APQL: A Process-model Query Language

Arthur H.M. ter Hofstede[1,2,3], Chun Ouyang[1,3], Marcello La Rosa[1,3], Liang Song[4,5,6,7], Jianmin Wang[4,6,7], and Artem Polyvyanyy[1]

[1] Queensland University of Technology, Brisbane, Australia
{a.terhofstede,c.ouyang,m.larosa}@qut.edu.au
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
[3] NICTA, Australia
[4] School of Software, Tsinghua University, Beijing, China
{songliang08}@mails.thu.edu.cn,jianmin@mail.thu.edu.cn
[5] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[6] Key Lab for Information System Security, Ministry of Education, Beijing, China
[7] National Laboratory for Information Science and Technology, Beijing, China

Abstract. As business process management technology matures, organisations acquire more and more business process models. The resulting collections can consist of hundreds, even thousands of models and their management poses real challenges. One of these challenges concerns model retrieval where support should be provided for the formulation and efficient execution of business process model queries. As queries based on only structural information cannot deal with all querying requirements in practice, there should be support for queries that require knowledge of process model semantics. In this paper we formally define a process model query language that is based on semantic relationships between tasks in process models and is independent of any particular process modelling notation.

1 Introduction

With the increasing maturity of business process management, more and more organisations need to manage large numbers of business process models, and among these may be models of high complexity. Business process models constitute valuable intellectual property capturing the way an organisation conducts its business. Processes may be defined along the entire value chain and over time a business may gather hundreds and even thousands of business process models. As an example consider Suncorp, one of the largest Australian insurers. Over the years, Suncorp have gone through a number of organizational mergers and acquisitions, as a result of which the company has accumulated over 3,000 process models for the various lines of insurance. In this context, support for business process retrieval, e.g. for the purposes of process reuse or process standardization, is a challenging proposition.

While several query languages exist that can be used to retrieve process models from a repository, e.g. BPMN-Q [1] or BP-QL [2, 3], these languages are based on syntactic relationships between tasks and not on semantic relationships between them.

While in a process graph, a task A may follow a task B this does not mean that during execution task B will always follow sometime after task A. Let us consider for example the two process models in Fig. 1. These models represent two variants of a business process for opening bank accounts in the BPMN notation [4]. Each variant consists of a number of tasks (represented by rectangles) and dependencies between these tasks. Arcs represent sequential dependencies, while diamonds represent decisions (if there is one incoming arc and multiple outgoing arcs) and simple merges (if there are multiple incoming arcs and one outgoing arc). These two variants could capture the way an account is opened in two different states in which the company operates, and could be part of a collection of various process models for all states in which the bank operates. Now let us assume that an analyst needs to find all states which require an assessment of the customer's credit history when opening an account. In this case, by only using the structural relationships between tasks, we cannot discern between the two variants, i.e. we would retrieve them both, since in both models there is at least a path from task "Receive customer request" to task "Analyse customer credit history". However, if we consider semantic relationships, we can see that task "Analyse customer credit history" always follows task "Receive customer request" in all instances of the first process variant, but this is not the case for one instance

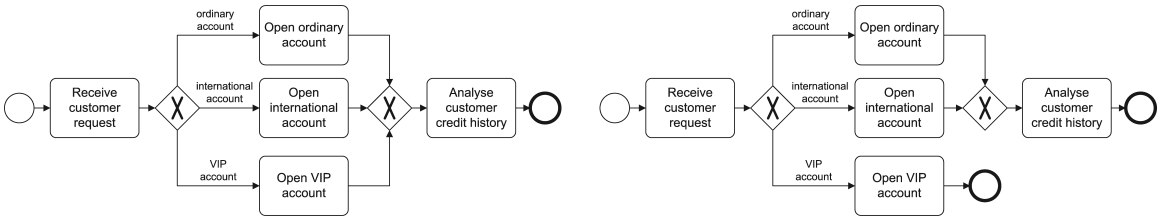

Fig. 1: Two variants of a business process for opening bank accounts.

of the second variant (the one where task "Open VIP account" is run). Thus we can correctly exclude the second variant from the results of our query, and return the first variant only.

A process model retrieval language based on semantic relationships is indeed in line with process execution, e.g. through a workflow management system. One challenge though, when it comes to determining semantic relationships between tasks, is how to determine these relationships in a feasible manner, i.e. without suffering from the well-known state space explosion problem.

In light of the above, in this work we aim to address the development of an *expressive* business process model query language. We do so by proposing a new query language for process model repositories, namely "A Process-model Query Language" (APQL). This language relies on a number of basic temporal relationships between tasks which can be composed to obtain complex relationships between them. These predicates allow us to express queries that can discriminate over single process instances or task instances.

Since the language relies on temporal relationships between tasks, it is independent of a specific business process modelling notation. In order to demonstrate its feasibility, in [5], we worked on a concrete realization of this language in the context of Petri nets. To this end, we adopted the theory of unfoldings [6], a well-known theory for dealing with the state space explosion problem; in particular, we explored the result on derivation of finite initial parts of unfoldings called *complete finite prefixes* [6, 7]. We proposed an index structure, called the *directed bigraphs*, as extension of the complete prefix unfolding of a net system and relied on this structure to compute temporal relationships between Petri net transitions. Later, flaws were detected by Artem Polyvyanyy in the algorithms for deriving temporal relationships between Petri net transitions from the directed bigraphs. While the directed bigraphs of a net system represent the system's behaviour, due to the fact that they are an extension of the complete finite prefix of the net system, a single representation of behaviour is by far not sufficient to suit all the proposed algorithms. For instance, one can validate that the $alwoccur$ predicate, cf. Algorithm 7 in [5], when applied to the net system in Fig. 5(a) in [5] and its transition I, computes to false erroneously. Note that the problem of finding an effective general purpose data structure for representing the behaviour of a concurrent system is an area of active research with many existing solutions often targeting the "convenient" validation of some limited set of attributes [8]. To address the identified flaws, one either needs to account for another representation of behaviour or to replace several of the algorithms with ones that are more expensive.

Hence, in this updated report, we only cover the APQL language with a slightly revised formalization and leave out its realization in Petri nets. The remainder of this report focuses on the definition of APQL including both its syntax (Section 2) and semantics (Section 3). Examples are included to assist in understanding of the language definition. We also briefly discuss related work (Section 4) and finally conclude the paper (Section 5).

