Ontology of the Process Specification Language

Michael Grüninger

Institute for Systems Research, University of Maryland, College Park, MD 20742
gruning@cme.nist.gov

1 Motivation

Representing activities and the constraints on their occurrences is an integral aspect of commonsense reasoning, particularly in manufacturing, enterprise modelling, and autonomous agents or robots. In addition to the traditional concerns of knowledge representation and reasoning, the need to integrate software applications in these areas has become increasingly important. However, interoperability is hindered because the applications use different terminology and representations of the domain. These problems arise most acutely for systems that must manage the heterogeneity inherent in various domains and integrate models of different domains into coherent frameworks. For example, such integration occurs in business process reengineering, where enterprise models integrate processes, organizations, goals and customers. Even when applications use the same terminology, they often associate different semantics with the terms. This clash over the meaning of the terms prevents the seamless exchange of information among the applications. Typically, point-to-point translation programs are written to enable communication from one specific application to another. However, as the number of applications has increased and the information has become more complex, it has been more difficult for software developers to provide translators between every pair of applications that must cooperate. What is needed is some way of explicitly specifying the terminology of the applications in an unambiguous fashion.

The Process Specification Language (PSL) ([13], [8]) has been designed to facilitate correct and complete exchange of process information among manufacturing systems [1]. Included in these applications are scheduling, process modeling, process planning, production planning, simulation, project management, workflow, and business process reengineering. This chapter will give an

[1] PSL has been accepted as project ISO 18629 within the International Organisation of Standardisation, and as of October 2002, part of the work is under review as a Draft International Standard.

overview of the PSL Ontology, including its formal characterization as a set of theories in first-order logic and the range of concepts that are axiomatized in these theories.

2 Formal Properties of the PSL Ontology

We will begin by considering the distinguishing formal characteristics of the PSL Ontology, independently from its content.

2.1 Semantics and Axiomatization

The PSL Ontology is a set of theories in the language of first-order logic, and the semantics of a first-order theory are based on the notion of an interpretation that specifies a meaning for each symbol in a sentence of the language. In practice, interpretations are typically specified by identifying each symbol in the language with an element of some algebraic or combinatorial structure, such as graphs, linear orderings, partial orderings, groups, fields, or vector spaces; the underlying theory of the structure then becomes available as a basis for reasoning about the concepts and their relationships.

We can evaluate the adequacy of the application's ontology with respect to some class of structures that capture the intended meanings of the ontology's terms by proving that the ontology has the following two fundamental properties:

- Satisfiability: every structure in the class is a model of the ontology.
- Axiomatizability: every model of the ontology is isomorphic to some structure in the class.

The purpose of the Axiomatizability Theorem is to demonstrate that there do not exist any unintended models of the theory, that is, any models that are not specified in the class of structures. In general, this would require second-order logic, but the design of PSL makes the following assumption (hereafter referred to as the Interoperability Hypothesis): *The ontology supports interoperability among first-order inference engines that exchange first-order sentences.* By this hypothesis, we do not need to restrict ourselves to elementary classes of structures when we are axiomatizing an ontology. Since the applications are equivalent to first-order inference engines, they cannot distinguish between structures that are elementarily equivalent. Thus, the unintended models are only those that are not elementary equivalent to any model in the class of structures.

Classes of structures for theories within the PSL Ontology are therefore axiomatized up to elementary equivalence – the theories are satisfied by any model in the class, and any model of the core theories is elementarily equivalent to a model in the class. Further, each class of structures is characterized up to isomorphism.

2.2 Invariants and Classification

Many ontologies are specified as taxonomies or class hierarchies, yet few ever give any justification for the classification. If we consider ontologies of mathematical structures, we see that logicians classify models by using properties of models, known as invariants, that are preserved by isomorphism. For some classes of structures, such as vector spaces, invariants can be used to classify the structures up to isomorphism; for example, vector spaces can be classified up to isomorphism by their dimension. For other classes of structures, such as graphs, it is not possible to formulate a complete set of invariants. However, even without a complete set, invariants can still be used to provide a classification of the models of a theory.

Following this methodology, the set of models for the core theories of PSL are partitioned into equivalence classes defined with respect to the set of invariants of the models. Each equivalence class in the classification of PSL models is axiomatized using a definitional extension of PSL. In particular, each definitional extension in the PSL Ontology is associated with a unique invariant; the different classes of activities or objects that are defined in an extension correspond to different properties of the invariant. In this way, the terminology of the PSL Ontology arises from the classification of the models of the core theories with respect to sets of invariants. The terminology within the definitional extensions intuitively corresponds to classes of activities and objects.

This approach can also be justified by the original motivation for PSL. If the ontologies of two software applications have the same language, then the applications will be interoperable if they share the semantics of the terminology in their corresponding theories. Sharing semantics between applications is equivalent to sharing models of their theories, that is, the theories have isomorphic sets of models. We therefore need to determine whether or not two models are isomorphic, and in doing so, we can use invariants of the models.

2.3 Types and Process Descriptions

If two software applications both used an ontology for algebraic fields, they would not exchange new definitions, but rather they would exchange sentences that expressed properties of elements in their models. For algebraic fields, such sentences are equivalent to polynomials. Similarly, the software applications that use PSL do not exchange arbitrary sentence, such as new axioms or even conservative definitions, in the language of their ontology. Instead, they exchange process descriptions, which are sentences that are satisfied by particular activities, occurrences, states, or other objects (see Figures 4 and 5).

