Specialisation and Generalisation of Processes

Christine Choppy[a], Jörg Desel[*,b], Laure Petrucci[a]

[a] LIPN, CNRS UMR 7030, Université Paris 13, Université Sorbonne Paris Cité, France
[b] FernUniversität in Hagen, Germany

Abstract. *In conceptual data modelling, one of the most important abstraction concepts is specialisation, with generalisation being the converse. Although there are already some approaches to define generalisation for behavioural modelling as well, there is no generally accepted notion of process generalisation. In this paper, we introduce a general definition of process specialisation and generalisation. Instead of concentrating on a specific process description language, we refer to labelled partial orders. For most process description languages, behaviour (if defined at all) can be expressed by means of this formalism. We distinguish generalisation from aggregation, and specialisation from instantiation. For Petri nets, we provide examples and suggest associated notations. Our generalisation notion captures various previous approaches to generalisation, for example ignoring tasks, allowing alternative tasks and deferring choices between alternative tasks. A general guideline is that a more general process contains less features and/or less information than a more specific one.*

Keywords. Process Modelling • Process Generalisation • Process Specialisation

1 Introduction

Specialisation, and its counterpart generalisation, is an important concept of conceptual data modelling which has been known for many years in database research (Smith and Smith 1977). The core idea of generalisation is to combine object types, which share common attributes, to a more general supertype, which only has the common attributes, whereas each more specific type inherits these attributes from the supertype and has its own, private attributes. We can also adopt a top-down view instead of a bottom-up one: starting with an object type, we might identify subtypes such that the objects in each of the subtypes share common additional attributes, which might be meaningless for other objects. The initial type can be specialised to these subtypes, and then common attributes and additional attributes of the subtypes can be distinguished. Such a specialisation can cover the supertype (every object belongs to at least one subtype) or not, and it can divide the supertype into disjoint subtypes (no object belongs to more than one subtype) or not.

Instead of attributes of objects, generalisation and specialisation also apply to classes and their methods in object-oriented modelling, and in an even broader scope, to arbitrary features that are inherited from the more general to the more specific component. Generalisation and specialisation are very important abstraction concepts in models that clarify mutual dependencies. They are the prerequisite for reuse of system components, and they allow to avoid redundancy. For the maintenance of systems and of models, only these concepts support that common features of different system or model components can be handled at a single place, instead of considering various copies.

Whereas generalisation and specialisation are abstraction techniques that have their own graphical representation in conceptual data modelling,

* Corresponding author.
E-mail. joerg.desel@fernuni-hagen.de
This work was achieved while the second author was visiting professor at University Paris 13.

there are only few suggestions how to apply comparable concepts to process models. On the other hand, the same arguments as above would apply to a generalisation and specialisation concept in behavioural modelling as well. At several places, this demand was explicitly expressed, see e.g. (Frank and Laak 2002).

Petri nets are often proposed for conceptual modelling of system behaviour, e.g. by (Mayr et al. 2007). In more recent work (Mayr and Michael 2012), a Petri net-like language is presented for conceptual models of human behaviour. Actually, this language has primitives for operations and their pre- and post-conditions. Therefore, a natural semantics for this language is based on causal relations between operation occurrences, as occurrence nets are a well established semantics for Petri nets. Basically, each run of a Petri net model can thus be represented by a partially ordered set of occurrences of operations or transitions.

There are already some papers dealing with specialisation for particular process models, such as Petri nets. In (Wyner and Lee 2005) a particular extension of Petri nets is suggested, based on (Lee and Wyner 2003). Unfortunately, it remains unclear how this approach can be transferred to other modelling languages. Another relevant approach is given in (Aalst and Basten 2002); however, this paper also restricts to a particular language. Moreover, it emphasises inheritance and change instead of specialisation and generalisation. In particular, the inheritance is based on *blocking* and *hiding* of transitions whereas in our notion blocking a transition will turn out to be a specialisation and hiding a transition will turn out to be a generalisation.

Another important difference between our approach and previous work is that we consider partial order behaviour of process models instead of strings and sequential automata. As mentioned above, this choice is justified by the observation that many process languages emphasise causality and concurrency between activities, which is most appropriately represented by means of partial orders.

