9.6 Grammar for Repetitive Branch Structure

The following grammar sentences describe process descriptions and auxiliary rules specified in KIF for Repetitive Branch Structure.

```
< repetitive_spec > ::=   (forall (< term > ?s1 ?s2)
                (iff  (do < term > ?s1 ?s2)
                      < rep_formula >))


< nondet_rep_spec > ::=      (forall (< term > ?s1 ?s2)
                (iff  (do < term > ?s1 ?s2)
                      (or   < rep_formula >*)))


< partial_rep_spec > ::=  (forall (< term > ?s1 ?s2)
                      (iff    (do < term > ?s1 ?s2)
                      < partial_rep_formula >


< rep_formula > ::=   (exists (< term >)
          (and(subactivity < term > < term >)
```

```
                    (forall (?s3)
                      (implies (do < term > ?s1 ?s3)
                            (or  (= ?s2 ?s3)
                                (do < term > ?s3 ?s2))))))


< partial_rep_formula > ::=   (exists (< term > ?s3)
                    (and(subactivity < term > < term >)
                         (do < term > ?s1 ?s3)
                         (or  (= ?s2 ?s3)
                              (do < term > ?s3 ?s2))))
```

10 Spectrum of activities: Permuting Activity Trees

This clause characterizes all definitions pertaining to Spectrum of Acitivities: Permuting Activity Tree.

10.1 Primitive lexicon of Permuting Activity Trees

No primitive relations are required by the lexicon of Permuting Activity Trees.

10.2 Defined relations of Permuting Activity Trees

The following relations are defined in this clause:

— (reordered ?a)

— (partial_reordered ?a)

— (nondet_reorder ?a)

— (nonorderable ?a)

Each concept is described by informal semantics and a KIF axiom.

10.3 Theories required by Permuting Activity Trees

This theory requires:

— act_occ.th;

- complex.th;
- atomic.th;
- subactivity.th;
- occtree.th;
- psl_core.th.

10.4 Definitional extensions required by Permuting Activity Trees

This extension requires Permuting Branch Structure.

10.5 Definitions of Permuting Activity Trees

The following concepts are defined for Permuting Activity Trees.

10.5.1 Reordered

An activity is reordered if and only if the set of atomic subactivity occurrences on each branch of one minimal activity tree is a permutation of the atomic subactivity occurrences on a branch in another activity tree for ?a.

```
(forall (?a) (iff (reordered ?a)
(forall (?occ1 ?occ2)
  (implies  (and      (occurrence_of ?occ1 ?a)
              (occurrence_of ?occ2 ?a))
          (branch_automorphic ?occ1 ?occ2)))))
```

10.5.2 Nondet_reordered

An activity is nondeterministically reordered if and only if the set of atomic subactivity occurrences on each branch of one minimal activity tree for ?a is a permutation of the atomic subactivity occurrences on some branch of another minimal activity tree for ?a.

```
(forall (?a) (iff (nondet_reordered ?a)
(forall (?occ1)
  (implies  (occurrence_of ?occ1 ?a)
          (exists (?occ2)
              (and(occurrence_of ?occ2 ?a)
```


(not (same_grove ?occ1 ?occ2))

(branch_automorphic ?occ1 ?occ2)))))))

10.5.3 Partial_reordered

An activity is partially reordered if and only if there exists a branch such that the set of atomic subactivity occurrences is a permutation of the atomic ubactivity occurrences on some other branch, and there exists a branch that is not a permutation of any other branch.

```
(forall (?a) (iff (partial_reordered ?a)
(exist (?occ1 ?occ2 ?occ3)
  (and(occurrence_of ?occ1 ?a)
      (occurrence_of ?occ2 ?a)
      (not (same_grove ?occ1 ?occ2))
      (branch_automorphic ?occ1 ?occ2)
      (occurrence_of ?occ3 ?a)
      (same_grove ?occ1 ?occ3)
      (forall (?occ4)
          (implies  (and      (occurrence_of ?occ4 ?a)
                    (not (= ?occ2 ?occ4))
                    (same_grove ?occ2 ?occ4))
                (not (branch_automorphic ?occ2 ?occ4))))))))
```

10.5.4 Unorderable

An activity is nonorderable if and only if none of the branches in an activity tree are branch automorphic to any other branches in other activity trees in the spectrum.

```
(forall (?a) (iff (unorderable ?a)
(forall (?occ1 ?occ2)
  (implies  (and      (occurrence_of ?occ1 ?a)
            (occurrence_of ?occ2 ?a)
```



```
(not (same_grove ?occ1 ?occ2)))

(not (branch_automorphic ?occ1 ?occ2))))))
```

10.6 Grammar for process descriptions of Permuting Activity Trees

The following grammar sentences describe process descriptions specified in KIF for Permuting Activity Trees (Branchwise Permuted Activity Axioms).

```
< reordered_spec > ::=  (forall (< variable >)

        (implies  (occurrence < variable > < term >)

            < occurrence_sentence >))


< nondet_reordered_spec > ::=   (forall (< variable >*)

        (implies  (occurrence < variable > < term >)

            (or < occurrence_sentence >*)))


< partial_reordered_spec > ::=     (or  < nondet_reordered_spec >

        < nonorderable_spec >)


< nonorderable_spec > ::=   (exists (< variable >*)

        (and< occurrence_formula >

            < branch_spec >))
```

11 Spectrum of Activities: Compacting Branch Structure

This clause characterizes all definitions pertaining to Spectrum of Activities: Compacting Branch Structure.

11.1 Primitive lexicon of Compacting Branch Structure

No primitive relations are required by the lexicon of Compacting Branch Structure.

11.2 Defined lexicon of Compacting Branch Structure

The following relations are defined in this clause:

— (compacted ?a);

— (nondet_compacted ?a);

— (partial_compacted ?a);

— (stiff ?a).

Each concept is described by informal semantics and a KIF axiom.

11.3 Theories required by Compacting Branch Structure

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

11.4 Definitional extensions required by Compacting Branch Structure

This extension requires Permuting Branch Structure.

11.5 Definitions of Compacting Branch Structure

The following concepts are defined for Compacting Branch Structure.

11.5.1 Compacted

An activity is compacted if and only if there exists a branch such that the set of atomic subactivity occurrences on each branch of the minimal activity tree is a permutation of the atomic subactivity occurrences on the first branch.

```
(forall (?a) (iff (compacted ?a)
(exists (?occ1)
  (and(occurrence_of ?occ1 ?a)
       (forall (?occ2)
```

```
(implies (and (occurrence_of ?occ2 ?a)
    (not (same_grove ?occ1 ?occ2)))
  (branch_homomorphic ?occ1 ?occ2))))))))
```

11.5.2 Nondet_compacted

An activity is nondeterministically compacted if and only if there exists a branch such that the set of atomic subactivity occurrences on each branch of the minimal activity tree is a permutation of the atomic subactivity occurrences on the first branch.

```
(forall (?a) (iff (nondet_compacted ?a)
(forall (?occ1)
  (implies (occurrence_of ?occ1 ?a)
    (exists (?occ2)
      (and(occurrence_of ?occ2 ?a)
        (not (same_grove ?occ1 ?occ2))
        (branch_homomorphic ?occ1 ?occ2)))))))
```