Within PSL, we formally characterize a process description as a boolean combination of n-types [2] for the PSL Ontology that are realized by some model of the ontology. In the algebra example, polynomials are n-types for elements in an algebraic field. In the PSL core theory $T_{complex}$, formulae that specify the constraints under which subactivities of an activity occur are types for complex activities. In the axiomatization of situation calculus in [11], precondition and effect axioms are types for actions.

For general theories, there may exist elements whose types cannot be defined by a sentence in the language. For example, although polynomials are types for elements in an algebraic field, there exist real numbers that are not definable using polynomials, namely transcendental numbers such as π and e. Within PSL, the definable types for elements in each class of models are specified as classes of sentences using a BNF grammar. If an activity does not have a definable type, then the process description cannot be axiomatized, just as transcendental numbers cannot be specified by polynomials.

2.4 Relationship to Other Process Ontologies

There have been a variety of process ontologies developed within the artificial intelligence community, particularly in the context of robotics and planning systems.

One family of projects has attempted to provide a sharable ontology of planning information for use by disparate and communicating systems. The Sharable Plan and Activity Representation (SPAR) [16] specified an abstract ontology setting out major categories (such as space, time, agents, actions, reasoning, and plans), and a set of modular specialised ontologies which augment the general categories with sets of concepts and alternative theories of more detailed notions commonly used by planning systems, such as specific ontologies and theories of time points, temporal relations, and complex actions. SPAR evolved out of earlier work with the Common Plan Representation [9] and the O-Plan project [15] at the University of Edinburgh.

Another plan-oriented process ontology is the Planning Domain Definition Language (PDDL) [1], which is used extensively in the AIPS planning competition. PDDL is intended to express the particular domain used in a planning system, including a specification of states, the set of possible activities, the structure of complex activities, and the effects of activities.

The Cognitive Robotics Group at the University of Toronto has proposed the language GOLOG [6] as a high-level robotics programming language. GOLOG provides mechanisms for specifying complex activities as programs in

[2] An n-type for a first-order theory T is a set of formulae $\Phi(x_1, ..., x_n)$, such that for some model $\mathcal{M}$ of T, and some n-tuple $\overline{a}$ of elements of $\mathcal{M}$, we have $\mathcal{M} \models \phi(\overline{a})$ for all ϕ in Φ. If t is an n-type, then a model $\mathcal{M}$ realizes t if and only if there are $a_1, ..., a_n \in M$ such that $\mathcal{M} \models \phi(a_1, ..., a_n)$ for each $\phi \in t$. An n-type for a theory is a therefore consistent set of formulae (each of which has n free variables) which is satisfied by a model of the theory.

a second-order language that extends the axiomatization of situation calculus found in [11].

The Workflow Management Coalition has been developing a standard terminology which can serve as a common framework for different workflow management system vendors. The ontology for this effort is the Workflow Process Definition Language (WPDL) [17].

In the context of the Semantic Web, much work has been done using the DARPA Agent Markup Language (DAML) [4]. In particular, the DAML-S ontology [7] provides a set of process classes that can be specialized to describe a variety of Web services.

These approaches to process representation lack one or more of the formal properties of the PSL Ontology. Ontologies such as SPAR, CPR, INOVA, WPDL, and DAML-S do not provide a model theory. The ontologies for WPDL, PDDL, and those found in [12] specify a formal semantics, but they do not provide any axiomatization of this semantics. Finally, ontologies such as Golog and PDDL only specify syntactic classes of process descriptions without any underlying theory for complex activities.

3 Overview of PSL Core Theories

Within the set of first-order theories that comprise the PSL Ontology, there is a distinction between core theories and definitional extensions [3]. Core theories introduce new primitive concepts, while all terms introduced in a definitional extension that are conservatively defined using the terminology of the core theories.

All core theories within the ontology are consistent extensions of PSL-Core (T_{psl_core}), although not all extensions need be mutually consistent. Also, the core theories need not be conservative extensions of other core theories. The relationships among the core theories in the PSL Ontology are depicted in Figure 1. The lexicon of the core theories can be found in Table 1.

In the remainder of this section, we will consider the intuitions for each of the core theories in Figure 1. The definitional extensions in the PSL Ontology will be discussed in the next section.

3.1 PSL-Core

The purpose of PSL-Core is to axiomatize a set of intuitive semantic primitives that is adequate for describing the fundamental concepts of manufacturing processes [4]. Consequently, this characterization of basic processes makes few

[3] The complete set of axioms for the PSL Ontology can be found at `http://www.mel.nist.gov/psl/psl-ontology/`. Core theories are indicated by a .th suffix and definitional extensions are indicated by a .def suffix

[4] The axiomatization of PSL-Core is based on the results of the earlier Process Interchange Format project [5].