Our approach should be applicable to as many process modelling languages as possible. Therefore, we do not start with any particular syntactical description but rather concentrate on the behaviour of models. As mentioned before, the behavioural notion used in this paper is given by partially ordered sets of activities representing runs, labelled by respective tasks that are expected to appear in a process model. Although this formalism is quite easy to understand, it needs some involved technical notations that will be carefully introduced in Section 2. In our examples, we use Petri nets as a modelling language, but this does not restrict our approach to this particular language. We use only elementary Petri nets and expect the reader to understand them without any formal definition.

Our core criteria for a specialisation definition are that specialisation means to add something (features, tasks, information) to a process model and that everything valid for a more general process model should also hold for its specialisation. For example, adding a task to a process model results in the addition of respective activities in the runs, where the remaining structure of the runs is not changed. If, according to a process model, two tasks can be executed independently (in any order or concurrently), then in a specialisation an order between these tasks can be specified. This results in runs that are also runs of the more general system. If two tasks can be executed alternatively, then a specialisation might add the information to decide which of the tasks occurs; in this case, some of the runs of the more general model are ruled out in the specialisation. All these relations can be formulated by means of the respective sets of runs, and will be the subject of Section 3.

We also identify more subtle specialisation relations that refer to the branching behaviour of process models; a more special model could have "earlier" information about the decision of a choice than a more general model, although their respective sets of runs are identical. For a motivating example and our solution to this phenomenon, see Section 4.

Section 5 comes back to the previous work on specialisation of Petri nets mentioned above. We show that these concepts can be viewed as a special case of our approach.

Finally, Section 6 provides a summary and relates our notion of specialisation to other abstraction concepts such as refinement and instantiation.

2 Basic Setting

In this paper, no formal definition of a process model is given, since we do not stick to any particular process modelling language. Instead, some characteristics of a process model are expected to be defined: its set of tasks; its runs containing task executions; and a precedence relation between task executions in runs. This precedence relation is formally given as a partial order, i. e. a transitive and irreflexive relation (in accordance with Workflow Management Coalition n.d., we use the term activity for a single execution of a task in a process run). Our definition resembles the definition of Partial Languages as defined by (Grabowski 1981) and Pomsets as defined by (Pratt 1986).

Given a process model P with a set of tasks T, its behaviour is defined by its set of possible runs.

Definition 1 (Process run) A *process run* (or just *run*) π of a process model P with set of tasks T is given by

- a finite set of *activities* A_π,
- partially ordered by a *precedence relation* $\rightsquigarrow_\pi$, and
- a *mapping* $\lambda_\pi : A \to T$ mapping each activity *to a task*.

Remark 1 We have decided to consider finite runs only because infinite runs of process models are only of theoretical interest. However, each infinite run can be approximated by an infinite sequence of finite runs, where each run is a proper prefix (defined below) of its successor.

If an activity x is mapped to a task $\lambda(x)$ then it represents an occurrence of $\lambda(x)$. Activities have to be distinguished from tasks since there can be more than one occurrence of a task in one run, with different precedence relations to other activities.

Definition 2 (Immediate precedence)
Given a process run π, the immediate precedence relation is denoted by $\rightarrow_\pi := \rightsquigarrow_\pi \setminus (\rightsquigarrow_\pi \circ \rightsquigarrow_\pi)$.

Remark 2 Since the set of activities A_π is finite, $\rightsquigarrow_\pi$ is the transitive closure of $\rightarrow_\pi$.

In graphical representations, we depict the relation $\rightarrow_\pi$ by means of directed arcs, i.e., we provide the Hasse-diagram of partial orders. Two activities x and y satisfy $x \rightsquigarrow_\pi y$ if and only if there is a nonempty path leading from x to y.

Example 1
Figure 1 depicts a process run π such that : $A_\pi = \{1, 2, 3, 4, 5\}$, $\rightarrow_\pi = \{(1, 2), (2, 3), (3, 4), (1, 5)\}$, and $\lambda_\pi(1) = a$, $\lambda_\pi(2) = b$, $\lambda_\pi(3) = c$, $\lambda_\pi(4) = b$, $\lambda_\pi(5) = b$. Thus, activities are denoted by numbers and tasks by lowercase letters.

