

A Formal Foundation for Process Modeling

Christopher Menzel
Department of Philosophy
Texas A&M University
College Station, TX 77840-4237
cmenzel@tamu.edu

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

Abstract: Process modeling is ubiquitous in business and industry. While a great deal of effort has been devoted to the formal and philosophical investigation of processes, surprisingly little research connects this work to real world process modeling. The purpose of this paper is to begin making such a connection. To do so, we first develop a simple mathematical model of activities and their instances based upon the model theory for the NIST Process Specification Language (PSL), a simple language for describing these entities, and a semantics for the latter in terms of the former, and a set of axioms for the semantics based upon the NIST Process Specification Language (PSL). On the basis of this foundation, we then develop a general notion of a process model, and an account of what it is for such a model to be realized by a collection of events.

Categories & Descriptors: I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods—*Predicate logic, Representation languages*

General Terms: Languages, Theory, Standardization

Keywords: Process modeling, Process Specification Language, PSL, Formal ontology

1. Introduction

Process modeling is ubiquitous in business and industry. While a great deal of effort has been devoted to the formal and philosophical investigation of processes, surprisingly little research connects this work to real world process modeling. The purpose of this paper is to begin making such a connection. To do so, we first develop a simple mathematical model of activities and their instances — which we refer to as “occurrences” or “events” — based upon the model theory for the NIST Process Specification Language (PSL), a simple language for describing these entities, a semantics for the latter in terms of the former, and a set of axioms for the semantics. On the basis of this foundation, we then develop a general notion of a process model, and an account of what it is for such a model to be realized by a collection of events.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FOIS '01, October 17-19, 2001, Ogunquit, Maine, USA.

Copyright 2001 ACM 1-58113-377-4/01/0010...\$5.00.

2. The Problem of Process Modeling

Processes are patterns of activities. Process modeling is the linguistic, diagrammatic, or numerical representation of such patterns. Process models, in these various forms, are ubiquitous in industry: there is a plethora of business and engineering applications — workflow, scheduling, discrete event simulation, process planning, business process modeling, and others — that are designed explicitly for the construction of process models of various sorts.

In general, business and engineering processes are described at the type level — a process model characterizes a certain general process *structure*. That structure, in turn, might admit of many instances which — depending on how constrained the structure is — might differ considerably from one another. A robust foundation for process modeling, therefore, should be able to characterize both the general process structure described by a model as well as the class of possible *instances* of that structure. Moreover, such a foundation must be able clearly to represent the constraints that a process model places on something's *counting* as an instance of the process, the constraints on process execution. However, in our view, existing frameworks that have been used to represent processes — Petri nets, for example (cf. [4]) — while often powerful, all fall short in at least one of these two respects.

Our purpose in this paper, then, is to make some significant first steps toward a sound theoretical foundation for process modeling that does exhibit either of these shortfalls. After describing the intuitive ontology underlying our approach, we provide a rigorous account of the language of the approach and its semantics. We then introduce a formal notion of a process model, or *process specification*, and define formally what it for a process model to be realized in the world by a series of appropriately structured events.

3. The Intuitive Ontology

Before launching into formal definitions it might be useful briefly to sketch the intuitive ontology that we will be trying to capture with those definitions. Some of the intuitions behind this ontology can arguably be traced back to Aristotle, but they probably comport most faithfully with the picture developed by Wittgenstein in [9]. Central to this picture is the notion of a *fact* or, more broadly, a *state of affairs*. A state of affairs is a simple configuration of one or more objects; for example, *the Washington monument's being 555.5 feet high*, *the moon's being larger than the sun*, *New York's being east of Chicago*, and so on. As the second example here shows, a state of affairs needn't *obtain*, or *hold*; the moon is not, in fact, larger than the sun. Those states of affairs that do obtain are known as *facts*.

The notion of a fact was critical to Wittgenstein's early philosophy. For him, and to a large degree for us here, as he famously put it, "the world is the totality of facts, not of things" ([9], 1.1). That is, the world is more than just its constituent objects; many different possible worlds contain those same objects as well. Rather, what gives the world its unique character is the way its constituent objects are configured, i.e., the properties they exhibit and the relations they stand in; in a word, the *facts*.

Wittgenstein's picture is appealing as far as it goes, but, like a lot of early contemporary philosophy, it largely ignores issues of time and change. For when time is factored in, what Wittgenstein calls "the world", viz., the current totality of all the facts, the current configuration of all the objects there are, just seems like a momentary "slice" of

the world — the world as configured *now*. Call such a slice of the world (regardless of whether there ever actually has been or will be such a slice, as long as it is *possible*) a *world state*, or *state*, for short. Intuitively, contrary to the Wittgensteinian picture, the world as a whole does not exhibit a single, unchanging state, a single static configuration of its constituent objects. For a state is simply a way the world might be, the facts that might obtain, *at some particular time or other*. The world itself, by contrast, is in flux; it persists through time, but constantly changes states. Thus, the world is better thought of, not as a single fixed state, a single totality of facts, but as a *series* of states — one state for each point in time. We will call any series of states that the world could exhibit over time in this way a *history*.