$T_{pslcore}$	$activity(a)$	a is an activity
	$activity\_occurrence(o)$	o is an activity occurrence
	$timepoint(t)$	t is a timepoint
	$object(x)$	x is an object
	$occurrence\_of(o, a)$	o is an occurrence of a
	$beginof(o)$	the beginning timepoint of o
	$endof(o)$	the ending timepoint of o
	$before(t_1, t_2)$	timepoint t_1 precedes timepoint t_2 on the timeline
$T_{subactivity}$	$subactivity(a_1, a_2)$	a_1 is a subactivity of a_2
	$primitive(a)$	a is a minimal element of the *subactivity* ordering
T_{atomic}	$atomic(a)$	a is either primitive or a concurrent activity
	$conc(a_1, a_2)$	the activity that the concurrent composition of a_1 and a_2
$T_{occtree}$	$successor(a, s)$	the element of an occurrence tree that is the next occurrence of a after the activity occurrence s
	$legal(s)$	s is an element of a legal occurrence tree
	$initial(s)$	s is the root of an occurrence tree
	$earlier(s_1, s_2)$	s_1 precedes s_2 in an occurrence tree
	$poss(a, s)$	there exists a legal occurrence of a that is a successor of s
T_{disc_state}	$holds(f, s)$	the fluent f is true immediately after the activity occurrence s
	$prior(f, s)$	the fluent f is true immediately before the activity occurrence s
$T_{complex}$	$min\_precedes(s_1, s_2, a)$	the atomic subactivity occurrence s_1 precedes the atomic subactivity occurrence s_2 in an activity tree for a
	$root(s, a)$	the atomic subactivity occurrence s is the root of an activity tree for a
T_{actocc}	$subactivity\_occurrence(o_1, o_2)$	o_1 is a subactivity occurrence of o_2
	$root\_occ(o)$	the initial atomic subactivity occurrence of o
	$leaf\_occ(s, o)$	s is the final atomic subactivity occurrence of o
$T_{duration}$	$timeduration(d)$	d is a timeduration
	$duration(t_1, t_2)$	the timeduration whose value is the "distance" from timepoint t_1 to timepoint t_2
	$lesser(d_1, d_2)$	the linear ordering relation over timedurations

Table 1. Lexicon for core theories in the PSL Ontology.

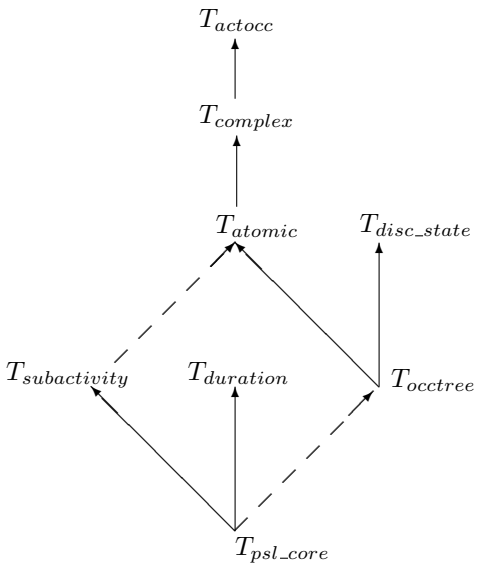

Fig. 1. The core theories of the PSL Ontology. Solid lines indicate conservative extension, while dashed lines indicate an extension that is not conservative.

assumptions about their nature beyond what is needed for describing those processes, and the Core is therefore rather weak in terms of logical expressiveness.

The basic ontological commitments of PSL-Core are based on the following intuitions:

> **Intuition 1:** *There are four kinds of entities required for reasoning about processes – activities, activity occurrences, timepoints, and objects.*

There are some approaches (e.g. [3]) that do not distinguish between timepoints and activity occurrences, so that activity occurrences form a subclass of timepoints. However, activity occurrences have preconditions and effects, whereas timepoints do not. Other approaches hold that timepoints are primitives but activity occurrences are not; for example, [14] claims that one can derive timepoints as "ticks" of a clock activity; however, such an approach ties the temporal ontology too closely to the process ontology.

> **Intuition 2:** *Activities may have multiple occurrences, or there may exist activities which do not occur at all.*

In contrast to many object-oriented approaches, activity occurrences cannot considered to be instances of activities, since activities are not classes within the PSL Ontology. [5]

There are also some historical reasons for making activity occurrences distinct from the notion of instances. The core theory $T_{occtree}$ introduces the notion of occurrence trees, which are isomorphic to situation trees without the initial situation S_0 (i.e. situations are identified as being occurrences of activities), and the situation calculus treats both activities and situations as first-order elements – situations are not instances of activities.

Intuition 3: *Timepoints are linearly ordered, forwards into the future, and backwards into the past.*

There are several options that may be taken to formalize this intuition. Within PSL-Core, an additional ontological commitment was made so that the timeline is infinite, with two endpoints, $inf-$ and $inf+$. This was done in order to capture the intuition that some objects exist only for a finite period of time, while other objects always exist, that is, there is no timepoint at which they are created or destroyed.

There are also different ontological commitments about time that are not made within the PSL Ontology, such as the denseness of the timeline. Any such commitments must be axiomatized within a core theory extension to PSL-Core.

Intuition 4: *Activity occurrences and objects are associated with unique timepoints that mark the begin and end of the occurrence or object.*

Note that the ontology allows for the existence of infinite activity occurrences; in these cases, $beginof$ or $endof$ will take on the values of $-inf$ or $+inf$.

3.2 Occurrence Trees

Models for the core theory $T_{occtree}$ are extensions of models of $T_{pslcore}$ by adding occurrence trees.

Intuition 5: *An occurrence tree is a partially ordered set of activity occurrences, such that for a given set of activities, all discrete sequences of their occurrences are branches of the tree.*

[5] One can of course specify classes of activities in a process description. For example the term $pickup(x, y)$ can denote the class of activities for picking up some object x with manipulator y, and the term $move(x, y, z)$ can denote the class of activities for moving object x from location y to location z. Ground terms such as $pickup(Block1, LeftHand)$ and $move(Shipment1, Seattle, Chicago)$ are instances of these classes of activities, and each instance can have different occurrences.