11.5.3 Partial_compacted

An activity is partially compacted if and only if there exists a branch such that the set of atomic subactivity occurrences on some branch, but not all, of the minimal activity tree is a permutation of the atomic subactivity occurrences on the first branch.

```
(forall (?a) (iff (partial_compacted ?a)
(exists (?occ1 ?occ2 ?occ3)
  (and(occurrence_of ?occ1 ?a)
    (occurrence_of ?occ2 ?a)
    (not (same_grove ?occ1 ?occ2))
    (branch_homomorphic ?occ1 ?occ2)
    (occurrence_of ?occ3 ?a)
    (same_grove ?occ3 ?occ1)
    (forall (?occ4)
      (implies (and (occurrence_of ?occ4 ?a)
```

```
(not (= ?occ2 ?occ4))
(same_grove ?occ2 ?occ4))
(not (branch_homomorphic ?occ2 ?occ4))))))))
```

11.5.4 Stiff

An activity is stiff if and only if none of the branches in an activity tree for ?a are branch epimorphic to the branches of any other activity tree for ?a.

```
(forall (?a) (if (stiff ?a)
(forall (?occ1 ?occ2)
  (implies  (and      (occurrence_of ?occ1 ?a)
              (occurrence_of ?occ2 ?a)
              (not (same_grove ?occ1 ?occ2)))
          (not (branch_homomorphic ?occ1 ?occ2))))))
```

11.6 Grammar for Compacting Branch Structure

The following grammar sentences describe process descriptions specified in KIF for Compacting Branch Structure.

```
< compacted_spec > ::=  (forall (< variable >)
  (implies  (occurrence < variable > < term >)
        (exists (< variable >)
           (and(occurrence < variable > < term >)
                < occurrence_axiom >)))))
```

```
< nondet_compacted_spec > ::=   (forall (< variable >)
  (implies  (occurrence < variable > < term >)
        (exists (< variable >)
           (and(occurrence < variable > < term >)
                (or < occurrence_axiom >*))))))
```

< partial_compacted_spec > ::= (or < nondet_compacted_spec >
< stiff_spec >)

< stiff_spec > ::= (exists (< variable >*)
(and< occurrence_formula >
< branch_spec >))

12 Spectrum of Activities: Activity Trees and Re-ordering

This clause characterizes all definitions pertaining to Spectrum Activities: Activity Tree and Re-ordering.

12.1 Primitive lexicon of Activity Trees and Re-ordering

No primitive relations are required by the lexicon of Activity Trees and Re-ordering.

12.2 Defined lexicon of Activity Trees and Re-ordering

The following relations are defined in this clause:

— (treeordered ?a);

— (partial_treeordered ?a);

— (nondet_treeordered ?a);

— (scrambled ?a).

Each concept is described by informal semantics and a KIF axiom.

12.3 Theories required by Activity Trees and Re-ordering

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

12.4 Definitional extensions required by Activity Trees and Re-ordering

This Extension requires Branch Structure and Ordering.

12.5 Definitions of Activity Trees and Re-ordering

The following concepts are defined for Activity Trees and Re-ordering.

12.5.1 Treeordered

An activity is treeordered if and only if the root subactivity occurrence of one occurrence is root automorphic with every other root subactivity occurrence of other occurrences of the activity.

```
(forall (?a) (iff (treeordered ?a)
(forall (?occ1 ?occ2)
    (implies  (and  (occurrence_of ?occ1 ?a)
        (occurrence_of ?occ2 ?a)
        (not (same_grove ?occ1 ?occ2))
        (root_automorphic ?occ1 ?occ2)))))
```

12.5.2 Nondet_treeordered

An activity occurrence is nondeterministically treeordered if and only if each root subactivity occurrence of the minimal activity tree is root automorphic to some other root subactivity occurrence in the occurrence tree.

```
forall (?a) (iff (nondet_treeordered ?a)
(forall (?occ1)
    (implies  (occurrence_of ?occ1 ?a)
        (exists (?occ2)
            (and    (occurence_of ?occ2 ?a)
                (not (same_grove ?occ1 ?occ2))
                (root_automorphic ?occ1 ?occ2)))))))
```


12.5.3 Partial_treeordered

An activity is partially treeordered if and only if there exist root subactivity occurrences of the minimal activity tree that are root automorphic to some other root subactivity occurrences in the occurrence tree, and there also exist root subactivity occurrences that are not root automorphic to any other root subactivity occurrence.

```
(forall (?a) (iff (partial_treeordered ?a)
(exists (?occ1 ?occ2)
    (and    (occurrence_of ?occ1 ?a)
        (occurrence_of ?occ2 ?a)
            (not (same_grove ?occ1 ?occ2))
            (root_automorphic ?occ1 ?occ2)
            (forall (?occ3)
                (implies  (and  (occurrence_of ?occ3 ?a)
                        (not (same_grove ?occ3 ?occ2)))
                    (not (root_automorphic ?occ3 ?occ2))))))))
```

12.5.4 Scrambled

An activity is scrambled if and only if none of the branches in an activity tree are root automorphic to any other activity tree.

```
(forall (?a) (iff (scrambled ?a)
(forall (?occ1 ?occ2)
  (implies  (and      (occurrence_of ?occ1 ?a)
                (occurrence_of ?occ2 ?a)
                (not (same_grove ?occ1 ?occ2)
            (not (root_automorphic ?occ1 ?occ2))))))
```

12.6 Grammar of Activity Trees and Re-ordering

The following grammar sentences describe process descriptions specified in KIF for Activity Trees and Re-ordering.

```
< treeordered_spec > ::=   (forall (< variable >)
```

```
                    (implies  (occurrence < variable > < term >)
                         < ordered_sentence >)) |
                 (forall (< variable >)
                    (implies  (occurrence < variable > < term >)
                         < ordered_formula >))

< nondet_treeordered_spec > ::=   (forall (< variable >)
                    (implies  (occurrence < variable > < term >)
                         (or < ordered_sentence >*)))

< partial_treeordered_spec > ::=   (forall (< variable >)
                    (implies  (occurrence < variable > < term >)
                         (or {< ordered_sentence >* |
                              < ordered_list >*})))

< scrambled_spec > ::=  (forall (< variable >)
                    (implies  (occurrence < variable > < term >)
                         (or < ordered_list >*)))
```

13 Spectrum and Subtree Containment

This clause characterizes all definitions pertaining to Spectrum and Subtree Containment.

13.1 Primitive lexicon of Spectrum and Subtree Containment

No primitive relations are required by the lexicon of Spectrum and Subtree Containment.

13.2 Defined lexicon of Spectrum and Subtree Containment

The following relations are defined in this clause:

— (subtree_embed ?occ1 ?occ2 ?a);

— (multiple_outcome ?a);

— (weak_outcome ?a);

— (nondet_outcome ?a);

— (imiscible ?a).

Each concept is described by informal semantics and a KIF axiom.

13.3 Theories required by Spectrum and Subtree Containment

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

13.4 Definitional extensions required by Spectrum and Subtree Containment

No Definitional Extensions are required by Spectrum and Subtree Containment.

13.5 Definitions of Spectrum and Subtree Containment

The following concepts are defined for Spectrum and Subtree Containement.