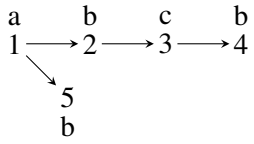


Figure 1: A process run π

Note that a process run can feature several branches and that a given task can occur at different places in the process run. In the above example, task b can occur after another occurrence of task b, and both occur concurrently to a third one. However, they are associated with different activities, respectively 4 and 2, which are ordered by the precedence relation, and 5.

In case of sequential semantics of processes, where each run is represented by a sequence of occurring tasks, we can easily distinguish the first, the second, etc. occurrence of a single task. Each sequence can be viewed as a mapping from the set $\{1, 2, 3, \ldots, \mathit{length}(\mathit{run})\}$ to the set of tasks. For partial-order semantics, there is no such

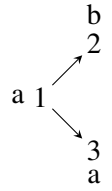
unique representation of a run. Hence the "same behaviour" can be represented by runs which only differ w.r.t. the activities. Such runs are said to be isomorphic.

Definition 3 (Isomorphic runs) Let π and π' be two process runs of the same process model P with set of tasks T. Then, π' is *isomorphic* to π if there is a bijection $\beta\colon A_\pi \to A_{\pi'}$ such that

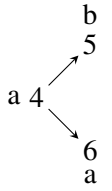
1. $\forall x, y \in A_\pi\colon\ x \rightarrow_\pi y \iff \beta(x) \rightarrow_{\pi'} \beta(y)$ and
2. $\forall x \in A_\pi\colon\ \lambda_\pi(x) = \lambda_{\pi'}(\beta(x))$.

Clearly, process run isomorphism is an *equivalence relation*. If isomorphic process runs are not distinguished, any representative of the equivalence class can be used.

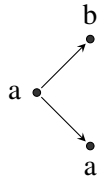
Example 2 Figure 2 schematises the isomorphism between two process runs. Process run π (Figure 2(a)) and process run π' (Figure 2(b)) yield the same graph of tasks when abstracting away the activities.



(a) Process run π



(b) Process run π'



(c) Task occurrences

Figure 2: A process run π' isomorphic to a process run π

In general, activities are formalised differently even though they have the same meaning in terms of tasks. Sometimes it is still necessary to distinguish activities within a process run in order to be able to refer to precise occurrences of a task. When this distinction is irrelevant in a figure, a $\bullet$ will be used instead of the actual activity (see e.g. Figure 2(c)).

3 Linear Time Specialisation

The meaning of $a \rightsquigarrow_\pi b$ is that $\lambda(a)$ is executed before $\lambda(b)$ in run π. If activities a and b are not ordered by means of $\rightsquigarrow_\pi$ then they occur in any order (this order is not captured by our notion of run) or concurrently. Each *linearisation* of a run π, obtained by adding elements to the order relation $\rightsquigarrow_\pi$, yields another run, which we consider more *specific* than π. In turn, each run *generalises* its set of linearisations.

The following definition not only compares the precedence relations between activities but also the respective sets of activities of two runs. It formalises the observation that a run of a more specific process definition might contain more details than a run of a less specific process definition. Hence the runs in the following definition can belong to distinct processes.

Definition 4 (Process run specialisation) Let P and P' be two process models with sets of tasks T and T', respectively. A process run π' of P' *specialises* a process run π of P (denoted by $\pi' \geq \pi$) if there is an injective mapping $\mu\colon A_\pi \to A_{\pi'}$ such that

1. $\forall x, y \in A_\pi\colon\ x \rightarrow_\pi y \Rightarrow \mu(x) \rightsquigarrow_{\pi'} \mu(y)$ and
2. $\forall x \in A_\pi\colon \lambda_\pi(x) = \lambda_{\pi'}(\mu(x))$.

Remark 3 As a consequence of Definition 4, if $x \rightsquigarrow_\pi y$ then $\mu(x) \rightsquigarrow_{\pi'} \mu(y)$.

Each activity in the process run π has a corresponding activity in the specialised process (by the

mapping μ), mapped to the corresponding task (2). Moreover, for each precedence relation in π, there is a corresponding sequence of precedences in π' (1).

Example 3 Figure 3 shows a process run π' specialising a process run π together with the mapping μ.