An occurrence tree contains all occurrences of *all* activities; it is not simply the set of occurrences of a particular (possibly complex) activity. Because the tree is discrete, each activity occurrence in the tree has a unique successor occurrence of each activity.

Occurrence trees are closely related to the situation trees that are models of Reiter's axiomatization of situation calculus ([11], [10]) if we interpret situations to be activity occurrences. However, there are some differences between $T_{occtree}$ and situation calculus. One interpretation of situation calculus is that the situations are sequences of actions, with the initial situation S_0 being the null sequence; since S_0 is not the occurrence of any activity, it has no corresponding object within PSL. Also, Reiter employs a second-order axiom to eliminate trees whose branches are isomorphic to nonstandard models of the natural numbers. However, such nonstandard trees are elementarily equivalent to standard trees, so that by the Interoperability Hypothesis, there is no need to invoke a second-order axiom.

Intuition 6: *There are constraints on which activities can possibly occur in some domain.*

This intuition is the cornerstone for characterizing the semantics of classes of activities and process descriptions. Although occurrence trees characterize all sequences of activity occurrences, not all of these sequences will intuitively be physically possible within the domain. We will therefore want to consider the subtree of the occurrence tree that consists only of *possible* sequences of activity occurrences; this subtree is referred to as the legal occurrence tree. For example, in Figure 2, there is no legal successor occurrence of $make\_frame$ after o_{16}^4, and there is no legal successor occurrence of $assemble$ after o_1^1. We will later discuss how the definitional extensions of the PSL Ontology use different constraints on possible activity occurrences as a way of classifying activities.

Intuition 7: *Every sequence of activity occurrences has an initial occurrence (which is the root of an occurrence tree).*

This intuition is closely related to the properties of occurrence trees. For example, one could consider occurrences to form a semilinear ordering (which need not have a root element) rather than a tree (which must have a root element). However, we are using occurrence trees to characterize the semantics of different classes of activities, rather than using the occurrence tree to represent history (which may not have an explicit initial event). In our case, it is sufficient to consider all possible interactions between the set of activities in the domain, and we lose nothing by restricting our attention to initial occurrences of the activities. For example, given the query "Can the factory produce 1000 widgets by Friday?", one can take the initial state to be the current state, and the initial activity occurrences being the activities that could be performed at the current time.

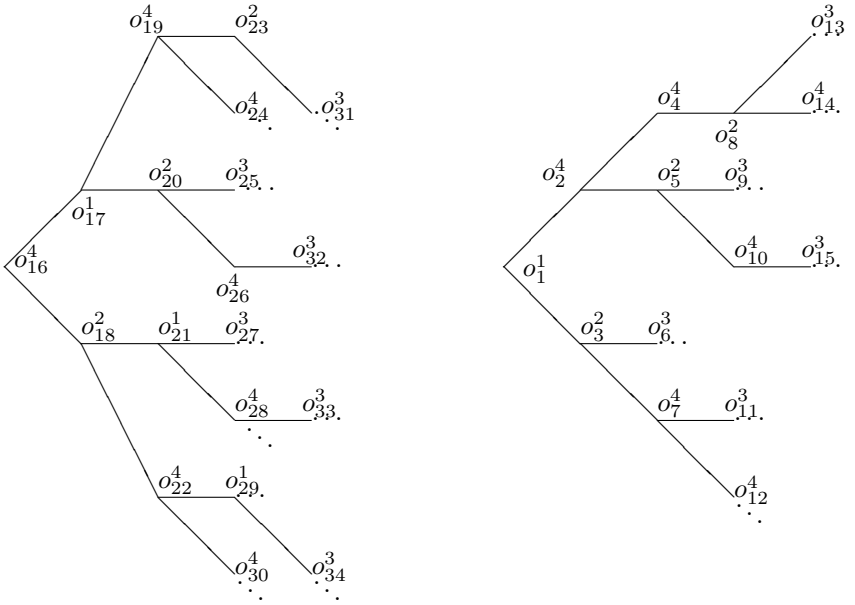

Fig. 2. Example of legal occurrence trees. The activities in the domain are *make_body*, *make_frame*, *paint*, and *polish*. The elements o^1_i denote occurrences of the activity *make_body*, o^2_i denote occurrences of the activity $make\_frame$, o^3_i denote occurrences of the activity *paint*, and o^4_i denote occurrences of the activity *polish*; for example, $o^2_4 = successor(make\_frame, o^1_1)$. The activity occurrences o^1_1 and o^4_{16} are the initial occurrences in their respective occurrence trees.

Intuition 8: *The ordering of activity occurrences in a branch of an occurrence tree respects the temporal ordering.*

Several approaches to temporal ontologies ([3]) conflate the ordering over activity occurrences with the ordering over timepoints. Within the theory of occurrence trees, these are distinct ordering relations. The set of activity occurrences is partially ordered (hence the intuition about occurrence *trees*), but timepoints are linearly ordered (since this theory is an extension of PSL-Core). However, every branch of an occurrence tree is totally ordered, and the intuition requires that the $beginof$ timepoint for an activity occurrence along a branch is $before$ the $beginof$ timepoints of all following activity occurrences on that branch.

3.3 Discrete States

Most applications of process ontologies are used to represent dynamic behaviour in the world so that intelligent agents may make predictions about the future and explanations about the past. In particular, these predictions

and explanations are often concerned with the state of the world and how that state changes. The PSL core theory T_{disc_state} is intended to capture the basic intuitions about states and their relationship to activities.