13.5.1 Subtree_embed

One minimal activity tree for ?a can be embedded into another as a subtree if and only if any atomic subactivity occurrence in the tree rooted in ?s1 can be mapped to an atomic subactivity occurrence in the tree rooted in ?s2 in such a way that the ordering is preserved.

```
(forall (?s1 ?s2 ?a) (iff (subtree_embed ?s1 ?s2 ?a)
(forall (?s3)
    (implies  (min_precedes ?s1 ?s3 ?a)
          (exists (?s4)
```



```
(and    (min_precedes ?s2 ?s4 ?a)
        (iso_occ ?s4 ?s3)))))))
```

13.5.2 Multiple_outcome

An activity is multiple outcome if and only if any two activity trees can be embedded into one another.

```
(forall (?a) (iff (multiple_outcome ?a)
(forall (?s1 ?s2)
  (implies  (and     (root ?s1 ?a)
                     (root ?s2 ?a))
            (or (subtree_embed ?s1 ?s2 ?a)
                (subtree_embed ?s2 ?s1 ?a))))))
```

13.5.3 Weak_outcome

An activity ?a is weak outcome if and only if there exists a minimal activity tree that can be embedded into all other minimal activity trees of ?a.

```
(forall (?a) (iff (weak_outcome ?a)
(exists (?s1)
  (and(root ?s1 ?a)
      (forall (?s2)
          (implies  (root ?s2 ?a)
                    (subtree_embed ?s2 ?s1 ?a)))))))
```

13.5.4 Nondet_outcome

An activity ?a has a nondeterministic outcome if and only if there exists a minimal activity tree that can be embedded into other minimal activity trees of ?a, and there also exists branches that cannot be embedded.

```
(forall (?a) (iff (nondet_outcome ?a)
(exists (?s1 ?s2 ?s3)
  (and(root ?s1 ?a)
```

(root ?s2 ?a)

(subtree_embed ?s2 ?s1 ?a)

(root ?s3 ?a)

(forall (?s4)

(implies (root ?s4 ?a)

(not (or (subtree_embed ?s3 ?s4 ?a)

(subtree_embed ?s4 ?s3 ?a)))))))))

13.5.5 imiscible

An activity is imiscible if and only if no tree in the spectrum can be embedded into any other.

(forall (?a) (iff (imiscible ?a)

(forall (?s1 ?s2)

(implies (and (root ?s1 ?a)

(root ?s2 ?a))

(not (or (subtree_embed ?s1 ?s2 ?a)

(subtree_embed ?s2 ?s1 ?a)))))))

13.6 Grammar for Permuting Branch Structure

The grammar for process descriptions of Permuting Branch Structure is equivalent to the grammar of a general process description sentence specified in ISO 18629-11.

14 Embedding constraints for activities

This clause characterizes all definitions pertaining to embedding constraints for activities.

14.1 Primitive lexicon of embedding constraints for activities

No primitive relations are required by the lexicon of embedding constraints for activities.

14.2 Defined lexicon of embedding constraints for activities

The following relations are defined in this clause:

— (live_branch ?s1 ?s2 ?a)

— (embedded ?s1 ?s2 ?a)

— (dead_branch ?s1 ?s2 ?a)

— (dead_occurrence ?s1 ?s2 ?a)

— (embed_tree ?s1 ?s2 ?s3 ?a)

— (subocc_equiv ?s1 ?s2 ?s3 ?a)

— (unrestricted ?occ)

Each concept is described by informal semantics and a KIF axiom.

14.3 Theories required by embedding constraints for activities

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

14.4 Definitional extensions required by embedding constraints for activities

No Definitional Extensions are required by embedding constraints for activities

14.5 Definitions of embedding constraints for activities

The following concepts are defined for embedding constraints for activities.

14.5.1 Live_branch

An atomic activity occurrence ?s2 is on a live branch of an embedded activity tree for ?a with root ?s1 if and only if it is a subactivity occurrence on the branch.

```
(forall (?s1 ?s2 ?a) (iff (live_branch ?s1 ?s2 ?a)

(exists (?occ)

  (and(occurrence_of ?occ ?a)
```

(root_occ ?s1 ?occ)

(min_precedes ?s1 ?s2 ?a)))))

14.5.2 Embedded

An activity occurrence is an element of the positive maximal embedded subtree if and only if it is an external activity occurrence that is between two subactivity occurrences on a branch of the occurrence tree.

```
(forall (?s1 ?s2 ?a) (iff (embedded ?s1 ?s2 ?a)
(exists (?s3)
  (and(root ?s2 ?a)
      (min_precedes ?s2 ?s3 ?a)
      (precedes ?s2 ?s1)
      (precedes ?s1 ?s3)
      (not (min_precedes ?s1 ?s3 ?a)))))
```

14.5.3 Dead_branch

An occurrence ?s2 is on a dead branch with respect to an activity tree for ?a with root occurrence ?s1 if and only if ?s2 is on a branch that is occurrence isomorphic to a branch of the embedded activity tree. The activity occurrence ?s2 is the earliest dead occurrence with respect to the activity tree for ?a with root occurrence ?s1 if and only if it is the successor of a non-leaf subactivity occurrence in the activity tree but there are no subactivity occurrences of the tree that are later than ?s2.

```
(forall (?s1 ?s2 ?a) (iff (dead_branch ?s1 ?s2 ?a)
(exists (?a1 ?s3)
  (and(root ?s1 ?a)
      (min_precedes ?s1 ?s3 ?a)
      (not (leaf ?s3 ?a))
      (= ?s2 (successor ?a1 ?s3))
      (not (exists (?s4)
          (and(precedes ?s3 ?s4)
              (min_precedes ?s3 ?s4 ?a))))))))
```

14.5.4 Dead_occurrence

An external activity occurrence ?s2 is a dead occurrence with respect to the activity tree for ?a with root occurrence ?s1 if and only if ?s2 is between two activity occurrences on a dead branch for ?a.

```
(forall (?s1 ?s2 ?a) (iff (dead_occurrence ?s1 ?s2 ?a)
(exists (?s3)
  (and(dead_branch ?s1 ?s3 ?a)
      (precedes ?s1 ?s2)
      (precedes ?s2 ?s3)))))
```

14.5.5 Embed_tree

?s1 and ?s2 are occurrences of subactivities of ?a that are elements of the same embedded activity tree for ?a with root ?s3.

```
(forall (?s1 ?s2 ?s3 ?a) (iff (embed_tree ?s1 ?s2 ?s3 ?a)
(exists (?a1 ?a2)
  (and(subactivity ?a1 ?a)
      (subactivity ?a2 ?a)
      (occurrence_of ?s1 ?a1)
      (occurrence_of ?s2 ?a2)
      (or  (live_branch ?s3 ?s1 ?a)
           (dead_branch ?s3 ?s1 ?a))
      (or  (live_branch ?s3 ?s2 ?a)
           (dead_branch ?s3 ?s2 ?a))))))
```

14.5.6 Subocc_equiv

Two activity occurrences ?s1,?s2 are subocc equivalent with respect to ?a if and only if they are elements of the same maximal embedded subtree for ?a with root occurrence ?s3, they are occurrences of subactivities, and they agree on whether or not they are subactivity occurrences of an occurrence of ?a.

```
(forall (?s1 ?s2 ?s3 ?a) (iff (subocc_equiv ?s1 ?s2 ?s3 ?a)
(and   (embed_tree ?s1 ?s2 ?s3 ?a)
```

(iff (min_precedes ?s3 ?s1 ?a)

(min_precedes ?s3 ?s2 ?a))))

14.5.7 unrestricted

An activity occurrence ?occ is unrestricted if and only if every branch in the maximal embedded subtree is a live branch.

```
(forall (?occ) (iff unrestricted ?occ)

(forall (?a ?s1 ?s2 ?s3)

    (implies (and   (occurrence_of ?occ ?a)

                    (root_occ ?s3 ?occ)

                    (embed_tree ?s1 ?s2 ?s3 ?a))

             (subocc_equiv ?s1 ?s2 ?s3 ?a)))))
```

14.6 Grammar for Embedding Constraints for Activities.

The grammar for process descriptions of Embedding Constraints for Activities is equivalent to the grammar of a general process description sentence specified in ISO 18629-11.

15 Skeletal Activity Trees

This clause characterizes all definitions pertaining to Skeletal Activity Trees.

15.1 Primitive lexicon of Skeletal Activity Trees

No primitive relations are required by the lexicon of Skeletal Activity Trees.