The state of the world at any given moment in its history determines which *propositions* are true at that moment. Propositions are not data, like sentences or bit streams. Rather, they are the *content*, or *meaning*, of such data, the *information* that the data carry (under some conventional interpretation). A proposition is said to be *true* at a given time when it is “good” information — if, as far as it goes, it accurately represents the state of the world at that time; if, that is, it “fits the facts”. (This is, of course, a version of the so-called “correspondence theory of truth” on which propositions are true in virtue of correctly characterizing, or corresponding to, the world.)

Propositions are things that can be true at a time; *activities* are things that can *happen*, or *occur*, over intervals of time. Just as the same proposition can be true at different times, the same activity — e.g., *baking a cake* or *painting a car body* — can occur over different intervals. Thus, activities proper are a type of *universal* and are not bound by time. By contrast, their instances, which we will call *events*, or *occurrences* — the actual bakings and paintings — have distinct temporal boundaries, a *beginning point* in time and an *ending point*.¹ Our ontology thus includes an *occurrence-of* relation that holds between a timebound event and the general activity of which it is an instance.

Intuitively, events can be thought of as things that jointly *bring about* future states in the history of the world from past states. A baking brings the world to a state involving a cake; a painting to a state in which a certain object has a certain color. A *process*, then, can be seen as a structured collection of activities whose instances, when occurring in accordance with the structure of the process, jointly carry the world from one state to another in a predictable and, in the case of engineered processes anyway, more or less controlled fashion. As we will see in Section 9, it will be unnecessary to include processes *per se* in our ontology. Rather, it will be enough simply to define a rigorous (syntactic) notion of a process specification and its (semantic) realization.

4. Histories and Activity Frames

We turn now to the formal representation of this intuitive ontology by means of a series of mathematical structures — based upon the basic model theory for PSL — that are meant to capture the actual properties these objects exhibit and logical relationships that they bear to one another. Note that these structures will be quite simple. For example, there is no *subactivity* relation on activities,² and propositions will have

¹At least, we will *model* events so that they all have temporal boundaries.

²Note, however, that there is an extension of PSL in which activities can have subactivities.

no quantificational structure. A more sophisticated approach will of course include these features. But for our purposes here — to indicate our basic approach to the formal foundations of process modeling — these features are tangential, and are best introduced in more advanced developments.

4.1 Histories

Timepoints, facts — or more generally, “states of affairs” — and world states are the basic ingredients in the notion of a history, a possible course that the world could take. We define the notion formally as follows.

Definition 1. A history is a 4-tuple $\langle \mathcal{T}, SOA, States, st \rangle$ such that

- $\mathcal{T} = \langle Time_{\mathcal{T}}, <_{\mathcal{T}} \rangle$ is a linear ordering that is infinite in both directions and has endpoints at infinity;³
- SOA and $Time_{\mathcal{T}}$ are disjoint;
- $States \subseteq \wp(SOA)$ (= the set of all subsets of SOA).
- $st : Time_{\mathcal{T}} \longrightarrow States$.

Time is represented by the pair $\langle Time_{\mathcal{T}}, <_{\mathcal{T}} \rangle$. Intuitively, $Time_{\mathcal{T}}$ is the set of timepoints and $<_{\mathcal{T}}$ is its natural linear ordering; the exact type of the ordering — discrete, dense, continuous — is left open. SOA is the set of (actual and possible) states of affairs (relative to some relevant domain). A world state is a complete way that the world could be at any given point in time. Hence, a state can be thought of as the set of states of affairs that are actual at that time, and hence simply as a subset of SOA .⁴ The set $States$, being a set of such subsets, thus represents the set of all possible states of the world, all the ways the world could be at some point in time. Finally, st is a mapping from each timepoint to the state of the world at that point — and thereby represents the way that the facts in the world change over time, i.e., the world’s history.⁵

4.2 Activity Frames

Given the notion of a history, we formalize activities and events in the notion of an activity frame.

Definition 2. Let $\mathfrak{H} = \langle \mathcal{T}, SOA, States, st \rangle$ be a history. An activity frame (over \mathfrak{H}) is a 6-tuple $\langle \mathfrak{H}, A, E, occ, beg, end \rangle$ such that

- The sets $Time_{\mathcal{T}}$, SOA , A , and E are pairwise disjoint;
- $occ : E \longrightarrow A$;
- $beg : E \longrightarrow Time_{\mathcal{T}}$, $end : E \longrightarrow Time_{\mathcal{T}}$, and for all $s \in E$, $beg(s) <_{\mathcal{T}} end(s)$.

³For instance, the most natural model of discrete time with these constraints would be $1 + \xi + 1$, where ξ is the order type of the integers. Many other (generally more complex) models of time are possible, of course; see, e.g., van Benthem’s compendious [1] and Hayes’s useful [2].

⁴In the model presented in this short paper, states of affairs have no internal structure. In more detailed developments of this account, states of affairs consist of an n -place relation (= property, for $n = 1$) and n arguments. It is also assumed that all states of affairs are logically independent. This assumption is made only for the sake of simplicity, and could easily be dropped.

⁵Sandewall defines a similar, though somewhat sparser notion of a history in [8] (p. 5). Despite a number of parallels, the approach here differs markedly from Sandewall’s.