Intuition 9: *State is changed by the occurrence of activities.*

Intuitively, a change in state is captured by a state that is either achieved or falsified by an activity occurrence. We therefore need a relation that specifies the state that is intuitively true prior to an activity occurrence and also a relation that specifies the state that is intuitively true after an activity occurrence.

This illustrates another difference between PSL and versions of the situation calculus that use only the *holds* relation for reified fluents [6]. Since the situation S_0 does not correspond to an activity occurrence, the state prior to an initial activity occurrence is the same as the state that holds for S_0. However, since PSL refers only to initial activity occurrences, it requires an additional relation for the prior state such that all initial activity occurrences agree on the states prior to their occurrence; that is, if a fluent f holds prior to the initial activity occurrence of one occurrence tree, then it holds prior to the initial activity occurrence of all occurrence trees in the model. For example, in Figure 2, the same fluents hold prior to both initial activity occurrences o_1^1 and o_{16}^4.

Intuition 10: *State can only be changed by the occurrence of activities.*

Thus, if some state holds after an activity occurrence, but after an activity occurrence later along the branch it is false, then an activity must occur at some point between that changes the state. This also leads to the requirement that the state holding after an activity occurrence will be the same state holding prior to any immediately succeeding occurrence, since there cannot be an activity occurring between the two by definition.

Intuition 11: *State does not change during the occurrence of an activity in the occurrence tree.*

T_{disc_state} cannot represent phenomena in which some feature of the world is changing as some continuous function of time (hence the name "Discrete State" for the extension). State can change only during the occurrence of complex activities.

[6] Within situation calculus, fluents are situation dependent properties. In a non-reified approach, these properties are represented by predicates, whereas in the reified approach (adopted by the PSL Ontology), fluents are elements of the domain.

3.4 Subactivities

The PSL Ontology uses the *subactivity* relation to capture the basic intuitions for the composition of activities. This relation is a discrete partial ordering, in which primitive activities are the minimal elements.

The core theory $T_{subactivity}$ alone does not specify any relationship between the occurrence of an activity and occurrences of its subactivities. For example, consider the activities used in Figure 2. We can compose *paint* and *polish* as subactivities of some other activity, say $surfacing$, and we can compose *make_body* and $make\_frame$ into another activity, say $fabricate$. However, this specification of subactivities alone does not allow us to say that $surfacing$ is a nondeterministic activity, or that $fabricate$ is a deterministic activity.

3.5 Atomic Activities

The primary motivation driving the axiomatization of T_{atomic} is to capture intuitions about the occurrence of concurrent activities. Since concurrent activities may have preconditions and effects that are not the conjunction of the preconditions and effects of their activities, T_{atomic} takes the following approach:

> **Intuition 12:** *Concurrency is represented by the occurrence of one concurrent activity rather than multiple concurrent occurrences.*

Atomic activities are either primitive or concurrent (in which case they have proper subactivities). The core theory T_{atomic} introduces the function *conc* that maps any two atomic activities to the activity that is their concurrent composition. Essentially, what we call an atomic activity corresponds to some set of primitive activities, leading to the following intuition for this theory:

> **Intuition 13:** *every concurrent activity is equivalent to the composition of a set of primitive activities.*

Although $T_{subactivity}$ can represent arbitrary composition of activities, the composition of atomic activities is restricted to concurrency; to represent complex, or nonatomic, activities requires the next core theory.

3.6 Complex Activities

The core theory $T_{complex}$ characterizes the relationship between the occurrence of a complex activity and occurrences of its subactivities. Occurrences of complex activities correspond to sets of occurrences of subactivities; in particular, these sets are subtrees of the occurrence tree.

> **Intuition 14:** *An activity tree consists of all possible sequences of atomic subactivity occurrences beginning from a root subactivity occurrence.*

In a sense, activity trees are a microcosm of the occurrence tree, in which we consider all of the ways in which the world unfolds *in the context of an occurrence of the complex activity.* For example, if occurrence of the complex activity $fabricate$ consists of an occurrence of $make\_frame$ followed by an occurrence of *make_body*, then $\{o_1^1, o_{20}^2, o_{23}^2\}$ and $\{o_1^1, o_5^2, o_8^2\}$ can be two activity trees for $fabricate$. If an occurrence of the complex activity $surfacing$ consists of either an occurrence of *polish* or an occurrence of *paint*, then $\{o_9^3, o_{10}^2\}$ and $\{o_{13}^3, o_{14}^4\}$ can be two activity trees for $surfacing$.

Any activity tree is actually isomorphic to multiple copies of a minimal activity tree arising from the fact that other external activities may be occurring during the complex activity. For example, $\{o_1^1, o_5^2\}$ and $\{o_1^1, o_8^2\}$ are two isomorphic copies of the tree that captures the intuition "make the frame and then make the body". In the first case, o_2^4 and o_4^4 are occurrences of the activity *polish* during the occurrence of $fabricate$.

> **Intuition 15:** *Different subactivities may occur on different branches of the activity tree i.e. different occurrences of an activity may have different subactivity occurrences or different orderings on the same subactivity occurrences.*

In this sense, branches of the activity tree characterize the nondeterminism that arises from different ordering constraints or iteration. For example, the $surfacing$ activity is intuitively nondeterministic; the activity trees for $surfacing$ contain two branches, one branch consisting of an occurrence of *polish* and one branch consisting of an occurrence of *paint*.

Finally, there may be branches of a subtree of the occurrence tree that are isomorphic to branches of an activity tree, yet they do not correspond to occurrences of the activity. For example, in Figure 2, $\{o_{17}^1, o_{23}^2\}$ need not be an activity tree for $fabricate$, even though it is isomorphic to a branch of an activity tree.