15.2 Defined lexicon of Skeletal Activity Trees

The following relations are defined in this clause:

— (fused ?occ);

— (embed_occ ?occ);

— (free ?occ);

— (assisted ?occ);

— (helpless ?occ);

— (unbound ?occ);

— (bound ?occ);

— (strict ?occ).

Each concept is described by informal semantics and a KIF axiom.

15.3 Theories required by Skeletal Activity Trees

This theory requires act_occ.th, complex.th, atomic.th, subactivity.th, occtree.th, psl_core.th.

15.4 Definitional extensions required by Skeletal Activity Trees

This extension requires Embedding Constraints for Activities

15.5 Definitions of Skeletal Activity Trees

The following concepts are defined for Skeletal Activity Trees.

15.5.1 Fused

An activity occurrence ?occ is fused if and only if every activity occurrence between a root and a leaf occurrence is also a subactivity occurrence of ?occ.

```
(forall (?occ) (iff (fused ?occ)
(forall (?s1 ?s2 ?s3)
  (implies (and    (root_occ ?s1 ?occ)
             (subactivity_occurrence ?s2 ?occ)
             (precedes ?s1 ?s3)
             (precedes ?s3 ?s2))
          (subactivity_occurrence ?s3 ?occ)))))
```

15.5.2 Embedd_occ

The branch of the occurrence tree containing the activity occurrence ?occ can be embedded into a dead branch in the same maximal embedded subtree.

```
(forall (?occ) (iff (embed_occ ?occ)
(forall (?a ?s ?s1 ?s2 ?s3)
```



```
(implies  (and      (occurrence_of ?occ ?a)
                    (root_occ ?s ?occ)
                    (next_subocc ?s1 ?s2 ?occ)
                    (precedes ?s1 ?s3)
                    (precedes ?s3 ?s2))
          (exists (?s4 ?s5 ?s6)
                    (and(precedes ?s4 ?s5)
                         (precedes ?s5 ?s6)
                         (iso_occ ?s4 ?s1)
                         (iso_occ ?s5 ?s3)
                         (iso_occ ?s6 ?s3)
                         (dead_branch ?s1 ?s4 ?a)
                         (dead_branch ?s1 ?s6 ?a)))))))
```

15.5.3 Free

An activity occurrence is free if and only if there exist activity occurrences in the same tree that are fused and there exist activity occurrences that are not fused. This is equivalent to saying that there do not exist necessary external activity occurrences.

```
(forall (?occ) (iff (free ?occ)
(exists (?occ1 ?occ2)
  (and(same_tree ?occ ?occ1)
       (same_tree ?occ ?occ2)
       (not (fused ?occ1))
       (fused ?occ2)))))
```

15.5.4 Assisted

An activity occurrence is assisted if and only if every activity occurrence in the same tree are not fused. This is equivalent to saying that there exist necessary external activity occurrences.

```
(forall (?occ) (iff (assisted ?occ)
```

```
(forall (?occ1)
  (implies  (same_tree ?occ ?occ1)
        (not (fused ?occ1))))))
```

15.5.5 Helpless

An activity is helpless if and only if between any two subactivity occurrences, there exists an external activity occurrence. This is equivalent to saying that external activities are always necessary.

```
(forall (?occ) (iff (helpless ?occ)
(forall (?s1 ?s2)
  (implies  (next_subocc ?s1 ?s2 ?occ)
        (exists (?s3)
           (and(precedes ?s1 ?s3)
                (precedes ?s3 ?s2)
                (not (subactivity_occurrence ?s3 ?occ))))))))
```

15.5.6 Unbound

An activity occurrence is unbound if and only if none of the activity occurrences in the same tree can be embedded into dead branches. This is equivalent to the nonexistence of forbidden external activities.

```
(forall (?occ) (iff (unbound ?occ)
(forall (?occ1)
  (implies  (same_tree ?occ1 ?occ)
        (not (embed_occ ?occ1))))))
```

15.5.7 Bound

An activity occurrence is bound if and only if there exist activity occurrences in the same tree that can be embedded into dead branches. This is equivalent to the existence of forbidden external activities.

```
(forall (?occ) (iff (bound ?occ)
(exists (?occ1 ?occ2)
  (and(same_tree ?occ1 ?occ)
```

(not (embed_occ ?occ1))

(same_tree ?occ2 ?occ)

(embed_occ ?occ2)))))

15.5.8 Strict

An activity occurrence is strict if and only if all occurrences in the same tree are fused. This is equivalent to saying that all external activity occurrences are forbidden.

```
(forall (?occ) (iff (strict ?occ)
(forall (?occ1)
  (implies  (same_tree ?occ ?occ1)
        (fused ?occ1)))))
```

15.6 Grammar for Skeletal Activity Tree

The grammar for process descriptions of Skeletal Activity Tree is equivalent to the grammar of a general process description sentence specified in ISO 18629-11.

16 Atomic Activities: Upwards Concurrency

This clause characterizes all definitions pertaining to Atomic Activities: Upwards Concurrency.

16.1 Primitive lexicon of Atomic Activities: Upwards Concurrency

No primitive relations are required by the lexicon of Atomic Activities: Upwards Concurrency.

16.2 Defined lexicon of Atomic Activities: Upwards Concurrency

The following relations are defined in this clause:

— (natural ?a ?s);

— (artificial ?a ?s);

— (performed ?a ?s);

— (up_ghost ?a ?s);

— (up_conflict ?a ?s);

— (quark ?a ?s).

Each concept is described by informal semantics and a KIF axiom.

16.3 Theories required by Atomic Activities: Upwards Concurrency

This theory requires act_occ.th, complex.th, atomic.th, subactivity.th, occtree.th, psl_core.th.

16.4 Definitional extensions required by Atomic Activities: Upwards Concurrency

No Definitional Extensions are required by Atomic Activities: Upwards Concurrency.

16.5 Definitions of Atomic Activities: Upwards Concurrency

The following concepts are defined for Atomic Activities: Upward Concurrency.

16.5.1 Natural

An activity is natural if and only if it is a subactivity of any activity that is possible whenever it is possible.

```
(forall (?a ?s) (iff (natural ?a ?s)
(forall (?a1)
  (implies  (and      (poss ?a ?s)
                 (poss ?a1 ?s))
            (subactivity ?a ?a1)))))
```

16.5.2 Artificial

An activity is an artificial activity if and only if it is possible concurrently with any another activity.

```
 (forall (?a ?s) (iff (artificial ?a ?s)
(forall (?a1)
  (implies  (and      (poss ?a ?s)
                 (poss ?a1 ?s)
                 (not (subactivity ?a1 ?a)))
            (poss (conc ?a ?a1) ?s)))))
```

16.5.3 Performed

An activity is a performed activity if and only if there exist subactivities that are not possible concurrently whenever the activity is possible.

```
 (forall (?a ?s) (iff (performed ?a ?s)
(exists (?a1 ?a2)
  (and(subactivity ?a1 ?a)
       (subactivity ?a2 ?a)
       (poss ?a ?s)
       (not (poss (conc ?a1 ?a2) ?s))))))
```

16.5.4 Up_ghost

An activity is an up_ghost activity if and only if whenever any superactivity is possible, then the activity is possible too.

```
 (forall (?a ?s) (iff (up_ghost ?a ?s)
(forall (?a1)
  (implies  (and      (poss ?a1 ?s)
              (subactivity ?a ?a1))
         (poss ?a ?s))))
```

16.5.5 Up_conflict

An activity is an up_conflict activity if and only if whenever a concurrent composition with any other activity is possible, then one of their subactivities is also possible.

```
 (forall (?a ?s) (iff (up_conflict ?a ?s)
(forall (?a1)
  (implies  (poss (conc ?a ?a1) ?s)
         (or (poss ?a ?s)
              (poss ?a1 ?s)
              (subactivity ?a ?a1))))))
```

16.5.6 Quark

An activity is a quark activity if and only if it is not possible in isolation but superactivities of it are possible.

```
(forall (?a ?s) (iff (quark ?a ?s)
(exists (?a1 ?s)
  (and(atomic ?a1)
      (subactivity ?a ?a1)
      (poss ?a1 ?s)
      (not (poss ?a ?s))))))
```

16.6 Grammar for Atomic Activities: Upwards Concurrency

The grammar for the process descriptions of Atomic Activities: Upward Concurrency is equivalent to the grammar of a general process description sentence specified in ISO 18629-11.