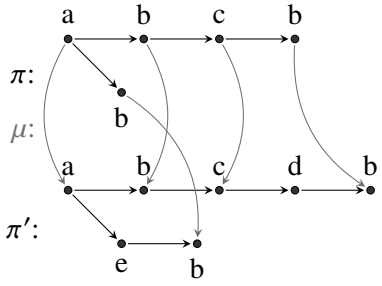


Figure 3: Process run π' specialises process run π

Up to an isomorphism, a process run π' of P' specialises a process run π of P if and only if $A_\pi \subseteq A_{\pi'}$ and $\rightsquigarrow_\pi \subseteq \rightsquigarrow_{\pi'}$. This justifies the "$\geq$"-notion for specialisations. It is formalised by the following lemma.

Lemma 1 *Let P and P' be two process models with sets of tasks T and T', and runs π and π', respectively. We have $\pi' \geq \pi$ if and only if there exists a process run $\overline{\pi}$ such that*

1. *$\overline{\pi}$ is isomorphic to π,*
2. *$A_{\overline{\pi}} \subseteq A_{\pi'}$,*
3. *$\rightarrow_{\overline{\pi}} \subseteq \rightsquigarrow_{\pi'}$, and*
4. *$\forall x \in A_{\overline{\pi}} : \lambda_{\overline{\pi}}(x) = \lambda_{\pi'}(x)$.*

Proof ($\Rightarrow$) Assume that π' specialises π by means of mapping μ. Process run $\overline{\pi}$ is constructed as follows:

$A_{\overline{\pi}} = \{x' \in A_{\pi'} \mid \exists x \in A_\pi : x' = \mu(x)\}$;

$\rightsquigarrow_{\overline{\pi}} = \{(x', y') \in A^2_{\overline{\pi}} \mid \exists (x, y) \in \rightsquigarrow_\pi : \mu(x) = x' \wedge \mu(y) = y'\}$;

$\forall x \in A_{\overline{\pi}} : \lambda_{\overline{\pi}}(x) = \lambda_{\pi'}(x)$.

($\Leftarrow$) Assume that $\overline{\mu} : A_{\overline{\pi}} \rightarrow A_{\pi'}$ is an isomorphism. Mapping $\mu : A_\pi \rightarrow A_{\pi'}$ is constructed by: $\forall x \in A_\pi, \mu(x) = \overline{\mu}(x)$.

A process run π' specialises π if π boils down to the same tasks graph as process $\overline{\pi}$ (4), $\overline{\pi}$ is isomorphic to π (1), but with activities in π' (2). The precedence relation in $\overline{\pi}$ respects the order relation in π' (3).

Definition 5 (Linear time specialisation) A process model P' is a *linear time specialisation* of a process model P if for each run π' of P', there exists a run π of P such that $\pi' \geq \pi$.

The process model P is then said to be a *a linear time generalisation* of P'.

As mentioned in the introduction, every valid statement about the more general process should hold for the specialised process as well. In a more formal setting, which is beyond the scope of this paper, any formula of an appropriate logic that evaluates to *true* for the general process, should be evaluated to *true* for the specialised process as well. Since, in this section, we concentrate on a relation based on the runs only, this logic would be Linear Time and based on partial orders, such as the one of (Bhat and Peled 1998). The idea is that a less general process contains more features/information than a more general one but inherits all properties from the more general one.

Representative examples of specialisation/generalisation:

Table 1 depicts some representative examples of generalisation and specialisation. The process models are given as Petri nets, and their runs are pictured as well. For each example, both a specialised and a general version are given. The example is named after the specialisation characteristic. Hence, the first one *forces an activity* since it prevents an activity associated with task b from occurring. The second example *adds an activity* associated with task b. Finally, the last example imposes a sequential *ordering between activities* that are concurrent in the general model, namely those associated with tasks b and c.