Intuitively, of course, A is the set of activities, E is the set of events, and occ maps each event to the activity of which it is an instance. beg and end map events to their beginning and ending timepoints, respectively, and hence the beginning timepoint $beg(e)$ of an event e must be earlier in the timepoint ordering than the ending timepoint $end(e)$. (We therefore rule out instantaneous events, though this is not essential to the account.)

5. Propositions

As indicated in our intuitive overview, it is essential to the robust representation of processes that there be some reasonably rich notion of proposition. Any number of constructions would suffice for the formal model we are developing here. But the present framework admits of a very convenient one. As noted, states are ways that the world, as a whole, could be. One useful way of thinking of a proposition, then, is as a *partial approximation* of a state, something that “specifies” only part of what would be the case were the world in a certain state. Equivalently, a proposition can be thought of as a *type* of state, a type that holds of all those states that are alike in some regard. Thus (using some of the notation of the following section), the proposition $George_loves_Martha$ approximates, or is the type of, any state in which George loves Martha.

More formally, then, continuing in the extensional style of the model we are developing, we can represent a proposition, a type of state, simply as a *set* of states.⁶ Thus, $George_loves_Martha$ will be represented as the set that contains all of the states that contain the state of affairs *George’s loving Martha*. A conjunctive proposition [and p q] is the type exhibited by states that are of both types p and q ; hence the conjunction of p and q can be represented by the intersection of those two propositions. And the negation [not p] of a proposition p is the type exhibited by any state that is not of type p , and hence can be represented by the complement of p . Since the class of propositions will be stipulated to include every subset of the set of states, the set of propositions will be closed under intersection and complement, so we can be sure that conjunctions and negations — and hence, with appropriate definitions, all boolean combinations — of propositions exist. Finally, a proposition p can be said to be *true* at a timepoint t just in case the state of the world at t is of the type represented by p .

Explicitly, then, let $\mathfrak{H} = \langle \mathcal{T}, SOA, States, st \rangle$ be a history.

Definition 3. A *proposition (over \mathfrak{H})* is a subset of *States*. Let $Prop_{\mathfrak{H}}$ be the set of all propositions over \mathfrak{H} , i.e., $Prop_{\mathfrak{H}} = \wp(States)$.

Definition 4. Let $p \in Prop_{\mathfrak{H}}$ be a proposition over \mathfrak{H} and $t \in Time_{\mathcal{T}}$ a timepoint. Then p is *true at t in \mathfrak{H}* — $True-at_{\mathfrak{H}}(p, t)$ — iff $st(t) \in p$.

6. Propositional Process Specification Languages

A propositional process specification language (PPSL) is (with minor changes) a subset of the basic first-order language of PSL together with a new set of terms for denoting propositions and a type of sentence to express that a proposition is true at a given time. We define the notion of such a language by means of the following BNF. Curly brackets are used as delimiters.

⁶This mirrors the standard definition of a proposition in Kripke semantics and its descendants as a set of possible worlds. See especially Kripke’s [3] and Montague’s [6].

6.1 Basic Tokens and Expressions

All PPSLs are based upon a set of basic tokens. Because the symbol ‘+’ has a special meaning in BNF, we will indicate the ASCII character ‘+’ with ‘<plus>’.⁷

```
<upper> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<lower> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<misc>  ::= |<|>|=|(|)|[|]|
<pws>   ::= -|_|<plus>
<wchar> ::= <upper>|<lower>|<digit>|<pws>
```

An *expression* is any string of basic tokens. We define five basic categories of expressions.

```
<lcon> ::= <lower><wchar>*
<ucon> ::= <upper><wchar>*
<ivar> ::= ?<lcon>
<pvar> ::= #<ucon>
<pcon> ::= '<ucon>
```

6.2 Lexicons

A particular first-order PPSL \mathcal{L} is given in terms of a lexicon and a grammar. The lexicon provides the basic “words” of the language. These will be chosen from among the basic categories of expressions. The grammar determines how the lexical elements may be used to build the complex, well-formed expressions of the language. A PPSL lexicon λ consists of eight, pairwise disjoint sets:

- The set consisting of the expressions not, and, or, =>, <=>, forall, and exists;
- A denumerable recursive set $Indvar_\lambda$ of <ivar>s (i.e., expressions derived from the nonterminal <ivar> in the above BNF), known as the *individual variables* of λ ;
- A denumerable recursive set $Propvar_\lambda$ of <pvar>s, known as the *propositional variables* of λ ;
- A recursive set $Indcon_\lambda$ of <lcon>s which includes at least the strings inf- and inf+; these are known as the *individual constants* of λ .
- A recursive set $Propcon_\lambda$ of <pcon>s, known as the *propositional constants* of λ .
- A recursive set $Func_\lambda$ of <lcon>s, known as the *function symbols* of λ , which includes at least the strings beginof and endof;
- A recursive set $Pred_\lambda$ of <ucon>s known as the *predicates* of λ , which includes at least the expressions =, Activity, Event, Timepoint, Occurrence, and Before.

To smooth later developments, we will assume (with no loss of generality) that the class of <pcons>s less $Propcon_\lambda$ is still denumerable.