17 Atomic Activities: Downwards Concurrency

This clause characterizes all definitions pertaining to Atomic Activities: Downwards Concurrency.

17.1 Primitive lexicon of Atomic Activities: Downwards Concurrency

No primitive relations are required by the lexicon of Atomic Activities: Downwards Concurrency.

17.2 Defined lexicon of Atomic Activities: Downwards Concurrency

The following relations are defined in this clause:

— (superpose ?a ?s);

— (assistance ?a ?s);

— (team ?a ?s);

— (ghost ?a ?s);

— (conflict ?a ?s);

— (dysfunction ?a ?s).

Each concept is described by informal semantics and a KIF axiom.

17.3 Theories required by Atomic Activities: Downwards Concurrency

This theory requires:

- act_occ.th;
- complex.th;
- atomic.th;
- subactivity.th;
- occtree.th;
- psl_core.th.

17.4 Definitional extensions required by Atomic Activities: Downwards Concurrency

No definitional extension is required by this extension.

17.5 Definitions of Atomic Activities: Downwards Concurrency

The following concepts are defined for Atomic Activities: Downward Concurrency.

17.5.1 Superpose

An activity is a superposition activity if and only if every subactivity is possible whenever the activity itself is possible.

```
(forall (?a ?s) (iff (superpose ?a ?s)
(forall (?a1)
   (implies  (and      (subactivity ?a1 ?a)
                 (poss ?a ?s))
            (poss ?a1 ?s)))))
```

17.5.2 Assistance

An activity is an assistance activity if and only if whenever it is possible, any concurrent composition of its subactivities is also possible.

```
(forall (?a ?s) (iff (assistance ?a ?s)
(forall (?a1 ?a2)
   (implies  (and      (subactivity ?a1 ?a)
```

```
        (subactivity ?a2 ?a)

        (poss ?a ?s))

      (poss (conc ?a1 ?a2) ?s)))))
```

17.5.3 Team

An activity is a team activity if and only if whenever it is possible, none of its subactivities are possible is isolation.

```
(forall (?a ?s) (iff (team ?a ?s)

(exists (?a1 ?a2)

  (and(subactivity ?a1 ?a)

      (subactivity ?a2 ?a)

      (poss ?a ?s)

      (not (poss (conc ?a1 ?a2) ?s))))))
```

17.5.4 Ghost

An activity is a ghost activity if and only if whenever it is not possible, then none of its subactivities are possible.

```
(forall (?a ?s) (iff (ghost ?a ?s)

(forall (?a1)

  (implies (and    (subactivity ?a1 ?a)

          (poss ?a1 ?s))

        (poss ?a ?s)))))
```

17.5.5 Conflict

An activity is a conflict activity if whenever it is not possible, then some of its subactivities are possible.

```
(forall (?a ?s) (iff (conflict ?a ?s)

(forall (?a1 ?a2)

  (implies (and    (subactivity ?a1 ?a)
```

(subactivity ?a2 ?a)

(not (poss ?a1 ?s))

(poss (conc ?a1 ?a2) ?s))

(poss ?a ?s))))))

17.5.6 Dysfunction

An activity is a dysfunction activity if and only if there exist subactivities that are not possible, but their concurrent composition is possible.

```
(forall (?a ?s) (iff (dysfunction ?a ?s)
(exists (?a1 ?a2)
  (and(subactivity ?a1 ?a)
      (subactivity ?a2 ?a)
      (not (poss ?a1 ?s))
      (not (poss ?a ?s))
      (poss (conc ?a1 ?a2) ?s)))))
```

17.6 Grammar for Atomic Activities: Downwards Concurrency

The grammar for the process descriptions of Atomic Activities: Downwards Concurrency is equivalent to the grammar of a general process description sentence specified in ISO 18629-11.

18 Spectrum of Atomic Activities

This clause characterizes all definitions pertaining to Spectrum of Activities.

18.1 Primitive lexicon of Spectrum of Atomic Activities

No primitive relations are required by the lexicon of Spectrum for Atomic Activities.

18.2 Defined lexicon of Spectrum of Atomic Activities

The following relations are defined in this clause:

— (global_ideal ?a);

— (global_nonideal ?a);

— (global_filter ?a);

— (global_nonfilter ?a).

Each concept is described by informal semantics and a KIF axiom.

18.3 Theories required by Spectrum of Atomic Activities

This theory requires:

— act_occ.th;

— complex.th;

— atomic.th;

— subactivity.th;

— occtree.th;

— psl_core.th.

18.4 Definitional extensions required by Spectrum of Atomic Activities

This extension requires Preconditions for Activities.

18.5 Definitions of Spectrum of Atomic Activities

The following concepts are defined for Spectrum of Atomic Activities.

18.5.1 Global_ideal

An activity is a global filter activity if and only if every superactivity has equivalent preconditions whenever the activity is possible.

```
(forall (?a) (iff (global_ideal ?a)
(forall (?a1 ?s1 ?s2)
  (implies  (and      (subactivity ?a ?a1)
              (poss ?a ?s1)
              (poss ?a ?s2))
          (poss_equiv ?a1 ?s1 ?s2)))))
```

18.5.2 global_nonideal

An activity is a global_nonideal filter activity if and only if every superactivity has equivalent preconditions whenever the activity is not possible.

```
(forall (?a) (iff (global_nonideal ?a)
(forall (?a1 ?s1 ?s2)
  (implies (and     (subactivity ?a ?a1)
            (not (poss ?a ?s1))
            (not (poss ?a ?s2)))
        (poss_equiv ?a1 ?s1 ?s2)))))
```

18.5.3 global_filter

An activity is a global filter activity if and only if every subactivity has equivalent preconditions whenever the activity is possible.

```
(forall (?a) (iff (global_filter ?a)
(forall (?a1 ?s1 ?s2)
  (implies (and     (subactivity ?a1 ?a)
            (poss ?a ?s1)
            (poss ?a ?s2))
        (poss_equiv ?a1 ?s1 ?s2)))))
```

18.5.4 global_nonfilter

An activity is a global_nonfilter activity if and only if every subactivity has equivalent preconditions whenever the activity is not possible.

```
(forall (?a) (iff (global_nonfilter ?a)
(forall (?a1 ?s1 ?s2)
  (implies (and     (subactivity ?a1 ?a)
            (not (poss ?a ?s1))
            (not (poss ?a ?s2)))
```

(poss_equiv ?a1 ?s1 ?s2)))))

18.6 Grammar for Spectrum of Atomic Activities

The grammar for the process descriptions of Spectrum for Atomic Activities is equivalent to the grammar of a general process description sentence specified in ISO 18629-11.

19 Preconditions for Activities

This clause characterizes all definitions pertaining to Preconditions for Activities.

19.1 Primitive lexicon of Preconditions for Activities

No primitive relations are required by the lexicon of Preconditions for Activities.