2 The Syntax of APQL

In this section we first look at the design rationale for APQL and provide an informal introduction to this language. The syntax of APQL is presented in the form of an abstract syntax, the advantages of which over a concrete syntax have been espoused by Meyer [9]. In essence, in an abstract syntax we can avoid committing ourselves prematurely to specific choices for keywords or to the order of various statements. For illustration purposes, we provide sample queries based on the abstract syntax of APQL.

2.1 Informal Description

APQL is designed as a process model retrieval language that is independent of the actual process modelling language used. This is important as in practice a variety of modelling languages is used (e.g. BPMN, EPCs) and the language should be generally applicable. Another important starting point is the fact that process models have a semantics and it should be possible to exploit this semantics when querying. For example, let t_i (where $i = 1, 2, ..., n$) be a task identifier, while task t_1 may follow task t_2 in process model r (i.e. there is a directed path from task t_1 to task t_2), it may be the case that due to the presence of certain splits and joins, task t_2 cannot actually be executed after task t_1. Hence, a language based on the syntax (i.e. structure) of a process model is not always powerful enough.

In order to achieve language independence we define a set of 20 basic predicates to capture, in business process models, the occurrences of tasks as well as the semantic relationships between tasks. Below, the first two predicates capture the occurrence of a task in some or every execution of a given process model.

1. $posoccur(t_1, r)$: there exists some execution of process model r where at least one instance of t_1 occurs.
2. $alwoccur(t_1, r)$: in every execution of process model r, at least one instance of t_1 occurs.

The next two predicates capture the exclusive and concurrent relationships between task occurrences. Note that these two predicates do not assume that an instance of t_1 or t_2 should eventually occur.

3. $exclusive(t_1, t_2, r)$: in every execution of process model r, it is never possible that an instance of t_1 and an instance of t_2 both occur.
4. $concur(t_1, t_2, r)$: t_1 and t_2 are not causally related, and in every execution of process model r, if an instance of t_1 occurs then an instance of t_2 occurs and vice versa.

Then we consider various forms of causal relationship between task occurrences. The relationship can be *precedence* or *succession*, where one task may occur *immediately* or *eventually* preceding or succeeding another task. It may hold for any or $every$ occurrence of the tasks in *some* or *every* process execution. Combining all these considerations results in 16 forms of causal relationships which are captured by the remaining 16 basic predicates as follows.

Let Φ be one of the following intermediate predicates,

1. $succ_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is eventually succeeded by an instance of t_2 (e.g. $...t_1...t_2...$).
2. $succ_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is eventually succeeded by an instance of t_2 (e.g. $t_1...t_1...t_2$).
3. $pred_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is eventually preceded by an instance of t_2 (e.g. $...t_2...t_1...$).
4. $pred_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is eventually preceded by an instance of t_2 (e.g. $t_2...t_1...t_1$).
5. $isucc_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is immediately succeeded by an instance of t_2 (e.g. $...t_1t_2...$).
6. $isucc_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is immediately succeeded by an instance of t_2 (e.g. $t_1t_2...t_1t_2$).
7. $ipred_{any}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and is immediately preceded by an instance of t_2 (e.g. $...t_2t_1...$).
8. $ipred_{every}(t_1, t_2, i)$: in process execution i, at least one instance of t_1 occurs and every instance of t_1 is immediately preceded by an instance of t_2 (e.g. $t_2t_1...t_2t_1$).

then

- $\Phi^{\forall}(t_1, t_2, r)$: $\Phi(t_1, t_2, i)$ holds for every process execution i of process model r, i.e. $\Phi(t_1, t_2, i)$ *always* holds in process r.
- $\Phi^{\exists}(t_1, t_2, r)$: there *exists* some process execution i of process model r where $\Phi(t_1, t_2, i)$ holds, i.e. it is *possible* that $\Phi(t_1, t_2, i)$ holds in process r.

Note that the above predicates are all defined in terms of the execution of a process.

2.2 Abstract Syntax

We define the abstract syntax of APQL using the notation introduced in [9]. In APQL a query is a set of *Assignments* combined with a *Predicate*.

$$\begin{aligned} Query &\triangleq s : Assignments; p : Predicate \\ Assignments &\triangleq Assignment^* \end{aligned}$$

The result is those process models that satisfy the *Predicate*. An *Assignment* assigns a *TaskSet* to a variable and when evaluating the *Predicate* every variable is replaced by the corresponding *TaskSet* (via the identifier of such task set).

$$\begin{aligned} Assignment &\triangleq v : Varname; ts : TaskSet \\ Varname &\triangleq identifier \end{aligned}$$

*TaskSet*s can be enumerations of tasks or they can be defined in terms of other *TaskSet*s either by *Construction* or by *Application*. A *TaskSet* can also be defined through a variable, a *TaskSetVar*.

$$TaskSet \triangleq SetofTasks \mid Construction \mid Application \mid TaskSetVar$$

A *Task* can be defined as a combination of a *TaskLabel* (which is a string) and a *SimDegree* (which is a real number). The idea is that one may be interested in *Tasks* of which the task label bears (at least) a certain degree of similarity to a given activity name. There are a number of definitions in the literature concerning label similarity and for a concrete implementation of the language one has to commit to one of these.

$$\begin{aligned} SetofTasks &\triangleq Task^+ \\ Task &\triangleq l : TaskLabel; d : SimDegree \\ TaskLabel &\triangleq string \\ SimDegree &\triangleq real[0..1] \end{aligned}$$

A *TaskSetVar* is simply a variable that carries the identifier of the set of the tasks. Such a task set may be used in assignments.

$$TaskSetVar \triangleq identifier$$

A *TaskSet* can be composed from other *TaskSet*s through the application of the well-known set operators such as *union*, *difference*, and *intersection*. Another way to construct a *TaskSet* is by the application of a *TaskCompOp* (i.e. one of the basic predicates introduced earlier, but now interpreted as a function) on another *TaskSet*. In that case we have to specify whether we are interested in the tasks that have that particular relation with *all* or with *any* of the tasks in the *TaskSet*. For example, an expression with *TaskSet* S, *TaskCompOp* *PosSuccAny* (i.e. $succ_{any}{}^{\exists}$) and with *AnyAll* indicator *all*, should yield those tasks that any instance of such a task succeeds an instance of each individual task in S in some process execution.