6.3 Grammars

Given a PPSL lexicon λ , the grammar G_λ based on λ is given in the following BNF. First we introduce non-terminals for each of the classes in our lexicon:

⁷The first character of the category <misc> is a literal space.

$\langle \text{indvar} \rangle ::=$ a member of $Indvar_\lambda$
 $\langle \text{indcon} \rangle ::=$ a member of $Indcon_\lambda$
 $\langle \text{propvar} \rangle ::=$ a member of $Propvar_\lambda$
 $\langle \text{propcon} \rangle ::=$ a member of $Propcon_\lambda$
 $\langle \text{func} \rangle ::=$ a member of $Func_\lambda$
 $\langle \text{pred} \rangle ::=$ a member of $Pred_\lambda$

From the individual variables, individuals constants, and function symbols we build the class of *individual terms*:

$\langle \text{term} \rangle ::= \langle \text{indvar} \rangle | \langle \text{indcon} \rangle | (\langle \text{func} \rangle \langle \text{term} \rangle^+)$

And from propositional variables and constants we build *propositional terms* and, from them and the individual terms, a special class of *propositional sentences*:

$\langle \text{pterm} \rangle ::= \langle \text{propvar} \rangle | \langle \text{propcon} \rangle | [\text{not } \langle \text{pterm} \rangle] |$
 $\quad \quad \quad [\text{and } \langle \text{pterm} \rangle \langle \text{pterm} \rangle^+]$
 $\langle \text{psent} \rangle ::= (\langle \text{pterm} \rangle \langle \text{term} \rangle)$

Finally, we build the general class of sentences:

$\langle \text{sent} \rangle ::= \langle \text{atomsent} \rangle | \langle \text{boolsent} \rangle | \langle \text{quansent} \rangle | \langle \text{psent} \rangle$
 $\langle \text{atomsent} \rangle ::= (\langle \text{pred} \rangle \langle \text{term} \rangle^+) | (= \langle \text{term} \rangle \langle \text{term} \rangle) |$
 $\quad \quad \quad (= \langle \text{pterm} \rangle \langle \text{pterm} \rangle)$
 $\langle \text{boolsent} \rangle ::= (\text{not } \langle \text{sent} \rangle) | (\text{and } \langle \text{sent} \rangle \langle \text{sent} \rangle^+) |$
 $\quad \quad \quad (\text{or } \langle \text{sent} \rangle \langle \text{sent} \rangle^+) | (= \Rightarrow \langle \text{sent} \rangle \langle \text{sent} \rangle)$
 $\quad \quad \quad | (\Leftarrow \Rightarrow \langle \text{sent} \rangle \langle \text{sent} \rangle)$
 $\langle \text{quansent} \rangle ::= (\{\text{forall} | \text{exists}\} (\{\langle \text{indvar} \rangle | \langle \text{propvar} \rangle\}^+) \langle \text{sent} \rangle)$

6.4 Languages

The PPSL \mathcal{L}_λ based on a lexicon λ is the set of expressions that can be derived from the nonterminal $\langle \text{sentence} \rangle$ in the grammar G_λ . The members of \mathcal{L}_λ will also be called (appropriately enough) the *sentences* of \mathcal{L}_λ . A *subsentence* of a given sentence ϕ of \mathcal{L}_λ is a substring of ϕ that is also a sentence. An occurrence of a variable v in a sentence ϕ of \mathcal{L}_λ is *bound* iff that occurrence is in a subsentence of ϕ that is of the form $(\text{exists } v \psi)$ or $(\text{forall } v \psi)$. Otherwise the occurrence is *free*. A sentence ϕ is *closed* iff no variable occurrence in ϕ is free. Let ϕ be a sentence containing zero or more free occurrences of the pairwise distinct variables v_1, \dots, v_n , and let τ_1, \dots, τ_n be n pairwise distinct terms of \mathcal{L}_λ . Then $\phi[v_1/\tau_1, \dots, v_n/\tau_n]$ is the result of replacing all free occurrences of each variable v_i in ϕ with τ_i . A term τ is *free for* a variable v in ϕ iff no (free) occurrence of a variable in τ is bound in $\phi[v/\tau]$.

7. Semantics for Propositional Process Specification Languages

7.1 Interpretations

Let \mathcal{L} be a PPSL. An *interpretation* of \mathcal{L} is a pair $I = \langle \mathfrak{A}, V \rangle$ where \mathfrak{A} is an activity frame and V is a function mapping elements of the lexicon of \mathcal{L} to objects of the appropriate sort. More specifically, where $\mathfrak{A} = \langle \mathfrak{S}, A, E, \text{occ}, \text{beg}, \text{end} \rangle$ and $\mathfrak{S} = \langle T, \text{SOA}, \text{States}, \text{st} \rangle$, let $Ind_{\mathfrak{A}} = \text{Time}_T \cup A \cup E$ and let ε be an expression of \mathcal{L} :