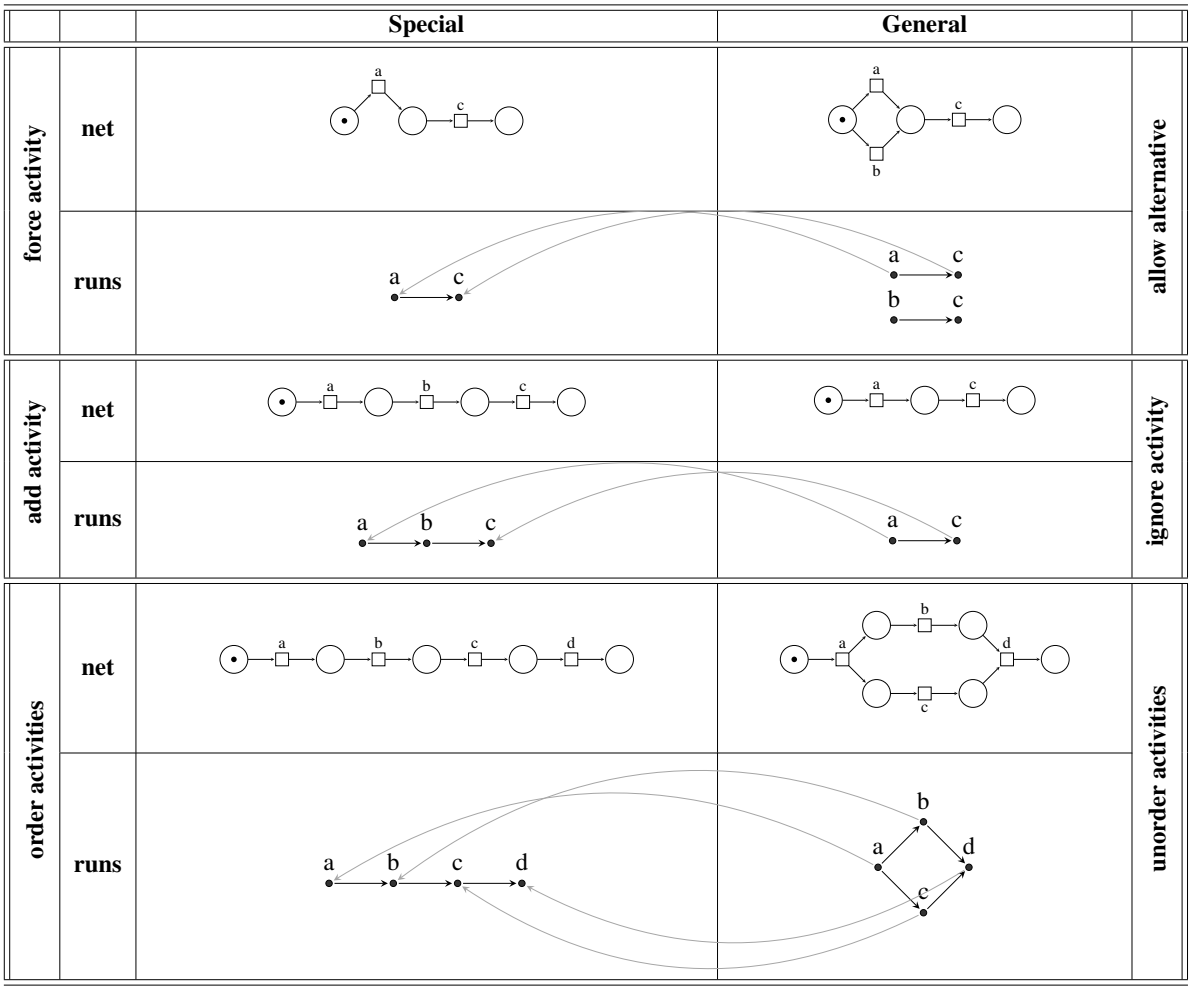


Table 1: Representative examples of specialisation/generalisation

4 Branching Time Specialisation

We begin this section with an example motivating further enhancement of the specialisation notion.

Example 4 Consider two processes P and P' modelled by the Petri nets in Table 2. These nets are actually labelled Petri nets. The two transitions of the net on the left-hand side labelled by a represent the same task a. As shown in the pictures, they have identical sets of runs. Both processes start with an occurrence of a and then either continue with an occurrence of b or with an occurrence of c. Hence they cannot be distinguished by just inspecting their runs. However, in process P, after the occurrence of a, there is a choice between continuing with b or with c, whereas in process P' we have to choose immediately between the run containing occurrences of a and b, and the one containing occurrences of a and c. In this case we might consider process P' as a specialisation of P since additional (more precisely, earlier) information about the choice between b and c is necessary.