$$\begin{aligned} Construction &\triangleq ts_1, ts_2 : TaskSet; o : Set\_Op \\ Set\_Op &\triangleq Union \mid Difference \mid Intersection \\ Application &\triangleq ts : TaskSet; o : TaskCompOp; a : AnyAll \\ TaskCompOp &\triangleq Exclusive \mid Concur \mid \\ &\quad AlwSuccAny \mid AlwSuccEvery \mid AlwPredAny \mid AlwPredEvery \mid \\ &\quad PosSuccAny \mid PosSuccEvery \mid PosPredAny \mid PosPredEvery \mid \\ &\quad AlwISuccAny \mid AlwISuccEvery \mid PosISuccAny \mid PosISuccEvery \mid \\ &\quad AlwIPredAny \mid AlwIPredEvery \mid PosIPredAny \mid PosIPredEvery \\ AnyAll &\triangleq Any \mid All \end{aligned}$$

A *Predicate* can consist of a simple *TaskPos*, with the intended semantics what the basic predicate *posoccur* specifies; a *TaskAlw*, with the intended semantics what the basic predicate *alwoccur* specifies; a *TaskRel*, with

the intended semantics that all process models satisfying that particular relation should be retreived; or, it can be recursively defined as a binary or unary *Predicate* through the application of logical operators.

$$
\begin{aligned}
Predicate &\triangleq TaskPos \mid TaskAlw \mid TaskRel \mid Bin\_Predicate \mid Un\_Predicate \\
Bin\_Predicate &\triangleq o : BinLogOp; p_1, p_2 : Predicate \\
Un\_Predicate &\triangleq o : UnLogOp; p : Predicate \\
BinLogOp &\triangleq And \mid Or \\
UnLogOp &\triangleq Not \\
TaskPos &\triangleq l : TaskLabel; d : SimDegree \\
TaskAlw &\triangleq l : TaskLabel; d : SimDegree
\end{aligned}
$$

A *TaskRel* can be 1) a relationship between a *Task* and a *TaskSet* checking whether that *Task* occurs in that *TaskSet* (*TaskInTaskSet*), 2) a relationship between a *Task* and a *TaskSet* and involving a *TaskCompOp* and an *AnyAll* indicator determining whether the *Task* has the *TaskCompOp* relationship with any/all *Tasks* in the *TaskSet* (*Task_TaskSet*), 3) a relationship between two *Tasks* involving a *TaskCompOp* predicate determining whether for the two *Tasks* that predicate holds (*Task_Task*), 4) a relationship between two *TaskSets* involving a *TaskCompOp* and an *AnyAll* indicator determining whether the *Tasks* in those *TaskSets* all have that *TaskCompOp* relationship to each other or whether for each *Task* in the first *TaskSet* there is a corresponding *Task* in the second *TaskSet* for which the relationship holds (*Elt_TaskSet_TaskSet*), or 5) a relationship between two *TaskSets* determined by a set comparison operator (*Set_TaskSet_TaskSet*).

$$
\begin{aligned}
TaskRel &\triangleq TaskInTaskSet \mid Task\_TaskSet \mid \\
&\quad\; Task\_Task \mid Elt\_TaskSet\_TaskSet \mid \\
&\quad\; Set\_TaskSet\_TaskSet \\
TaskInTaskSet &\triangleq t : Task; ts : TaskSet \\
Task\_TaskSet &\triangleq t : Task; ts : TaskSet; \\
&\quad\; o : TaskCompOp; a : AnyAll \\
Task\_Task &\triangleq t_1, t_2 : Task; o : TaskCompOp \\
Elt\_TaskSet\_TaskSet &\triangleq ts_1, ts_2 : TaskSet; o : TaskCompOp; \\
&\quad\; a : AnyAll \\
Set\_TaskSet\_TaskSet &\triangleq ts_1, ts_2 : TaskSet; o : SetCompOp \\
SetCompOp &\triangleq Identical \mid Subsetof \mid Overlap
\end{aligned}
$$

2.3 Sample Queries

In this section we will show some sample queries and how they can be captured in APQL in order to further illustrate the language. The sample queries, specified in natural language, are listed below (and numbered Q_1 to Q_{10}). In these queries, by default the value for the *AnyAll* identifier, when applicable, is *all*, and by default the value for the SimDegree is 1. Fig. 2 shows the grammar trees for queries Q_1 to Q_6, while Fig. 3 shows the grammar trees for queries Q_7 to Q_{10}. Note that in the following A to L are task labels (i.e. activity names).

Q_1**.** Select all process models where task A occurs in some process execution and task B occurs in every process execution.
Q_2**.** Select all process models where in every process execution task A may occur before task D.
Q_3**.** Select all process models where in every process execution task A always occurs before task D.
Q_4**.** Select all process models where in some process execution task A may occur before task B and task B may occur before task K.
Q_5**.** Select all process models where in some process execution task A always occurs before task B.
Q_6**.** Select all process models where task B occurs in parallel with task C.
Q_7**.** Select all process models where task B occurs in parallel with task C and where task A occurs in parallel with task H.

Q_8. Select all process models where in every process execution task B and task C never occur together.

Q_9. Select all process models where in every process execution the immediate predecessors of task H are among the immediate successors of task B.

Q_{10}. Select all process models where in some process execution the immediate predecessors of task H may occur after the common immediate successors of task B and task C.