- (1) If ε is an individual constant or variable, then $V(\varepsilon) \in Ind_{\mathfrak{A}}$; in particular, $V(\text{inf } -)$ and $V(\text{inf } +)$ are the “left” and “right” points at infinity of $Time_{\mathcal{T}}$.
- (2) If ε is a propositional variable or constant, then $V(\varepsilon) \in Prop_{\mathfrak{S}}$;
- (3) If ε is a function symbol, then $V(\varepsilon) \in \{f \mid f : \bigcup_{1 \leq n < \omega} Ind_{\mathfrak{A}}^n \longrightarrow Ind_{\mathfrak{A}}\}$; in particular, $V(\text{beginof}) \supseteq beg$ and $V(\text{endof}) \supseteq end$.⁸
- (4) If ε is a predicate, then $V(\varepsilon) \in \wp(\bigcup_{1 \leq n < \omega} Ind_{\mathfrak{A}}^n)$.⁹ In particular, $V(\text{Activity}) = A$, $V(\text{Event}) = E$, $V(\text{Timepoint}) = Time_{\mathcal{T}}$, $V(\text{Occurrence}) = occ$, $V(\text{Before}) = <_{\mathcal{T}}$, and $V(=) = \{\langle b, b \rangle \mid b \in Ind_{\mathfrak{A}} \cup Prop_{\mathfrak{S}}\}$.

For an expression ε of \mathcal{L} , we will usually write ‘ ε^I ’ instead of ‘ $V(\varepsilon)$ ’ to indicate its semantic value under I .

7.2 Truth

A notion of truth for the sentences of \mathcal{L} relative to an interpretation will be determined in the usual way (see, e.g., [5]). A couple of auxiliary definitions will be useful. Let I be an interpretation of \mathcal{L} . If v is an individual variable of \mathcal{L} and $b \in Ind_{\mathfrak{S}}$, or if v is a propositional variable and $b \in Prop_{\mathfrak{S}}$, we say that b is an *appropriate value* for v . Let v_1, \dots, v_n be any variables of \mathcal{L} , and let b_1, \dots, b_n , respectively, be appropriate values for these variables. Then we let $I[v_1/b_1, \dots, v_n/b_n]$ be the interpretation that is just like I except that $v_i^{I[v_1/b_1, \dots, v_n/b_n]} = b_i$ ($1 \leq i \leq n$).

First, we provide denotations for complex terms:

- (1) For functions symbols ξ and individual terms τ_1, \dots, τ_n , $(\xi \tau_1 \dots \tau_n)^I = \xi^I(\tau_1^I, \dots, \tau_n^I)$.
- (2) For any propositional term π , $[\text{not } \pi]^I = Prop_{\mathfrak{S}} - \pi^I$.
- (3) For propositional terms π_1, \dots, π_n , $[\text{and } \pi_1 \dots \pi_n]^I = \bigcap \{\pi_1^I, \dots, \pi_n^I\}$.

Truth is now defined as follows:

- (4) A propositional sentence $(\rho \ \tau)$ is *true in I* iff $True\text{-at}_{\mathfrak{S}}(\rho^I, \tau^I)$.
- (5) An atomic sentence $(\pi \ \tau_1 \dots \tau_n)$ is true in I iff $\langle \tau_1^I, \dots, \tau_n^I \rangle \in \pi^I$.
- (6) A negation $(\text{not } \varphi)$ is true in I iff φ isn’t.
- (7) A conjunction $(\text{and } \varphi \ \psi)$ is true in I iff both φ and ψ are. Similarly for the other binary connectives.
- (8) An existential quantification $(\text{exists } (v_1 \dots v_n) \ \psi)$ is true in I iff there are appropriate objects b_1, \dots, b_n for v_1, \dots, v_n , respectively, such that ψ is true in $I[v_1/b_1, \dots, v_n/b_n]$. Similarly for universal quantifications.

⁸ $Ind_{\mathfrak{A}}^1$ is just $Ind_{\mathfrak{A}}$. For $n > 1$, $Ind_{\mathfrak{A}}^n$ is the n^{th} cartesian product of $Ind_{\mathfrak{A}}$. Note that, by this clause, every function is defined on every n -tuple of individuals. But we can suppose that such functions simply take arbitrary values on ‘irrelevant’ arguments. This will have no effect on soundness or completeness, as the axioms only care about what functions do on relevant arguments.

⁹Thus, like functions, relations do not have a fixed arity, but can have n -tuples of any finite length n in their extensions.

8. Axioms for a PPSL

The axioms for the above model theory consist of fourteen axioms lifted pretty much as is from PSL-core [7], and capture the logical properties of, and relations between, activities, events, and timepoints found in any interpretation. For the sake of brevity, they will not be repeated here. The remaining three axioms concern the logic of propositions:

P1. Propositions are only true at timepoints.

```
(forall (#P ?t)
  (=> (#P ?t)
    (Timepoint ?t)))
```

P2. A conjunctive proposition is true at a timepoint iff each of its conjuncts is true there.

```
(forall (#P #Q ?t)
  (<=> ([and #P #Q] ?t)
    (and (#P ?t) (#Q ?t))))
```

P3. The complement of a proposition is true at a timepoint iff the proposition is not true there.

```
(forall (#P ?t)
  (<=> ([not #P] ?t)
    (not (#P ?t))))
```

Call this system of seventeen axioms *PPSL*.