19.2 Defined lexicon of Preconditions for Activities

The following relations are defined in this clause:

— (poss_equiv ?a ?occ1 ?occ2);

— (trunc ?occ);

— (unconstrained ?a).

Each concept is described by informal semantics and a KIF axiom.

19.3 Theories required by Preconditions for Activities

This theory requires act_occ.th, complex.th, atomic.th, subactivity.th, occtree.th, psl_core.th.

19.4 Definitional extensions required by Spectrum of Atomic Activities

This extension does not require any definitional extensions.

19.5 Definitions of Preconditions for Activities

The following concepts are defined for Preconditions for Activities.

19.5.1 poss_equiv

Two activity occurrences are precondition equivalent with respect to an activity iff the two occurrences agree on the legality of the occurrences of the activity.

(forall (?a ?occ1 ?occ2) (iff (poss_equiv ?a ?occ1 ?occ2)

```
(implies (and (occurrence_of ?s1 ?a)
              (occurrence_of ?s2 ?a))
   (iff (legal ?occ1)
        (legal ?occ2)))))
```

19.5.2 trunc

This relation is true if the activity occurrence ?occ1 is a successor of a legal activity occurrence.

```
(forall (?occ) (iff (trunc ?occ)
(forall (?occ1 ?a)
   (implies (= ?occ (successor ?a ?occ1))
            (legal ?occ1)))))
```

19.5.3 unconstrained

An activity is unconstrained iff all activity occurrences in the occurrence tree are precondition equivalent.

```
(forall (?a) (iff (unconstrained ?a)
(forall (?s1 ?s2)
  (implies (and (activity_occurrence ?s1)
                (activity_occurrence ?s2))
           (poss_equiv ?a ?s1 ?s2)))))
```

19.6 Grammar for Preconditions for Activities

The grammar for the process descriptions of Preconditions for Activities is equivalent to the grammar of a general process description sentence specified in ISO 18629-11.

Annex A
(normative)
ASN.1 Identifier of ISO 18629-41

To provide for unambiguous identification of an information object in an open system, the object identifier

```
iso standard 18629 part 41 version 1
```

is assigned to this part of ISO 18629. The meaning of this value is defined in ISO/IEC 8824-1 and is described in ISO 18629-1.

Annex B
(informative)
Example of process description using ISO 18629-41

The purpose of this annex is to provide a detailed scenario in which the ISO 18629 PSL is used in a knowledge-sharing effort which involves multiple manufacturing functions.

This scenario is an "interoperability" manufacturing scenario. This means that its goal is to show how PSL can be used to facilitate the communication of process knowledge in a manufacturing environment. Specifically, this scenario is centred around the exchange of knowledge from a process planner to a job shop scheduler.

This annex extends the test case introduced in ISO 18629-11 : 2005, Annex C to illustrate the application of outercore concepts in the specification of the manufacturing process of a product named GT-350.

B.1 GT-350 Manufacturing Processes

This section unites the various departmental processes into a high-level collection of activities which are enacted to create a GT-350 product. As described in the GT-350 product structure (see ISO 18629-11 : 2005, Table C.1), subcomponents of this product are either purchased, sub-contracted, or made internally. These process descriptions address the activities performed to manufacture the internal subcomponents. This top-down view of the manufacturing process provides an overall picture from an abstract, “make GT350” activity which is expanded down to the detailed departmental levels.

As the Figure B.1 below shows, the GT-350 manufacturing process is divided into 6 main areas of work. The first five: make interior, make drive, make trim, make engine and make chassis are all unordered with respect to each other but they must all be completed before final assembly takes place.

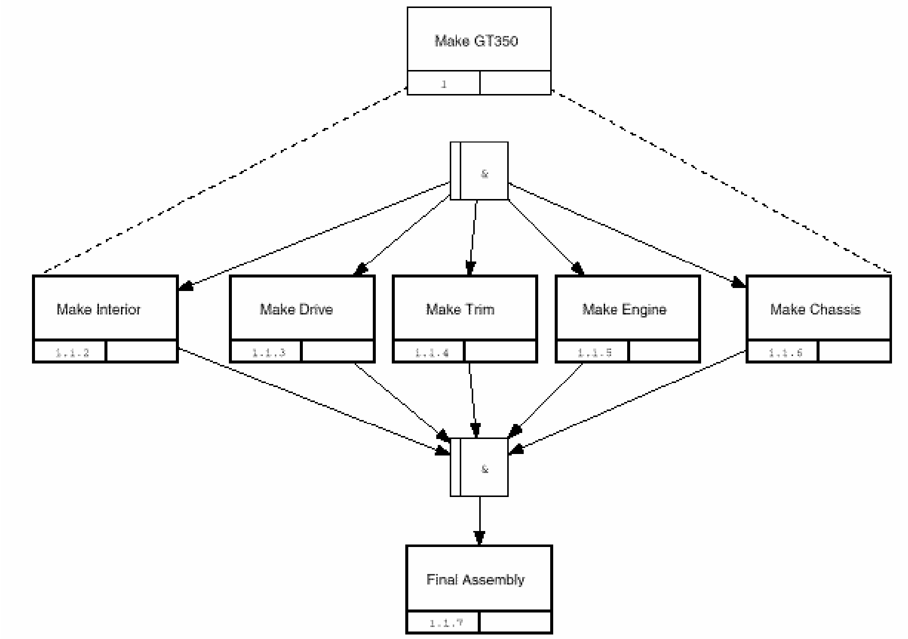

Figure B.1: TOP level process for manufacturing a GT350

The PSL-Outercore-based representation of the top level process is :

(subactivity make-chassis make_gt350)

(subactivity make-interior make_gt350)

(subactivity make-drive make_gt350)

(subactivity make-trim make_gt350)

(subactivity make-engine make_gt350)

(subactivity final-assembly make_gt350)

(artificial make_chassis)

(artificial make_interior)

(artificial make_drive)

(artificial make_trim)

(artificial make_enngine)

```
(forall (?occ)
    (implies  (occurrence_of ?occ make_gt350)
              (permuted ?occ)))
 (forall (?occ)
    (implies  (occurrence_of ?occ make_gt350)
              (ordered ?occ)))
 (forall (?occ)
    (implies  (occurrence_of ?occ make_gt350)
              (folded ?occ)))

(forall (?occ)
    (implies  (occurrence_of ?occ make_gt350)
              (amorphous ?occ)))
(not (uniform make_gt350))
(unrestricted make_gt350)
(not (atomic make_gt350))

(forall (?occ)
      (iff  (occurrence_of ?occ make_gt350)
            (exists (?occ1 ?occ2 ?occ3 ?occ4 ?occ5 ?occ6)
                  (and(occurrence_of ?occ1 make_chassis)
                      (occurrence_of ?occ2 make_interior)
                      (occurrence_of ?occ3 make_drive)
                      (occurrence_of ?occ4 make_trim)
```



```
(occurrence_of ?occ5 make_engine)
(occurrence_of ?occ6 final_assembly)
(subactivity_occurrence ?occ1 ?occ)
(subactivity_occurrence ?occ2 ?occ)
(subactivity_occurrence ?occ3 ?occ)
(subactivity_occurrence ?occ4 ?occ)
(subactivity_occurrence ?occ5 ?occ)
(subactivity_occurrence ?occ6 ?occ)

(forall (?s1 ?s2 ?s3 ?s4 ?s5 ?s6)
    (implies (and (leaf_occ ?s1 ?occ1)
        (leaf_occ ?s2 ?occ2)
        (leaf_occ ?s3 ?occ3)
        (leaf_occ ?s4 ?occ4)
        (leaf_occ ?s5 ?occ5)
        (root_occ ?s6 ?occ6))
      (and (min_precedes ?s1 ?s6 make_gt350)
        (min_precedes ?s2 ?s6 make_gt350)
        (min_precedes ?s3 ?s6 make_gt350)
        (min_precedes ?s4 ?s6 make_gt350)
        (min_precedes ?s5 ?s6 make_gt350) ))))))
```

This representation formalizes the process depicted in Figure B.1.