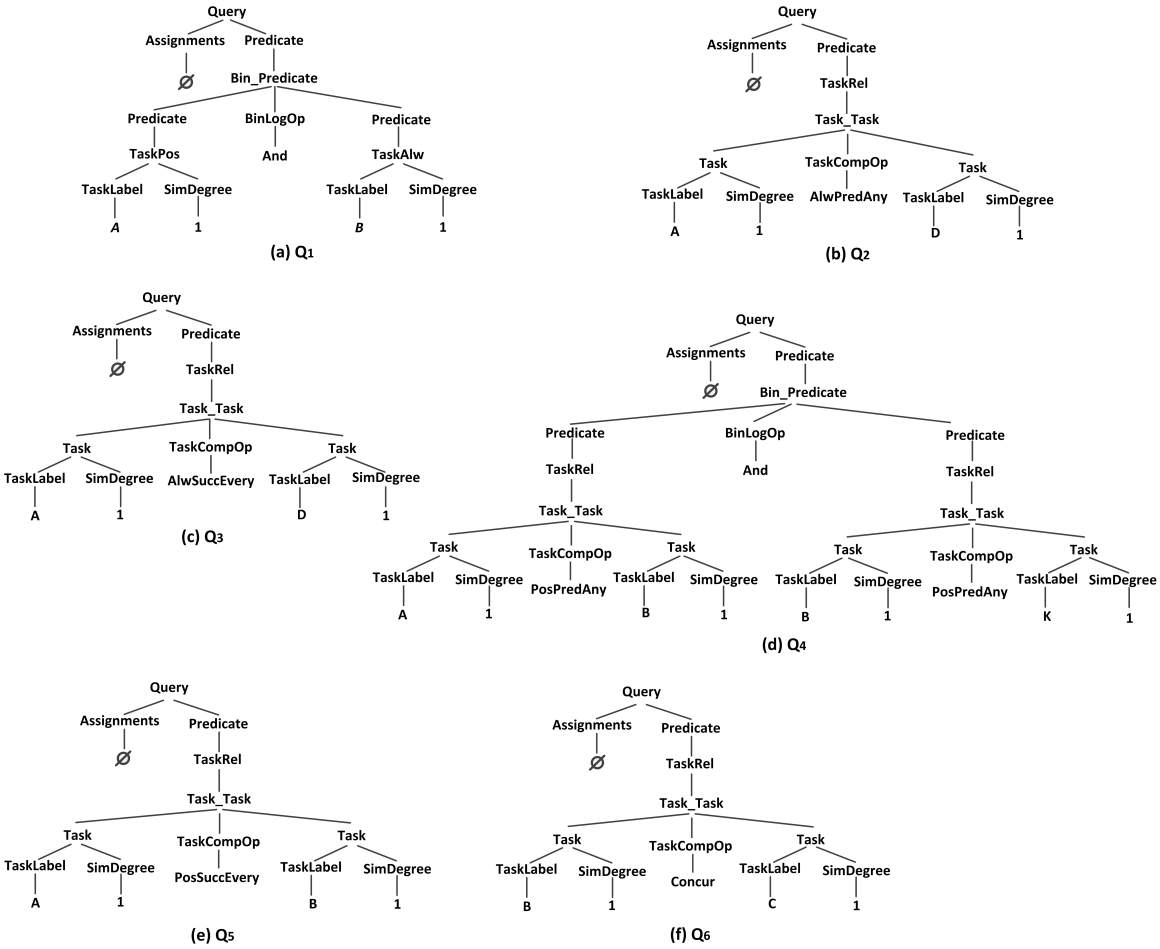

Fig. 2: The APQL grammar trees of sample queries $Q_1 - Q_6$

3 APQL Semantics

In this section, we use denotational semantics to formally describe the semantics of APQL. For each nonterminal T we introduce a semantic function M_T which defines the meaning of the nonterminal in terms of its parts. The notation that we adopt throughout this section is the notation used in [9].

First, we introduce some auxiliary notation in order to facilitate the subsequent definition of the semantics.

Definition 1 (overriding union). *The overriding union of $f : X \to Y$ by $g : X \to Y$, denoted as $f \oplus g$, is defined by $g \cup f \backslash \{(x, f(x)) \mid x \in dom(f) \cap dom(g)\}$.*

With the set of 20 basic predicates defined in the previous section, we use $\mathbb{BP}_u$ to denote the set of two *unary predicates* $\{posoccur, alwoccur\}$ which specify unary task relations, and similarly we use $\mathbb{BP}_b$ to denote the set of other 18 *binary predicates* which specify binary task relations. The following two definitions introduce