We cannot distinguish processes P and P' by means of their runs as in the previous section. Therefore, it is necessary to carefully examine all possible behaviours. To do so, we first define the prefix of a run.

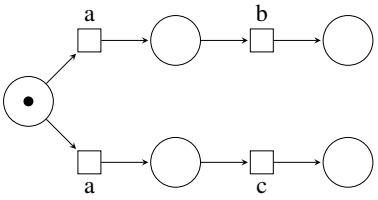
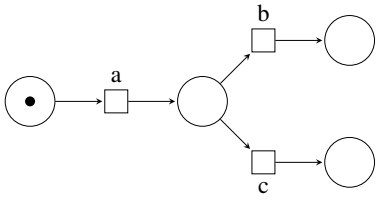
	P'	P
net		
runs		

Table 2: Runs do not always distinguish processes

Definition 6 (Prefix of a run) A run π is a *prefix of a run* π' if there is an injective mapping $\mu\colon A_\pi \to A_{\pi'}$ such that

1. $\forall x, y \in A_\pi, x \to_\pi y \iff \mu(x) \to_{\pi'} \mu(y)$,
2. $\forall x \in A_\pi, \lambda_\pi(x) = \lambda_{\pi'}(\mu(x))$, and
3. $\forall x', y' \in A_{\pi'} : x' \to_{\pi'} y' \Rightarrow [(\exists y \in A_\pi, y' = \mu(y)) \Rightarrow (\exists x \in A_\pi, x' = \mu(x))]$.

Explanation:
Each precedence relation in the prefix has a correspondence in the complete run π', (1) and corresponding activities are mapped to the same task (2). Moreover, each precedence relation of the complete run π' leading to an activity that corresponds to one of the prefix also has its starting point corresponding to an activity of the prefix (3).

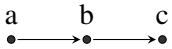
Example 5 The process run of Figure 4(a) is a prefix of the process runs of Figures 4(b) and 4(c).

Roughly speaking, a prefix of a run is constituted by a set of activities together with all their predecessors, up to an isomophism. This observation is formalised in the following lemma:

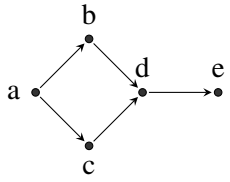
Lemma 2 *A run π is a prefix of a run π' if and only if there exists a process run $\overline{\pi}$ such that*

1. $\overline{\pi}$ is isomorphic to π,

(a) Process run π



(b) Continuation π'



(c) Continuation π''

Figure 4: A process run π with different extended runs π' and π''

2. $A_{\overline{\pi}} \subseteq A_{\pi'}$,
3. $\to_{\overline{\pi}} = \to_{\pi'} \cap A_{\overline{\pi}} \times A_{\overline{\pi}}$,
4. $\forall x \in A_{\overline{\pi}}, \lambda_{\overline{\pi}}(x) = \lambda_{\pi'}(x)$,
5. $\forall y \in A_{\overline{\pi}}, x \to_{\pi'} y \Rightarrow x \in A_{\overline{\pi}}$.

Proof ($\Rightarrow$) Let μ be the mapping mentioned in the definition of a prefix. Set $A_{\overline{\pi}} = \{x' \in A_{\pi'} \mid \exists x \in A_\pi : x' = \mu(x)\}$. The other defining components of $\overline{\pi}$ are items 3 and 4 of the lemma. ($\Leftarrow$) Choose $\mu(x) = \mu_i(x)$ for every $x \in A_\mu$ where μ_i is the isomorphism from π to $\overline{\pi}$.

Explanation:
This lemma is very similar to Lemma 1. Run $\overline{\pi}$ is exactly the restriction of run π' to the activities in $\overline{\pi}$.

Now, for a run that is a prefix of another run, we define the set of its continuations.

Definition 7 (Extended run, continuations)
An *extended run* is a run π together with a set of runs $C(P)$, called its *continuations*, such that π is a prefix of each run in $C(P)$.

Note that in general $C(P)$ does not contain all runs with prefix π. Two extended runs are different if their continuations are different, even if their respective runs are isomorphic.