> **Intuition 16:** *An activity will in general have multiple activity trees within an occurrence tree, and not all activity trees for an activity need be isomorphic. Different activity trees for the same activity can have different subactivity occurrences.*

Following this intuition, the core theory $T_{complex}$ does not constrain which subactivities occur. For example, conditional activities are characterized by cases in which the state that holds prior to the activity occurrence determines which subactivities occur. In fact, an activity may have subactivities that do not occur; the only constraint is that any subactivity occurrence must correspond to a subtree of the activity tree that characterizes the occurrence of the activity.

> **Intuition 17:** *Not every occurrence of a subactivity is a subactivity occurrence. There may be other external activities that occur during an occurrence of an activity.*

This theory does not force the existence of complex activities; there may be subtrees of the occurrence tree that contain occurrences of subactivities, yet not be activity trees. This allows for the existence of activity attempts, intended effects, and temporal constraints; subtrees that do not satisfy the desired constraints will simply not correspond to activity trees for the activity.

3.7 Complex Activity Occurrences

Within $T_{complex}$, complex activity occurrences correspond to activity trees, and consequently occurrences of complex activities are not elements of the legal occurrence tree. The axioms of the core theory T_{actocc} ensure complex activity occurrences correspond to branches of activity trees. Each complex activity occurrence has a unique atomic root occurrence and each finite complex activity occurrence has a unique atomic leaf occurrence. A subactivity occurrence corresponds to a sub-branch of the branch corresponding to the complex activity occurrence.

3.8 Duration

The core theory for duration essentially adds a metric to the timeline by mapping every pair of timepoints to a new sort called *timeduration* that satisfy the axioms of algebraic fields. All models of this theory are isomorphic to a projective vector space (since timedurations must also be values for durations with the timeline's endpoints at infinity). Of course, the duration of an activity occurrence is of most interest, and is equal to the duration between the endof and beginof timepoints of the activity occurrence.

4 Definitional Extensions

As discussed earlier, the set of models for the core theories of PSL are partitioned into equivalence classes defined with respect to the set of invariants of the models. Each equivalence class in the classification of PSL models is axiomatized within a definitional extension of the PSL Ontology. In this section, we will highlight the classes of activities in some of the definitional extensions of the PSL Ontology; in each case we will consider the invariants associated with the concepts and give examples of process descriptions for activities that are members of these classes.

Many of the invariants with definitional extensions in the PSL Ontology are related to the automorphism groups [7] for different substructures of the models. We will use the following notation for the relevant substructures:

[7] An automorphism is a bijection from a structure to itself that preserves the extensions of the relations and functions in the structure. Intuitively, automorphisms are the symmetries of a structure.

$\mathcal{T}$ is the timeline and $\mathcal{F}$ is the structure isomorphic to the extension of the *prior* relation. $Aut(\mathcal{F})$ is the group of permutations that map activity occurrences only to other activity occurrences that agree on the set of fluents that hold prior to them. $Aut(\mathcal{T})$ is the group of permutations that map activity occurrences only to other activity occurrences that agree on their *beginof* timepoints.

4.1 Occurrence Constraints

We first consider permutations of activity occurrences that map the predecessor of a legal occurrence of an activity **a** to other predecessors of legal occurrences of **a** in the occurrence tree. This set of permutations forms a group, which we will refer to as $OP(\mathbf{a})$. Each invariant related to occurrence constraints is based on subgroups of this group. For example, if we consider the activity *paint* and the occurrence trees in Figure 2, $OP(paint)$ is the group of permutations on the set that includes the activity occurrences $o_1^1, o_3^2, o_5^2, o_7^4, o_8^2, o_{10}^4$.

The most prevalent class of occurrence constraints is the case of Markovian activities, that is, activities whose preconditions depend only on the state prior to the occurrences (e.g. see Equation 6). The class of Markovian activities is defined in the definitional extension *state_precond.def* (see Figure 3). The invariant associated with this extension is the group $\mathcal{P}^{\mathcal{F}}(\mathbf{a})$, which is the maximal normal subgroup of $Aut(\mathcal{F})$ that is also a subgroup of $OP(\mathbf{a})$. If $\mathcal{P}^{\mathcal{F}}(\mathbf{a}) = Aut(\mathcal{F})$, then these permutations preserve the legal occurrences of an activity, and the activity's preconditions are strictly Markovian; this is axiomatized by the *markov_precond* class in Figure 3. If $\mathcal{P}^{\mathcal{F}}(\mathbf{a})$ is only a subgroup of $Aut(\mathcal{F})$, then there exist additional nonmarkovian constraints on the legal occurrences of the activity; this is axiomatized by the *partial_state* class in Figure 3. If $\mathcal{P}^{\mathcal{F}}(\mathbf{a})$ is the trivial identity group, then there are no Markovian constraints on the legal occurrences of the activity; this is axiomatized by the *rigid_state* class in Figure 3.

Additional relations are defined to capture the action of the automorphism groups on the models. Two activity occurrences o_1, o_2 are *state_equiv* iff there exists a permutation in $Aut(\mathcal{F})$ that maps o_1 to o_2; the two activity occurrences are *poss_equiv* iff there exists a permutation in $OP(\mathbf{a})$ that maps o_1 to o_2.