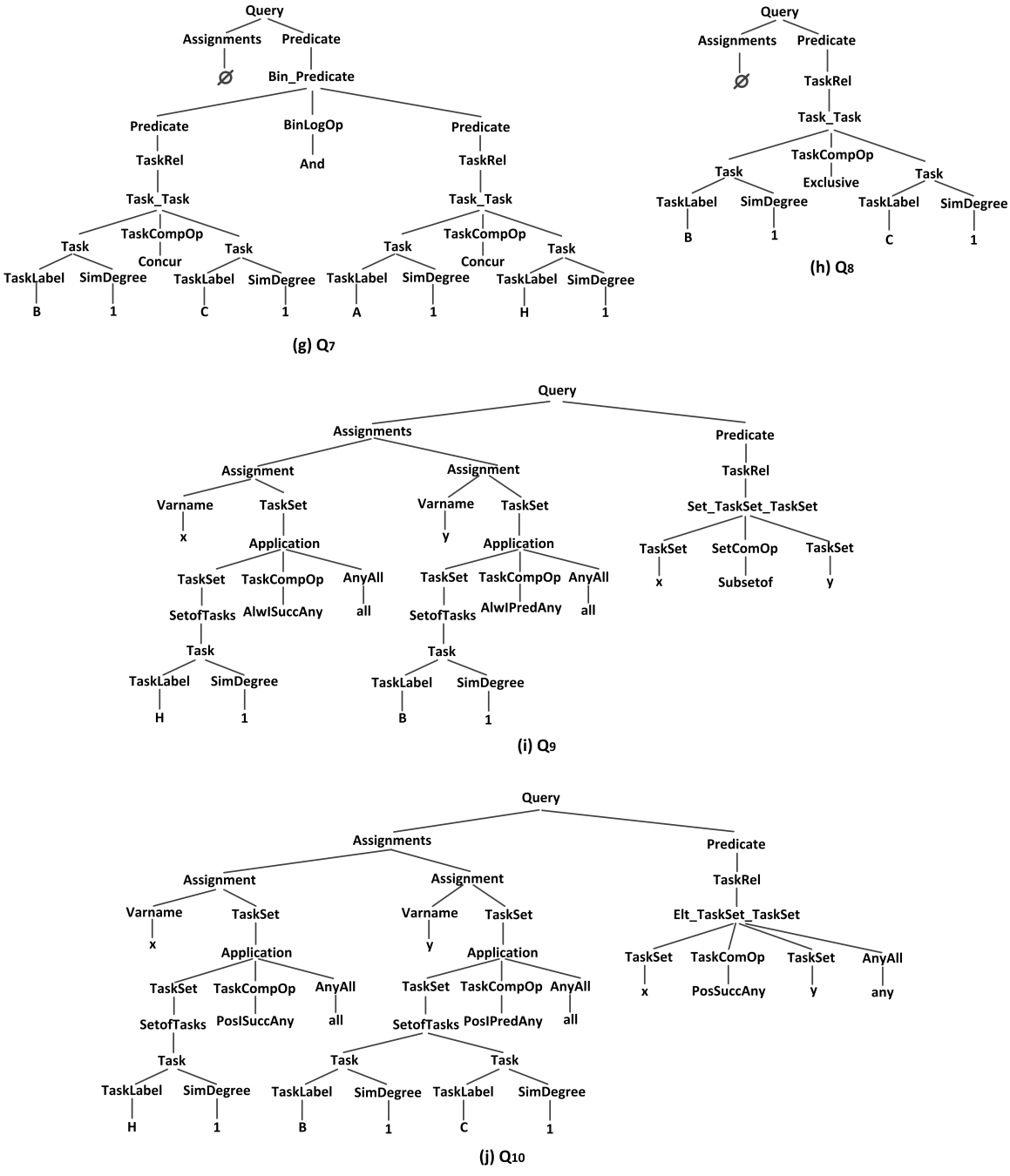

Fig. 3: The APQL grammar trees of sample queries $Q_7 - Q_{10}$

a higher order predicate that takes as input a unary or binary predicate, respectively. Note that the semantics of each predicate (ϕ/ψ) is language independent. For a task t in process model N, $L_N(t)$ specifies the label of t. A process model may have *silent tasks* which do not capture any task or activity in the process but are used for modelling purposes, e.g. a silent task used to capture an internal action that cannot observed by external uses. For a silent task t, we let $L(t) = \tau$.

Definition 2. *Let N be a process model and T the set of tasks in N, for $t_1, t_2 \in T$ and $\phi \in \mathbb{BP}_b$*

$$\mathsf{ref}^{\phi}(t_1, t_2, N) = \begin{cases} \phi(t_1, t_2, N) \textbf{ if } L_N(t_1) \neq \tau \wedge L_N(t_2) \neq \tau \\ FALSE \qquad \textbf{otherwise} \end{cases}$$

i.e. the relation ϕ should hold between t_1 and t_2 in net N if both are non-silent tasks.

Definition 3. *Let N be a process model and T the set of tasks in N, for $t_1 \in T$ and $\psi \in \mathbb{BP}_u$*

$$\mathsf{ref}^{\psi}(t_1, N) = \begin{cases} \psi(t_1, N) \textbf{ if } L_N(t_1) \neq \tau \\ FALSE \quad \textbf{otherwise} \end{cases}$$

i.e. the relation ψ should hold for t_1 in net N if t_1 is a non-silent task.

As queries may use variables, we must know their values during query evaluation. A *Binding* is an assignment of task sets to variables:

$$Binding \triangleq ProcessModel \times Varname \rightarrowtail 2^{Task}$$

Queries are applied to a repository of process models, i.e.

$$Repository \triangleq 2^{ProcessModel}$$

A process model r consists of a collection of tasks T_r. For each task t in process model r we can retrieve its label as $L_r(t)$. Label similarity can be determined through the function Sim, where $Sim(l_1, l_2)$ determines the degree of similarity between labels l_1 and l_2 (which yields a value in the range [0,1]). Note that Sim is a parameter of the approach in which case one can choose his/her own similarity notion and returns the similarity evaluation result to this parameter.

The query evaluation function M_{Query} takes a query and a repository as input and yields the collection of process models in that repository that satisfy the query:

$$M_{Query} : Query \times Repository \rightarrow 2^{ProcessModel}$$

This function is defined as follows:

$$M_{Query}[q : Query, R : Repository] \triangleq M_{Predicate}(q.p, R, M_{Assignments}(q.s, R, \varnothing))$$

The evaluation of the query evaluation function depends on the evaluation of the predicate involved and the assignments involved. When evaluating a sequence of assignments we have to remember the values that have been assigned to the variables involved. Inititally this set of assignments is empty.

$$M_{Assignments} : Assignments \times Repository \times Binding \rightarrow Binding$$

The result of a sequence of assignments is a binding where the variables used in the assignments are bound to sets of tasks. If a variable was already assigned a set of tasks in an earlier assignment in the sequence the latest assignment takes precedence over the earlier assignment.

$M_{Assignments}[s : Assignments, R : Repository, B : Binding] \quad \triangleq$
  if $\neg(s.TAIL).EMPTY$ **then**
    $M_{Assignments}(s.TAIL, R, B \oplus M_{Assignment}(s.FIRST, R, B))$
  else B

The result of an individual assignment is also a binding where the variable is linked to the set of tasks involved.

$M_{Assignment} : Assignment \times Repository \times Binding \rightarrow Binding$

$M_{Assignment}[a : Assignment, R : Repository, B : Binding)] \quad \triangleq$
$\{((r, a.v), M_{TaskSet}(a.ts, R, B)(r)) \mid r \in R\}$

A predicate can be evaluated in the context of a repository and a binding and the result is a set of process models from that repository.

$$M_{Predicate} : Predicate \times Repository \times Binding \rightarrow 2^{ProcessModel}$$

A predicate which is a task yields all process models in the repository that contain a task sufficiently similar to that task (with respect to the task label and similarity degree). A predicate which is a relationship between tasks (i.e. a *TaskRel*) yields all the process models that satisfy this relationship. A disjunction yields the union of the process models of the predicates involved, while a conjunction yields the intersection. The negation of a predicate yields the process models in the repository that do not satisfy the predicate.

$M_{Predicate}(p : Predicate, R : Repository, B : Binding) \quad \triangleq$
case p **of**
$TaskPos \Rightarrow \{r \in R \mid \exists t \in T_r[Sim(p.l, L_r(t)) \geq p.d \wedge posoccur(t, r)]\}$
$TaskAlw \Rightarrow \{r \in R \mid \exists t \in T_r[Sim(p.l, L_r(t)) \geq p.d \wedge alwoccur(t, r)]$
$TaskRel \Rightarrow M_{TaskRel}(p, R, B)$
$Bin\_Predicate \Rightarrow$
case $p.o$ **of**
$And \Rightarrow M_{Predicate}(p.p_1, R, B) \cap M_{Predicate}(p.p_2, R, B)$
$Or \Rightarrow M_{Predicate}(p.p_1, R, B) \cup M_{Predicate}(p.p_2, R, B)$
end
$Un\_Predicate \Rightarrow R \backslash M_{Predicate}(p, R, B)$
end

A *TaskRel* in the context of a repository and a binding yields a set of process models in that repository.

$$M_{TaskRel} : TaskRel \times Repository \times Binding \rightarrow 2^{ProcessModel}$$

A *TaskRel* can be used to determine whether a task in a process model occurs in a given task set, whether a given basic predicate holds between a task in a process model and one or all tasks in a given task set, whether a given basic predicate holds between tasks in a process model, whether a given basic predicate holds between two or between all tasks in two given task sets, or whether a given set comparison relation holds between two given task sets.