THEOREM 1. *PPSL is sound with respect to the class of interpretations of \mathcal{L}^* .*

PROOF. Easy. \square

Grüniger has proved the completeness of a system lacking the three propositional axioms with respect to a similar semantics for a language without propositional terms. The algebra of propositions, which is closed under complementation and intersection in the current semantics, corresponds directly to the structure of propositional terms in \mathcal{L} , and truth for propositions at a timepoint appears to be captured fully by axioms 15-17. Hence, it is reasonable to think that Grüniger's completeness proof can be extended to *PPSL* with respect to the class of interpretations of \mathcal{L} .

9. Propositional Process Specifications

The notion of a process specification will be our general, formal representation of the notion of a business or industrial process model. Intuitively, a process specification is a description of the activities that constitute the given process as well as the temporal/causal structure those activities exhibit within the process. It also specifies the preconditions and postconditions that must obtain for an activity to occur at a given point in a realization of the process. The richness of a process specification is a function of the expressive power of the underlying process specification language. PPSL's are rather weak. But at this point, we are primarily interested in the basic idea of a

specification, as well as the notion of a process realization, and so a PPSL is at this point an appropriate linguistic foundation.

A process specification is essentially a collection of “activity role declarations”. More exactly, let \mathcal{L}^+ be a PPSL. We first introduce a new category of expression:

`<num> ::= <digit> | <num><digit>`

Definition 5. An *activity role declaration* (relative to \mathcal{L}^+) is any expression of the following form (we add linefeeds and indentation for readability):

```
(define-activity-role
  :id <num>
  :name <indcon>
  :successors <num>*
  :preconditions <propterm>*
  :postconditions <propterm>*)
```

The value of the `:id` field in an activity role declaration D is known as D 's *role identifier*, and will be referred to as “ id_D ”. The value of the `:name` field of D is known as D 's *activity name*, and will be referred to as “ $name_D$ ”. The values of the `:successor` field are known as D 's *successor identifiers*, and the set of these values will be referred to as “ sc_D ”. Similarly, the sets of values of the `:preconditions` and `:postconditions` fields will be referred to as “ pre_D ” and “ $post_D$ ”, respectively.

The reason that a declaration has both `:name` and `:id` fields is that the same activity can play different roles in the same process. In such cases, we will typically have two or more distinct declarations with the same activity name but with distinct activity identifiers, as it is the identifiers that indicate the distinct structural roles being played by the activity in the overarching process.

Note that activity role declarations are *not* part of a process specification language. They are separate linguistic entities.

Given the notion of an activity role declaration, we define a process specification as follows:

Definition 6. A (*propositional*) *process specification* (relative to a PPSL \mathcal{L}^+) is a set \mathcal{P} of activity role specifications (relative to \mathcal{L}^+) such that

- For distinct $D, D' \in \mathcal{P}$, $id_D \neq id_{D'}$, and
- For all $D \in \mathcal{P}$, if $\kappa \in sc_D$, then there is a $D' \in \mathcal{P}$ such that $\kappa = id_{D'}$.

That is, a process specification is a set of activity role declarations such that (i) no two distinct declarations have the same role identifier, and (ii) every successor identifier of every activity declaration in \mathcal{P} is the role identifier of an activity declaration in \mathcal{P} .

Intuitively, of course, a process specification describes a process. However, in our model theory, there will actually be no semantic object corresponding to a process specification, i.e., there will be no processes *per se*. One might imagine several, more philosophical reasons for this — for example, that processes are artifacts that human beings construct from more ontologically basic activities based on their goals and interests, and hence are not really part of the furniture of the world. Perhaps so. But there

is a simpler, more straightforward reason that we do not introduce processes into our semantics, namely that, on the current approach, there is simply no need to postulate them over and above process specifications, as the specifications themselves have all the structure we need in order to capture salient intuitions about processes.

A process specification \mathcal{P} , then, picks out a collection of activities, defines roles for them, structures those roles according to a *successor* relation, and constrains the occurrence of the activities in those roles by means of preconditions and postconditions. The successor relation can be thought of as type-level temporal precedence; roughly: B is a successor of A if occurrences of A precede occurrences of B in any realization of \mathcal{P} . (Intuitively, a realization of \mathcal{P} is a collection of occurrences of some or all of the activities in \mathcal{P} that satisfy the ordering constraints imposed by the *successor* relation as well as the relevant pre- and postconditions — this notion is defined precisely below.) The preconditions and postconditions of a declaration are simply propositions that must be true at the beginning and ending points, respectively, of an occurrence of the indicated activity in a realization of \mathcal{P} . As noted already, the same activity can play different roles in a process specification. Typically, the preconditions and postconditions associated with the activity in those different roles will differ. Thus, it is more accurate to say that the pre- and postconditions of a declaration are conditions that must obtain in order for the corresponding activity to have an occurrence *at the appropriate point in a realization of \mathcal{P}* .

The notion of process specification realization is made precise in the following section.

10. Process Specification Realizations

10.1 Process Specification Graphs

To clarify the notion of process realization, we need to make clear the idea of the *structure* of the activities in a process specification \mathcal{P} imposed by the `:successors` field in the declarations in \mathcal{P} . Specifically, this field generates a certain sort of graph, which we will call a “process specification graph”:

Definition 7. Let \mathcal{P} be a process specification. The *process specification graph for \mathcal{P}* (or the *graph for \mathcal{P}* , for short) is a directed graph $\langle \mathcal{P}, \rightarrow_{\mathcal{P}} \rangle$, where $\rightarrow_{\mathcal{P}} = \{ \langle D, D' \rangle : D \in \mathcal{P} \text{ and } id_{D'} \in sc_D \}$.