There are other kinds of preconditions that are independent of state. For example, there may be temporal preconditions, in which the legal occurrences of the activity depend only on the time at which the activity is to occur (e.g. Equation 7). The invariant for this extension (*time_precond.def*) in the PSL Ontology is the group $\mathcal{P}^{\mathcal{T}}(\mathbf{a})$, which is the maximal normal subgroup of $Aut(\mathcal{T})$ that is also a subgroup of $OP(\mathbf{a})$. If $\mathcal{P}^{\mathcal{T}}(\mathbf{a}) = Aut(\mathcal{T})$, then all legal occurrences of the activity are preserved by the permutations in $Aut(\mathcal{T})$, which is the case with temporal preconditions.

$$(\forall o_1, o_2)\ state\_equiv(o_1, o_2) \equiv (\forall f)\ (prior(f, o_1) \equiv prior(f, o_2)) \tag{1}$$

$$(\forall a, o_1, o_2)\ poss\_equiv(a, o_1, o_2) \equiv (poss(a, o_1) \equiv poss(a, o_2)) \tag{2}$$

$$(\forall a)\ markov\_precond(a) \equiv$$
$$((\forall o_1, o_2)\ state\_equiv(o_1, o_2) \supset poss\_equiv(a, o_1, o_2)) \tag{3}$$

$$(\forall a)\ partial\_state(a) \equiv$$
$$(\exists o_1)\ ((\forall o_2)\ state\_equiv(o_1, o_2) \supset poss\_equiv(a, o_1, o_2))$$
$$\wedge(\exists o_3, o_4)\ state\_equiv(o_3, o_4) \wedge \neg poss\_equiv(a, o_3, o_4) \tag{4}$$

$$(\forall a)\ rigid\_state(a) \equiv$$
$$(\forall o_1)(\exists o_2)\ state\_equiv(o_1, o_2) \wedge \neg poss\_equiv(a, o_1, o_2) \tag{5}$$

Fig. 3. Classes of activities with state-based preconditions (from the definitional extension $state\_precond.def$).

4.2 Effects

Effects characterize the ways in which activity occurrences change the state of the world. Such effects may be context-free, so that all occurrences of the activity change the same states, or they may be constrained by other conditions. The most common constraints are state-based effects that depend on the context (e.g. Equation 8). However, other kinds of constraints also arise in practice, such as time-based effects (e.g. Equation 9) and duration-based effects (e.g. Equation 10).

With respect to effects, activities are classified by the automorphism group of the structure that specifies which activity occurrences achieve or falsify a fluent. For example, if permutations of activity occurrences that preserve state are also automorphisms of this structure, then the effects of the activity depend only on the state; the associated classes of activities are found in the definitional extension $state\_effects.def$.

4.3 Conditional and Triggered Activities

There are several distinct classes in the PSL Ontology that are based on the relationship between fluents and activity occurrences. For conditional activities, fluents determine which subactivities occur (e.g. Equation 11). For triggered activities, fluents determine determine the conditions under which an activity

Mixing is not performed unless the moulding machine is clean.

$$(\forall o, x)\, occurrence\_of(o, mixing(x)) \wedge legal(o) \supset prior(clean(x), o) \tag{6}$$

The pre-heating operation can only be performed on Tuesday or Thursday.

$$(\forall o, x)\, occurrence\_of(o, preheat(x)) \wedge legal(o) \supset$$

$$(beginof(o) = Tuesday) \vee (beginof(o) = Thursday) \tag{7}$$

If the object is fragile, then it will break when dropped.

$$(\forall o, x)\, occurrence\_of(o, drop(x) \wedge prior(fragile(x), o)$$

$$\supset holds(broken(x), o) \tag{8}$$

If we remove the coffee pot before the brewing activity completes, then the burner will be wet.

$$(\forall o_1, o_2, x, y)\, occurrence\_of(o_1, brew(x, y)) \wedge occurrence\_of(o_2, remove(x, y))$$

$$\wedge before(beginof(o_2), beginof(o_1)) \supset holds(wet(y), o_1) \tag{9}$$

The time on the clock display will change after pressing the button for three seconds.

$$(\forall o, x)\, occurrence\_of(o, press(x)) \wedge duration(endof(o), beginof(o)) = 3$$

$$\supset holds(display(x), o) \tag{10}$$

Within the painting activity, if the surface of the product is rough, then sand the product.

$$(\forall s, o_1, x)\, occurrence\_of(o_1, paint(x)) \wedge root_occ(o_1) = s \wedge (prior(rough(x), s)$$

$$\supset (\exists o_2)\, occurrence\_of(a_2, sand(x)) \wedge root\_occ(o_2) = s) \tag{11}$$

Deliver the product when we have received three orders.

$$(\forall s, x)\, prior(order\_quantity(x, 3), s) \supset$$

$$(\exists o)\, occurrence\_of(o, deliver(x)) \wedge s = root\_occ(o) \tag{12}$$

Fig. 4. Examples of process descriptions in PSL associated with occurrence constraints, effects, conditional activities, and triggered activities.

must occur (e.g. Equation 12). Triggered activities differ from preconditions, which only specify the conditions under which an activity may possibly occur.

The automorphism groups for conditional activities consist of permutations of root occurrences of an activity tree that also preserve the structure of the minimal activity tree. In particular, if two activity occurrences are *state_equiv*, then they are the roots of isomorphic minimal activity trees.