$M_{TaskRel}(tr : TaskRel, R : Repository, B : Binding) \quad \triangleq$
case tr **of**
$TaskInTaskSet \Rightarrow$
$\{r \in R \mid \exists v \in M_{TaskSet}(tr.ts, R, B)(r)[Sim(tr.t.l, L_r(v)) \geq tr.t.d]\}$
$Task\_TaskSet \Rightarrow$
case $tr.a$ **of**
$Any \Rightarrow \{r \in R \mid \exists t_1 \in T_r \; \exists t_2 \in M_{TaskSet}(tr.ts, R, B)(r)$
$[Sim(tr.t.l, L_r(t_1)) \geq tr.t.d \wedge rel^{tr.o}(t_1, t_2, r)]\}$
$All \Rightarrow \{r \in R \mid \exists t_1 \in T_r \; \forall t_2 \in M_{TaskSet}(tr.ts, R, B)(r)$
$[Sim(tr.t.l, L_r(t_1)) \geq tr.t.d \wedge rel^{tr.o}(t_1, t_2, r)]\}$
end
$Task\_Task \Rightarrow \{r \in R \mid \exists v_1, v_2 \in T_r[Sim(tr.t_1.l, L_r(v_1)) \geq tr.t_1.d \wedge$
$Sim(tr.t_2.l, L_r(v_2)) \geq tr.t_2.d \wedge rel^{tr.o}(v_1, v_2, r)]\}$
$Elt\_TaskSet\_TaskSet \Rightarrow$
case $tr.a$ **of**
$Any \Rightarrow \{r \in R \mid \exists t_1 \in M_{TaskSet}(tr.ts_1, R, B)(r)$

$\exists t_2 \in M_{TaskSet}(tr.ts_2, R, B)(r)[rel^{tr.o}(t_1, t_2, r)]\}$
$All \Rightarrow \{r \in R \mid \forall t_1 \in M_{TaskSet}(tr.ts_1, R, B)(r)$
$\forall t_2 \in M_{TaskSet}(tr.ts_2, R, B)(r)[rel^{tr.o}(t_1, t_2, r)]\}$
end
$Set\_TaskSet\_TaskSet \Rightarrow$
case $tr.o$ **of**
$Identical \Rightarrow$
$\{r \in R \mid M_{TaskSet}(tr.ts_1, R, B)(r) = M_{TaskSet}(tr.ts_2, R, B)(r)\}$
$Subsetof \Rightarrow$
$\{r \in R \mid M_{TaskSet}(tr.ts_1, R, B)(r) \subseteq M_{TaskSet}(tr.ts_2, R, B)(r)\}$
$Overlap \Rightarrow$
$\{r \in R \mid M_{TaskSet}(tr.ts_1, R, B)(r) \cap M_{TaskSet}(tr.ts_2, R, B)(r) \neq \varnothing\}$
end
end

A *TaskSet* within the context of a repository and a binding yields a mapping which assigns to each process model in the repository the collection of tasks within that model that satisfy the restriction imposed by the *TaskSet*.

$$M_{TaskSet} : TaskSet \times Repository \times Binding \rightarrow (ProcessModel \rightarrow 2^{Task})$$

When a *TaskSet* is a set of tasks, then for each process model the result is the set of tasks within that process model that are sufficiently similar to at least one of the tasks in that *TaskSet*. When the *TaskSet* is a variable, then the evaluation is similar except that the task set used is the task set currently bound to that variable. *TaskSets* can also be formed through *Construction* (where the set operators union, difference, and intersection are used) or *Application* (where task sets are formed through set comprehension, i.e. they are defined through properties that they have - these properties relate to the basic predicates).

$M_{TaskSet}(tks : TaskSet, R : Repository, B : Binding) \quad \triangleq$
case tks **of**
$SetofTasks \Rightarrow$
$\{(r, \{t \in T_r \mid \exists_{1 \leq i \leq tks.LENGTH}[Sim(tks(i).l, L_r(t)) \geq tks(i).d]\}) \mid r \in R\}$
$TaskSetVar \Rightarrow$
$\{(r, X) \mid r \in R\}$ **where**
$X = \begin{cases} B(r, tks) \textbf{ if } (r, tks) \in dom(B) \\ \varnothing \qquad \textbf{otherwise} \end{cases}$
$Construction \Rightarrow$
case $tks.o$ **of**
$Union \Rightarrow$
$\{(r, M_{TaskSet}(tks.ts_1, R, B)(r) \cup M_{TaskSet}(tks.ts_2, R, B)(r)) \mid r \in R\}$
$Difference \Rightarrow$
$\{(r, M_{TaskSet}(tks.ts_1, R, B)(r) \backslash M_{TaskSet}(tks.ts_2, R, B)(r)) \mid r \in R\}$
$Intersection \Rightarrow$
$\{(r, M_{TaskSet}(tks.ts_1, R, B)(r) \cap M_{TaskSet}(tks.ts_2, R, B)(r)) \mid r \in R\}$
end
$Application \Rightarrow$
case $tks.a$ **of**
$Any \Rightarrow$
$\{(r, \{t \in T_r \mid \exists v \in M_{TaskSet}(tks.ts, R, B)(r)[rel^{tks.o}(t, v, r)]\}) \mid r \in R\}$
$All \Rightarrow$
$\{(r, \{t \in T_r \mid \forall v \in M_{TaskSet}(tks.ts, R, B)(r)[rel^{tks.o}(t, v, r)]\}) \mid r \in R\}$
end
end

Semantics of Sample Queries In order to illustrate the formal semantics of APQL, a number of process models, represented in BPMN, are presented in Fig. 4. For each sample query of Section 2.3 and for each

model it is indicated whether the model is part of the answer to the query (in that case the box corresponding to the query is ticked otherwise the box is not ticked). Note that in some models tasks with the same label occur (e.g. there are two tasks labeled A in model f), in which case, APQL will treat these tasks as same tasks during query evaluation.[8]

4 Related work

Mindful of the importance of query languages for business process models, the Business Process Management Initiative (BPMI) proposed to define a standard process model query language in 2004[9]. While such a standard has never been published, two major research efforts have been dedicated to the development of query languages for process models. One is known as BP-QL [3], a graphical query language based on an abstract representation of BPEL and supported by a formal model of graph grammars for processing of queries. BP-QL can be used to query process specifications written in BPEL rather than possible executions, and ignores the run-time semantics of certain BPEL constructs such as conditional execution and parallel execution.