Given a process specification \mathcal{P} and activity role declarations $D, D' \in \mathcal{P}$, say that D' is a *successor of D in \mathcal{P}* just in case $D \rightarrow_{\mathcal{P}} D'$, i.e., just in case D' 's role identifier is one of D 's successor identifiers. Then the graph for a process specification \mathcal{P} is just a graph whose set of nodes is \mathcal{P} itself, and where an arc extends from D to D' if and only if D' is a successor of D in \mathcal{P} . Thus, a process graph simply makes explicit the graphical structure of a process description that is hidden in the `:successors` field of a process description's component declarations.

10.2 Courses of Events

Processes will be realized by courses of events, defined as follows. Let $\mathfrak{H} = \langle \mathcal{T}, SOA, States, st \rangle$ be a history, and let $\mathfrak{A} = \langle \mathfrak{H}, A, E, occ, beg, end \rangle$ be an activity frame over a \mathfrak{H} .

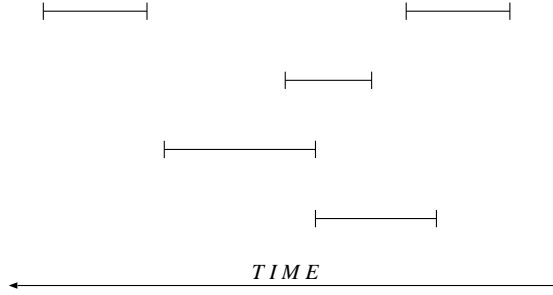


Fig. 1. A collection of occurrences

Definition 8. A course of events \mathcal{E} over \mathcal{A} is a directed graph $\langle S_{\mathcal{E}}, \rightarrow_{\mathcal{E}} \rangle$ such that $S_{\mathcal{E}} \subseteq E$ and, for all $s, s' \in S_{\mathcal{E}}$, $s \rightarrow_{\mathcal{E}} s'$ only if $end(s) \leq_{\mathcal{T}} beg(s')$.

A course of events, then, is a directed graph whose nodes are occurrences, i.e., events. Thus, not just any collection of occurrences counts as a course of events. Rather, a collection must be viewed as temporally (or causally) *structured* in some relevant way — relevance being determined by context, interests, intentions, etc. The only constraint on how the occurrences in a course of events can be structured is that one occurrence s can be an immediate successor of another s' only if s precedes s' in time.

It will be important to note that every course of events, viewed as a graph, is acyclic.

THEOREM 2. Let $\mathcal{E} = \langle S_{\mathcal{E}}, \rightarrow_{\mathcal{E}} \rangle$ be a course of events. Then \mathcal{E} is an acyclic graph.

PROOF. By Definition 8, $s \rightarrow_{\mathcal{E}} s'$ only if $end(s) \leq_{\mathcal{T}} beg(s')$. It follows from the transitivity of $\leq_{\mathcal{T}}$ that if there is a path from s_1 to s_2 in \mathcal{E} , then $end(s_1) \leq_{\mathcal{T}} beg(s_2)$, from which it follows by Definition 2 that $s_1 \neq s_2$, and hence that \mathcal{E} is acyclic. \square

The full motivation for the definition of a course of events will come clearer after we give the definition of process realization, but for now we note simply that, by the acyclic nature of courses of events, paths within a course of events indicate a temporal succession of occurrences; each occurrence s within a path temporally precedes its successor s' , in the sense that s ends no later than the beginning of s' . (If s temporally overlaps s' , they are considered incomparable in the course of events.) Other bases for the partial ordering of a course of events could have been chosen, of course, but this one seems most useful and the one most commonly highlighted in process modeling methods and tools.

It is important to note that an arbitrary collection of occurrences does not in general determine a unique course of events. For instance, consider the collection of occurrences represented by Figure 1. The upper case letters to the left indicate activities, and the lower case letters to their right indicate occurrences of those activities. The lines beneath the lower case letters indicate the durations of those occurrences, and their placement relative to one another indicates their temporal ordering relative to one another. Thus a1 occurs before all other occurrences, c occurs before a2 and d, and b occurs before a2 only. c overlaps b, b overlaps d, and d overlaps

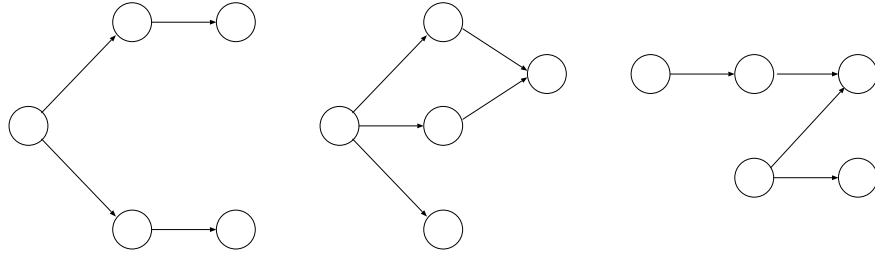


Fig. 2. Same occurrences, distinct courses of events

a2. This collection of occurrences, however, determines several distinct courses of events, as illustrated in Figure 2.