All occurrences of make_gt350 are permuted because all subactivities occur whenever make_gt350 occur.

All occurrences of make_gt350 are ordered because the subactivities make_chassis, make_interior, make_drive, make_engine, and make_trim must occur before the subactivity final_assembly.

Any of the subactivities can occur concurrently with any of the other subactivities; thus, all of the atomic subactivities are artificial, and all occurrences of the make_gt350 activity are folded.

Since there is no iteration within the make_gt350 activity, all occurrences are amorphous.

The subactivity make_engine may not occur if an engine already exists, so that not all occurrence of make_gt350 contains occurrences of the same set of subactivities with the same ordering constraints; thus, make_gt350 is not uniform.

There are no external activity occurrences that are necessary or forbidden with respect to the make_gt350 activity, and consequently it is unrestricted.

Each of these abstract activities can be further detailed, however for the example proposed in this annex, we will not develop all of them.

On the basis of the IDEF3 representation (in terms of process representation) of the abstract activities met during the different stages of the manufacturing process, we will extract some examples of process descriptions using the PSL-Outercore presented in ISO 18629-12.

B.2 The "make-engine" abstract activity

The 350-Engine is assembled from work performed in several CMW departments. The manufacturing process is shown in Figure B.2. The part is made up of an engine block, a harness, and wiring. The sub-processes are detailed in the sub-sections below. The 350-Engine is assembled at the A004 assembly bench and takes 5 minutes per piece.

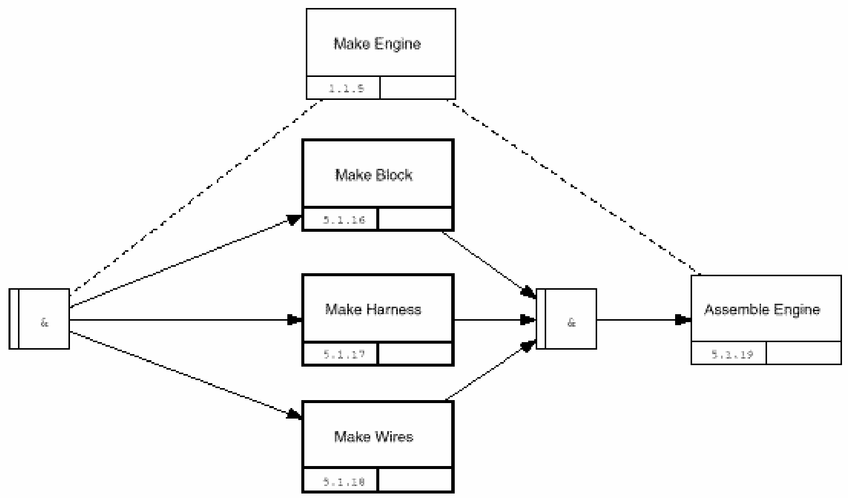

Figure B.2: PROCESS for manufacturing the 350–Engine

The PSL-based representation of some activities and of the process related information at the make-engine stage is :

(subactivity make_block make_engine)

```
(subactivity make-harness make_engine)

(subactivity make-wires make_engine)

(subactivity assemble_engine make_engine)

(artificial make_block)

(artificial make_harness)

(artificial make_wires)


(forall (?occ)

    (implies   (occurrence_of ?occ make_engine)

               (permuted ?occ)))

 (forall (?occ)

    (implies   (occurrence_of ?occ make_engine)

               (ordered ?occ)))

 (forall (?occ)

    (implies   (occurrence_of ?occ make_engine)

               (folded ?occ)))


(forall (?occ)

    (implies   (occurrence_of ?occ make_engine)

               (amorphous ?occ)))


 (uniform make_engine)

(unrestricted make_engine)

(not (atomic make_engine))
```

```
(forall (?occ)
 (iff(occurrence_of ?occ make_engine)
    (exists (?occ1 ?occ2 ?occ3 ?occ4)
       (and(occurrence_of ?occ1 make_block)
          (occurrence_of ?occ2 make_harness)
          (occurrence_of ?occ3 make_wires)
          (occurrence_of ?occ4 assemble_engine)
          (subactivity_occurrence ?occ1 ?occ)
          (subactivity_occurrence ?occ2 ?occ)
          (subactivity_occurrence ?occ3 ?occ)
          (subactivity_occurrence ?occ4 ?occ)
          (forall (?s1 ?s2 ?s3 ?s4)
             (implies  (and      (leaf_occ ?s1 ?occ1)
                  (leaf_occ ?s2 ?occ2)
                  (leaf_occ ?s3 ?occ3)
                              (root_occ ?s4 ?occ4))
                (and     (min_precedes ?s1 ?s4 make_engine)
                              (min_precedes ?s2 ?s4 make_engine)
                              (min_precedes ?s3 ?s4 make_engine)))))
```

This representation formalizes the process depicted in Figure B.2.

All occurrences of make_engine are permuted because all subactivities occur whenever make_engine occur.

All occurrences of make_engine are ordered because the subactivities make_block, make_harness, , and make_wires must occur before the subactivity assemble_engine.

Any of the subactivities can occur concurrently with any of the other subactivities; thus, all of the atomic subactivities are artificial, and all occurrences of the make_engine activity are folded.

Since there is no iteration within the make_engine activity, all occurrences are amorphous.

Any occurrence of make_engine contains occurrences of the same set of subactivities with the same ordering constraints; thus, make_gt350 is uniform. Thus, the make_engine activity can occur at any time under any conditions

There are no external activity occurrences that are necessary or forbidden with respect to the make_engine activity, and consequently it is unrestricted.

B.3 Make Block

The 350-Block is manufactured as part of the 350–Engine sub-assembly. This involves an integration of work from the foundry and machine shop, as shown in the Figure B.3.

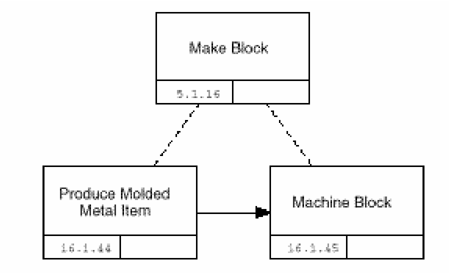

Figure B.3: PROCESS for manufacturing the 350–Block

The PSL-based representation of some activities and of the process related information is :

(subactivity produce_molded_metal make_block)

(subactivity machine_block make_block)

(primitive machine_block)

(primitive produce_molded_metal)

(performed machine_block)

(performed produce_molded_metal)

(forall (?occ)

(implies (occurrence_of ?occ make_block)

(permuted ?occ)))

```
 (forall (?occ)
    (implies   (occurrence_of ?occ make_block)
               (ordered ?occ)))
 (forall (?occ)
    (implies   (occurrence_of ?occ make_block)
               (rigid ?occ)))


(forall (?occ)
    (implies   (occurrence_of ?occ make_block)
               (amorphous ?occ)))



 (uniform make_block)
(unrestricted make_block)
(not (atomic make_block))


(forall (?occ)
      (iff  (occurrence_of ?occ make_block)
            (exists (?occ1 ?occ2)
                  (and(occurrence_of ?occ1 produce_molded_metal)
                        (occurrence_of ?occ2 machine_block)
                        (min_precedes ?occ1 ?occ2 make_block)))))
```

This representation formalizes the process depicted in Figure B.3.