The other effort, namely BPMN-Q [1, 10], is also a visual query language which extends a subset of the BPMN modelling notation and supports graph-based query processing. Similarly to BP-QL, BPMN-Q only captures the structural (i.e. syntactical) relationships between tasks, and not their behavioral interrelationships. In [11], the authors explore the use of an information retrieval technique to derive similarities of activity names, and develop an ontological expansion of BPMN-Q to tackle the problem of querying business processes that are developed with different terminologies. A framework of tool support for querying process model repositories using BPMN-Q and its extensions is presented in [12].

APQL presents three distinguishing features compared to the above languages. First, its abstract syntax and semantics have been purposefully defined to be independent of a specific process modelling language (such as BPEL or BPMN). This will allow APQL and its query evaluation technique to be implemented for a variety of process modelling languages. Second, APQL can express all possible temporal-ordering relations (precedence/succession, concurrence and exclusivity) between individual tasks, between an individual task and a set of tasks as well as between different sets of tasks. Third, APQL querying constructs need to be evaluated over the execution semantics of process models, rather than their structural relationships. In fact, structural characteristics alone are not able to capture all possible order relations among tasks which can occur during execution, in particular with respect to cycles and task occurrences.

In earlier work [13], we provided an initial attempt at defining a query language based on execution semantics of process models. The language was written in linear temporal logic (LTL) and only supported precedence/succession relations among individual tasks (not sets of tasks).

In addition to the development of a specific process model query language, other techniques are available in the literature which can be useful for querying process model repositories. In [14, 15] the authors focus on querying the content of business process models based on metadata search. VisTrails system [16] allows users to query scientific workflows by example and to refine workflows by analogies. WISE [17] is a workflow information search engine which supports keyword search on workflow hierarchies. In [18] the authors use graph reduction techniques to find a match to the query graph in the process graph for querying process variants, and the approach however works on acyclic graphs only. In [19–21], a group of similarity-based techniques have been proposed which can be used to support process querying. In previous work, we designed a technique to query process model repositories based on an input Petri net [22]. Finally, in [23], the notion of behavioural profile of a process model is defined, which captures dedicated behavioural relations like exclusiveness or potential occurrence of activities. However, these behavioural relations are derived from the structure of a process model. Thus, for the reasons mentioned above, behavioral profiles only provide an approximation of a process model's behavior, whereas we can precisely determine whether a process model satisfies or not a given query, since we work at the behavioral level.

[8] This is expected because APQL is query language rather than a process modelling language.

[9] http://www.bpmi.org/downloads/BPMI_Phase_2.pdf

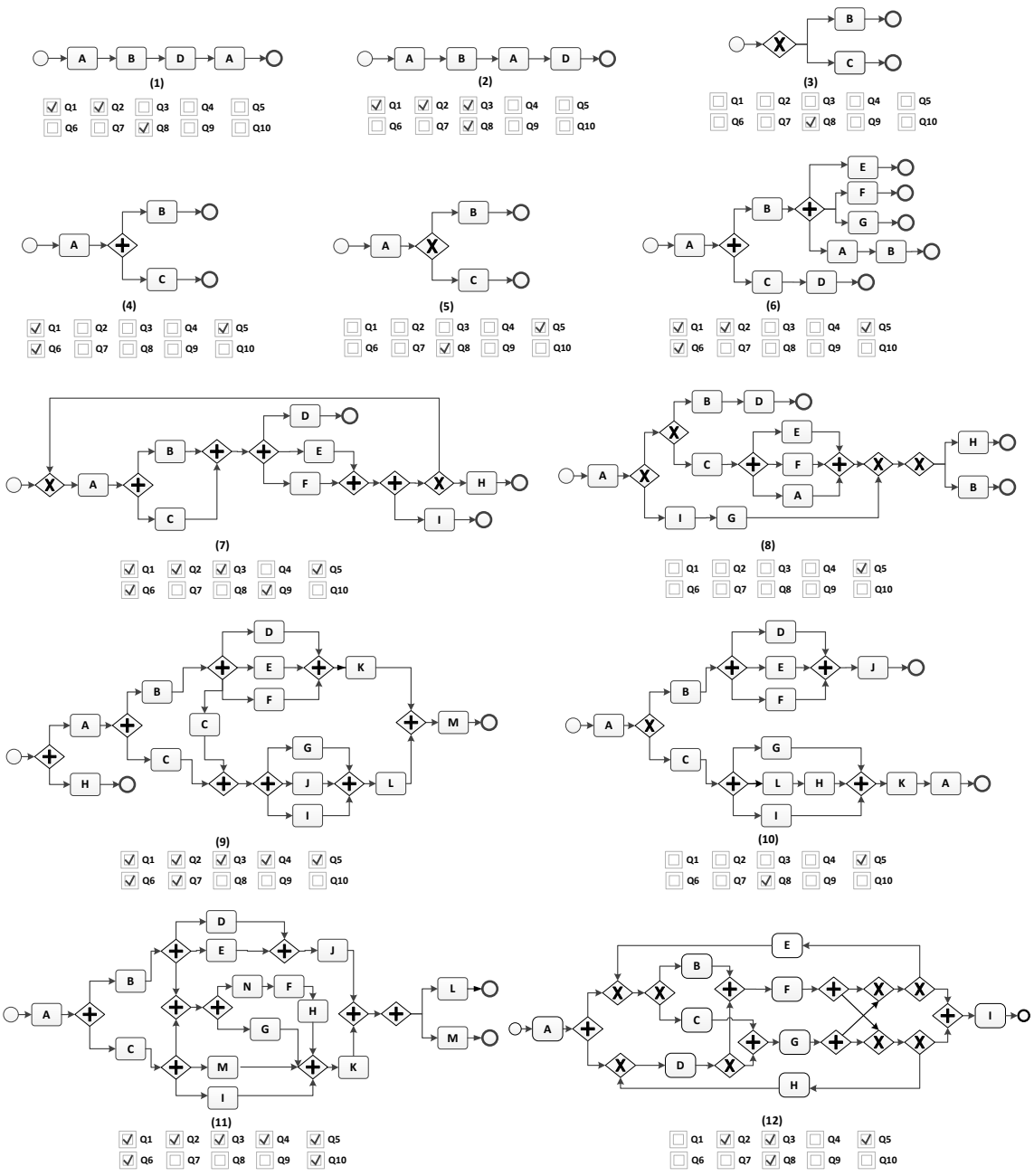

Fig. 4: A list of BPMN business process models and evaluation of sample queries $Q_1 - Q_{10}$ in Section 2.3 over these processes.