10.3 Process Specification Realizations

The graphical “indeterminacy” of collections of occurrences is critical for representing process realization correctly, as the same set of occurrences can, in the guises of different courses of events, be mapped onto many different processes. The idea here is that, depending on one’s context, interests, etc., the same set of occurrences can be seen as relevantly (e.g., causally) structured in a variety of ways, and hence as exhibiting a variety of general patterns depending on the structure chosen.

This insight enables us to define a reasonable semantics for process specifications in terms of the notion of a process specification realization.

Definition 9. Let \mathcal{L} be a PPSL. Let \mathcal{P} be a process specification relative to \mathcal{L} and let $\langle \mathcal{P}, \rightarrow_{\mathcal{P}} \rangle$ be the graph for \mathcal{P} . Let $\mathfrak{A} = \langle \mathfrak{H}, A, E, occ, beg, end \rangle$ be an activity frame, and let $I = \langle \mathfrak{A}, V \rangle$ be an interpretation for \mathcal{L} such that, for every declaration $D \in \mathcal{P}$, $(name_D)^I \in A$. Finally, let $\mathcal{E} = \langle S_{\mathcal{E}}, \rightarrow_{\mathcal{E}} \rangle$ be a course of events over \mathfrak{A} . Then we say that \mathcal{E} realizes \mathcal{P} (relative to I) iff there is a homomorphism¹⁰ $h : S_{\mathcal{E}} \rightarrow \mathcal{P}$ such that, for all $s \in S_{\mathcal{E}}$,

- $(name_{h(s)})^I = occ(s)$; and
- For all $\pi \in pre_{h(s)}$, π^I is true at $beg(s)$, and for all $\pi \in post_{h(s)}$, π^I is true at $end(s)$.

Thus, a course of events \mathcal{E} realizes a process specification \mathcal{P} (relative to an \mathcal{L} interpretation I) just in case \mathcal{E} can be mapped into \mathcal{P} in a structure-preserving way such that (i) each element e of \mathcal{E} is an occurrence of the activity A named in the declaration D to which it is mapped, and (ii) the preconditions and postconditions specified for A in D hold at the beginning and ending points of e , respectively.¹¹

¹⁰That is, for all $s, s' \in S_{\mathcal{E}}$, if $s \rightarrow_{\mathcal{E}} s'$, then $h(s) \rightarrow_{\mathcal{P}} h(s')$.

¹¹The observant reader might have noticed that activity role declarations, and hence full-blown process specifications, only make use of the individual constants and propositional terms of \mathcal{L} . However, a more complex notion of process specification — to be defined in a further paper — can include sentences of \mathcal{L} that put additional “global” constraints on the models in which realizations can occur. Such sentences could be thought of as expressing process model “metadata”.

11. Conclusion

A PPSL language itself provides the basis for the activity role declarations that constitute a process specification — which corresponds formally to the notion of a process model. The axioms for a PPSL provide a simple theory of the activities described by a process specification, and the events that can jointly constitute an instance of that specification. The notion of a course of events provides a robust representation of the sort of thing that can serve as an instance of a process model. In particular, there is a clear definition for the truth of a proposition (hence of a constraint in an activity role declaration) in a course of events. Finally, the notion of a process specification realization provides a robust representation of what it is for a course of events to count as an instance of a process model. We therefore believe that the language, axioms, and model theory described in this paper provide a promising formal foundation for process modeling.

Acknowledgements

Menzel's work on this paper was largely conducted at Knowledge Based Systems, Inc. of College Station, Texas (<http://www.kbsi.com>) on a project funded by a Phase II Small Business Innovation Research (SBIR) grant from the National Institute of Standards and Technology.

References

- [1] van Benthem, J. F. A. K., *The Logic of Time*, D. Reidel Publishing Co., Dordrecht, 1983.
- [2] Hayes, P. J., "A Catalog of Temporal Theories," Tech report UIUC-BI-AI-96-01, University of Illinois, 1995.
- [3] Kripke, S., "Semantic Considerations on Modal Logic," *Acta Philosophica Fennica* **24** (1963), 83–94.
- [4] Marti-Oliet, N., and J. Meseguer, "From Petri Nets to Linear Logic," *Mathematical Structures in Computer Science* **1(1)**, 69-101, 1991.
- [5] Mendelson, E., *Introduction to Mathematical Logic*, 4th edition, International Thomson Publishers, New York, 1997.
- [6] Montague, R., "On the Nature of Certain Philosophical Entities," *The Monist* **53** (1969), 159–194. Reprinted in R. Montague, *Formal Philosophy*, Yale University Press, New Haven, CT, 1974, pp. 148–187.
- [7] The PSL Working Group, "PSL Ontology — Current Theories and Extensions," <http://www.mel.nist.gov/psl/psl-ontology>.
- [8] Sandewall, E., *Features and Fluents: The Representation of Knowledge about Dynamical Systems*, Clarendon Press, Oxford, 1994.
- [9] Wittgenstein, L., *Tractatus Logico-Philosophicus*, Routledge & Kegan Paul, London, 1922.