All occurrences of make_block are permuted because all subactivities occur whenever make_block occur.

All occurrences of make_engine are ordered because the subactivity produce_molded_metal must occur before machine_block.

None of the subactivities can occur concurrently with any of the other subactivities; thus, all of the atomic subactivities are performed, and all occurrences of the make_block activity are rigid.

Since there is no iteration within the make_block activity, all occurrences are amorphous.

Any occurrence of make_block contains occurrences of the same set of subactivities with the same ordering constraints; thus, make_block is uniform.

There are no external activity occurrences that are necessary or forbidden with respect to the make_block activity, and consequently it is unrestricted.

B.4 Make Harness

The 350-Harness (Figure B.4) is manufactured as part of the 350–Engine sub-assembly. This involves work performed at the wire and cable department. Figure B.5 expands the harness wire production process. The 350-Harness is assembled by a bench worker at a wire and cable bench. It takes 10 minutes per set.

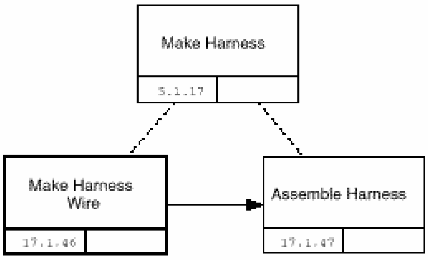

Figure B.4: PROCESS for manufacturing the 350–Harness

The PSL-based representation of some activities and of the process related information is :

(subactivity make_harness_wire make_harness)

(subactivity assemble_harness make_harness)

(primitive assemble_harness)

(performed make_harness_wire)

(performed assemble_harness)

(forall (?occ)

(implies (occurrence_of ?occ make_harness)

```
                  (permuted ?occ)))
 (forall (?occ)
     (implies   (occurrence_of ?occ make_harness)
                  (ordered ?occ)))
 (forall (?occ)
     (implies   (occurrence_of ?occ make_harness)
                  (rigid ?occ)))

(forall (?occ)
     (implies   (occurrence_of ?occ make_harness)
                  (amorphous ?occ)))

 (uniform make_harness)
(unrestricted make_harness)
(not (atomic make_harness))

(forall (?occ)
       (iff  (occurrence_of ?occ make_harness)
             (exists (?occ1 ?occ2 ?occ3)
                          (and(occurrence_of ?occ1 make_harness_wire)
                                (occurrence_of ?occ2 assemble_harness)
                                (leaf_occ ?occ3 ?occ1)
                                (min_precedes ?occ3 ?occ2 make_harness)))))
```

This representation formalizes the process depicted in Figure B.5.

All occurrences of make_harness are permuted because all subactivities occur whenever make_harness occur.

All occurrences of make_harness are ordered because the subactivity make_harness_wire must occur before assemble_harness.

None of the subactivities can occur concurrently with any of the other subactivities; thus, all of the atomic subactivities are performed, and all occurrences of the make_harness activity are rigid.

Since there is no iteration within the make_harness activity, all occurrences are amorphous.

Any occurrence of make_harness contains occurrences of the same set of subactivities with the same ordering constraints; thus, make_harness is uniform.

There are no external activity occurrences that are necessary or forbidden with respect to the make_harness activity, and consequently it is unrestricted.

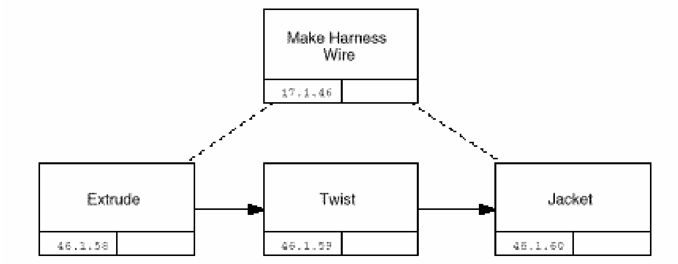

Figure B.5: PROCESS for manufacturing the harness wire

B.5 Make Harness Wires

The 350-Wire-Set is manufactured as part of the 350–Engine sub-assembly. This involves work performed at the wire and cable department.

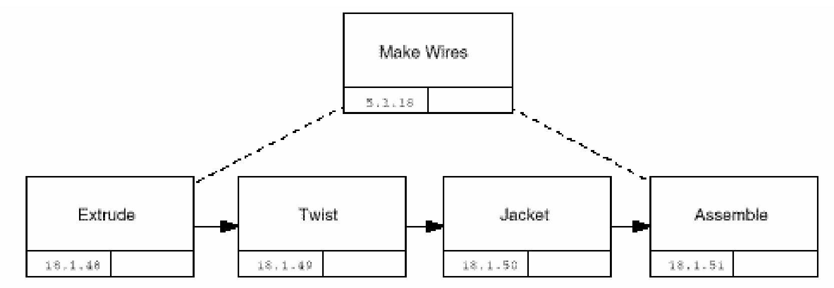

Figure B.6 : Process for manufacturing the 350-Wire

The PSL-Outercore-based representation of some activities and of the process related information is :

The PSL-Outercore-based representation of some activities and of the process related information is :

```
(subactivity extrude make_harness_wire)

(subactivity twist make_harness_wire)

(subactivity jacket make_harness_wire)

(primitive extrude)

(primitive twist)

(primitive jacket)


(performed extrude)

(performed twist)

(performed jacket)


(forall (?occ)
    (implies   (occurrence_of ?occ make_harness_wire)
               (permuted ?occ)))
 (forall (?occ)
    (implies   (occurrence_of ?occ make_harness_wire)
               (ordered ?occ)))
 (forall (?occ)
    (implies   (occurrence_of ?occ make_harness_wire)
               (rigid ?occ)))


(forall (?occ)
    (implies   (occurrence_of ?occ make_harness)
               (amorphous ?occ)))
```


(uniform make_harness_wire)

(not (unrestricted make_harness_wire))

(not (atomic make_harness_wire))

```
(forall (?occ)
    (iff  (occurrence_of ?occ make_harness_wire)
        (exists (?occ1 ?occ2 ?occ3)
            (and(occurrence_of ?occ1 extrude)
                (occurrence_of ?occ2 twist)
                (occurrence_of ?occ3 jacket)
                (min_precedes ?occ1 ?occ2 make_harness_wire)
                (min_precedes ?occ2 ?occ3 make_harness_wire))))
```

This representation formalizes the process depicted in Figure B.5.

All occurrences of make_harness_wires are permuted because all subactivities occur whenever make_harness_wires occur.

All occurrences of make_harness_wires are ordered because the subactivities extrude, twist, and jacket must occur in a unique ordering.

None of the subactivities can occur concurrently with any of the other subactivities; thus, all of the atomic subactivities are performed, and all occurrences of the make_harness_wires activity are rigid.

Since there is no iteration within the make_harness_wires activity, all occurrences are amorphous.

Any occurrence of make_harness_wires contains occurrences of the same set of subactivities with the same ordering constraints; thus, make_harness_wires is uniform.

Setup and changeover activities must occur between any two subactivity occurrences in an occurrence of the make_harness activity, and consequently it is not unrestricted.

Bibliography

[1] ISO 10303-49, *Industrial automation systems and integration — Product data representation and exchange — Part 49 : Integrated generic resources: Process structure and properties*

[2] ISO 18629-14, *Industrial automation systems and integration — Process specification language — Part 14 :Resource theories*

[3] ISO 18629-44, *Industrial automation systems and integration — Process specification language — Part 44 :Definitional extensions: Resource extensions*

Index

ICS 25.040.40

Price based on 75 pages