5 Conclusions

This paper contributes an innovative language, namely APQL, for querying process model repositories. APQL provides three main advantages over the state of the art. First, the language is expressive since it allows users to specify all possible order relationships among tasks or sets thereof. Second, the language is precise, since APQL queries are defined for evaluation over process model behavior, while existing query languages only support structural process characteristics. Third, the language's syntax and semantics are defined independently of any specific process modeling language.

Currently APQL only focuses on the control flow perspective of business process models. In the future, we will extend the language definition in order to include other process perspectives such as data and participating resources. Moreover, we plan to run structured interviews with domain experts to assess the overall ease of use and usefulness of APQL.

Acknowledgements

Song and Wang are supported by the National Basic Research Program of China (2009CB320700), the National High-Tech Development Program of China (2008AA042301), the Project of National Natural Science Foundation of China (90718010), and the Program for New Century Excellent Talents in University of China. ter Hofstede and La Rosa are supported by the ARC Linkage Grant "Facilitating Business Process Standardization and Reuse" (LP110100252). In 2010 and 2011, ter Hofstede was a senior visiting scholar of Tsinghua University. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

We would also like to acknowledge the valuable detailed review comments from an anonymous reviewer on the original version of this report [5]. In particular, taking into account his/her comments led to an improved definition of the APQL language in this updated report.

References

1. A. Awad. BPMN-Q: A language to query business processes. In M. Reichert, S. Strecker, and K. Turowski, editors, *Enterprise Modelling and Information Systems Architectures - Concepts and Applications , Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8-9, 2007*, volume P-119 of *LNI*, pages 115–128. GI, 2007.
2. C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 343–354. ACM, 2006.
3. C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes with BP-QL. *Inf. Syst.*, 33(6):477–507, September 2008.
4. OMG. *Business Process Model and Notation (BPMN) ver. 2.0*, January 2011. Available via `http://www.omg.org/spec/BPMN/2.0`.
5. Liang Song, Wang Jianmin, Arthur H.M. ter Hofstede, Marcello La Rosa, Chun Ouyang, and Lijie Wen. A semantics-based approach to querying process model repositories, December 2011. QUT ePrint 47791. Available via `http://eprints.qut.edu.au/47791/`.
6. K. L. McMillan. A technique of state space search based on unfolding. *Form. Method. Syst. Des.*, 6(1):45–65, 1995.
7. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Form.Method.Syst.Des.*, 20(3):285–310, 2002.
8. Javier Esparza and Keijo Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. EATCS Monographs in Theoretical Computer Science. Springer, 2008.
9. B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice-Hall, 1990.
10. A. Awad, G. Decker, and M. Weske. Efficient compliance checking using BPMN-Q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pages 326–341, Berlin, Heidelberg, 2008. Springer-Verlag.

11. A. Awad, A. Polyvyanyy, and M. Weske. Semantic querying of business process models. In *12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, September 15-19, 2008, Munich, Germany*, pages 85–94. IEEE Computer Society, 2008.
12. S. Sakr and A. Awad. A framework for querying graph-based business process models. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 1297–1300, New York, NY, USA, 2010. ACM.
13. L. Song, J. Wang, L. Wen, W. Wang, S. Tan, and H. Kong. Querying process models based on the temporal relations between tasks. In *Workshops Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOCW 2011, Helsinki, Finland, August 29 - September 2, 2011*, pages 213–222. IEEE Computer Society, 2011.
14. J. Vanhatalo, J. Koehler, and F. Leymann. Repository for business processes and arbitrary associated metadata. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 426–431, Vienna, Austria, 2006. Springer.
15. A. Wasser, M. Lincoln, and R. Karni. ProcessGene Query - a tool for querying the content layer of business process models. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 1–8, Vienna, Austria, 2006. Springer.
16. C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and re-using workflows with vistrails. In J. T. Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1251–1254, New York, NY, USA, 2008. ACM.
17. Q. Shao, P. Sun, and Y. Chen. Wise: A workflow information search engine. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 1491–1494. IEEE, 2009.
18. R. Lu and S. W. Sadiq. Managing process variants as an information resource. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 426–431, Vienna, Austria, 2006. Springer.
19. Wil M. P. van der Aalst, Ana Karla A. de Medeiros, and A. J. M. M. Weijters. Process equivalence: Comparing two process models based on observed behavior. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2006.
20. M. Ehrig, A. Koschmider, and A. Oberweis. Measuring similarity between semantic business process models. In John F. Roddick and Annika Hinze, editors, *Conceptual Modelling 2007, Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Victoria, Australia, January 30 - February 2, 2007, Proceedings*, volume 67 of *CRPIT*, pages 71–80, Darlinghurst, Australia, 2007. Australian Computer Society, Inc.
21. B. Dongen, R. Dijkman, and J. Mendling. Measuring similarity between business process models. In Z. Bellahsene and M. Léonard, editors, *Advanced Information Systems Engineering, 20th International Conference, CAiSE, Montpellier, France, June 16-20, 2008, Proceedings*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464, Berlin, Heidelberg, 2008. Springer-Verlag.
22. T. Jin, J. Wang, N. Wu, M. La Rosa, and A.H.M. ter Hofstede. Efficient and accurate retrieval of business process models through indexing - (short paper). In Robert Meersman, Tharam S. Dillon, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2010 - Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, October 25-29, 2010, Proceedings, Part I*, volume 6426 of *Lecture Notes in Computer Science*, pages 402–409. Springer, 2010.
23. M. Weidlich, J. Mendling, and M. Weske. Efficient consistency measurement based on behavioral profiles of process models. *Software Engineering, IEEE Transactions on*, 37(3):410 –429, 2011.