The invariant for a triggered activity characterizes which subtrees are activity trees by looking at the automorphism groups that preserve occurrences

Making the frame involves the cutting and punching in parallel, followed by painting.

$$(\forall o, x)\ occurrence\_of(o, make(x)) \supset (\exists o_1, o_2, o_3)\ occurrence\_of(o_1, cut(x))$$
$$\wedge occurrence\_of(o_2, punch(x)) \wedge \wedge occurrence\_of(o_3, paint(x))$$
$$\wedge min\_precedes(root\_occ(o_1), root\_occ(o_3))$$
$$\wedge min\_precedes(root\_occ(o_2), root\_occ(o_3)) \quad (13)$$

Final assembly of the car consists of installation of either the manual or automatic transmission.

$$(\forall o, x)\ occurrence\_of(o, final(x)) \supset (\exists o_1)\ subactivity\_occurrence(o_1, o)$$
$$\wedge (occurrence\_of(o_1, manual(x)) \vee occurrence\_of(o_1, automatic(x))) \quad (14)$$

If the machine is not ready, then perform the painting before final assembly.

$$(\forall o, o_1, o_2, x, y)\ occurrence\_of(o, assembly(x, y))$$
$$\wedge occurrence\_of(o_1, paint(x)) \wedge occurrence\_of(o_1, final(x))$$
$$\wedge \neg prior(ready(y), root\_occ(o)) \supset min\_precedes(root\_occ(o_1), root\_occ(o_2)) \quad (15)$$

Fig. 5. Examples of process descriptions in PSL associated with ordering constraints.

of the activity. In this way, triggered activities are characterized by mappings in which permutations that preserve fluents also preserve the existence of activity trees for the activity.

4.4 Ordering Constraints

One of the most common intuitions about processes is the notion of process flow, or the specification of some ordering over the subactivities of an activity (e.g. Equation 13). The orderings themselves may also be nondeterministic. For example, there could be alternative process plans to produce the same product depending on the customer, (such as the process description in Equation 14) or the ordering may depend on state (as in the process description in Equation 15) Note that this latter constraint is distinct from conditional activities, since both painting and final assembly will occur; the nondeterminism in this case arises from the ordering of the occurrences of these activities.

The automorphism groups in this case consist of permutations of subactivity occurrences that are also automorphisms of the activity trees considered as graphs; the corresponding classes of activities are found in the definitional extension $ordering.def$.

5 Summary

Within the increasingly complex environments of enterprise integration, electronic commerce, and the Semantic Web, where process models are maintained in different software applications, standards for the exchange of this information must address not only the syntax but also the semantics of process concepts. PSL draws upon well-known mathematical tools and techniques to provide a robust semantic foundation for the representation of process information. This foundation includes first-order theories for concepts together with complete characterizations of the satisfiability and axiomatizability of the models of these theories. The PSL Ontology also provides a justification of the taxonomy of activities by classifying the models with respect to invariants. Finally, process descriptions are formally characterized as syntactic classes of sentences that are satisfied elements of the models.

References

1. Ghallab, M. et al. (1998) *PDDL: The Planning Domain Definition Language v.2*. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control.
2. Gruninger, M., and Fox, M.S., (1995), The Role of Competency Questions in Enterprise Engineering, *Benchmarking: Theory and Practice*, Rolstadas, A. (ed). Kluwer Academic Publishers, Boston.
3. Hayes, P. (1996) *A Catalog of Temporal Theories*. Artificial Intelligence Technical Report UIUC-BI-AI-96-01, University of Illinois at Urbana-Champaign.
4. Hendler, J. amd McGuinness, D.L. (2001): DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15:72-73.
5. Lee, J., Gruninger, M., Jin, Y., Malone, T., Tate, A., Yost, G. (1998) The PIF Process Interchange Format and Framework, *Knowledge Engineering Review*, 2:1-30.
6. Levesque, H., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. (1997): GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:92-128.
7. McIlraith, S., Son, T.C. and Zeng, H. (2001) Semantic Web Services, *IEEE Intelligent Systems*, Special Issue on the Semantic Web. 16:46–53, March/April, 2001.
8. Menzel, C. and Gruninger, M. (2001) A formal foundation for process modeling, *Second International Conference on Formal Ontologies in Information Systems*, Welty and Smith (eds), 256-269.
9. Pease, A. and Carrico, T.D. *Core Plan Representation*. Armstrong Lab Report AL/HR-TP-96-9631, Armstrong Laboratory, US Air Force, January 1997. Object Modeling Working Group.
10. Pinto, J. and Reiter, R. (1993) Temporal reasoning in logic programming: A case for the situation calculus. *Proceedings of the 10th International Conference on Logic Programming*, Budapest, Hungary, June 1993.

11. Reiter, R. (2001) *Knowledge in Action : Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.
12. Sandewall, E. (1994) *Features and Fluents*. Oxford Science Publications.
13. Schlenoff, C., Gruninger, M., Ciocoiu, M., (1999) The Essence of the Process Specification Language, *Transactions of the Society for Computer Simulation* vol.16 no.4 (December 1999) pages 204-216.
14. Sowa, J. (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole Publishing.
15. Tate, A., Drabble, B., Kirby, R. (1994) O-Plan: An open architecture for command, planning, and control. In M. Fox and M. Zweben (eds.) *Intelligent Scheduling*. Morgan Kaufmann, 1994.
16. Tate, A. (1998), "Roots of SPAR - Shared Planning and Activity Representation", *The Knowledge Engineering Review*, Vol 13(1), pp. 121-128, Special Issue on Putting Ontologies to Use (eds. Uschold. M. and Tate, A.), Cambridge University Press, March 1998.
17. Workflow Management Coalition (1999) *Process Definition Meta-Model and WPDL*, WfMC-TC-1016-P v1.1.