Example 6 Figure 4 pictures a process run π (Figure 4(a)), and two different continuations: π' in Figure 4(b) and π'' in Figure 4(c). Then $(\pi, \{\pi'\})$ and $(\pi, \{\pi''\})$ are two different extended runs of π.

Isomorphic runs can lead to different states. Definition 7 avoids considering states of processes. For applying our concept to concrete process definition languages, we might consider the states reached by runs instead, determining the possible continuations.

Remark 4 For unlabelled Petri nets the distinction does not occur because isomorphic runs lead to identical markings. For labelled Petri nets the problem can occur.

Definition 8 (Branching Time Specialisation)
A process model P' is a *branching time specialisation* of a process model P if, for each extended run π' of P' with continuation $C(P')$, there exists an extended run π of P with continuation $C(P)$ such that

- $\pi' \geq \pi$, and
- for each run $\tilde{\pi}' \in C(P')$ there exists a run $\tilde{\pi} \in C(P)$ such that $\tilde{\pi}' \geq \tilde{\pi}$.

Example 7 The nets in Table 2 can be distinguished using the specialisation of Definition 8, while they could not be with the linear time specialisation of Definition 5.

In Table 3, the activities are numbered after transitions names. The process model P' has a run π', consisting only of activity 1 labelled by a. Its set of continuations $C(P')$ contains this run itself and also the run $1 \rightarrow 2$, as given in the figure. For P, we also have the run $\pi = 1$, labelled by a. Its set of continuations $C(P)$ contains also the run $1 \rightarrow 4$, as given in the figure. So, in this example, for every continuation in $C(P')$ we find an isomorphic continuation in $C(P)$. Conversely, consider the extended run π consisting only of activity 1 labelled by a in process model P, which is a prefix of both depicted runs. Its continuation is the set of both runs depicted in the table. So it is possible to continue with a b-activity and it is also possible to continue with a c-activity. Now activity 1 (i.e., the run consisting only of this activity) can not be an associated run of P' because it misses a continuation with an activity labelled with c. Similarly, activity 3 can not be an associated run of P' because this one misses a continuation with an activity labelled with b. Arguing about all possible runs and continuations of P' as above shows that P' is a specialisation of P.

We call this concept of specialisation Branching Time Specialisation because, again, we have that the valid statements (now expressable in Branching Time Temporal Logic) of the more general process should also hold for the specialised one.

5 Related Works

Abstraction and refinement have been the subject of numerous researches, and we just mention some of them here. The goals include to keep the modelling and reasoning as close as possible to the essence of the system to be developed, while still taking into account that a concrete representation (also called concrete implementation) of data should be provided at some later point together with the way it is related with the abstract data.

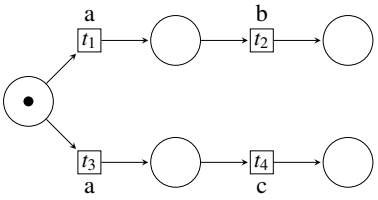
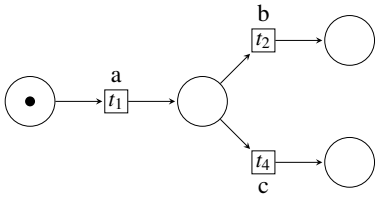
	P'	P
net		
runs	a b 1 → 2 a c 3 → 4	a b 1 → 2 a c 1 → 4

Table 3: Runs display different activities

(Hoare 1972) proposed a method to prove program correctness in the context of stepwise refinement where a representation of abstract data is chosen. Later, (Guttag et al. 1978) showed how the use of algebraic axiomatizations can simplify the process of proving the correctness of an implementation of an abstract data type. A number of works followed in the algebraic specification field. Now, an explicit data type refinement construct is introduced in programming languages like Scala (The Scala Programming Language n.d.). Refinement can also be used for program derivation as in (Diallo et al. 2015), who define a correctness based concept of refinement.

As mentioned in the introduction, abstraction was studied for database modelling in (Smith and Smith 1977), with the concepts of generalisation and aggregation. These concepts are also part of the object-oriented approaches (e.g. UML based approaches), or languages.

The same concepts should also be adapted to the modelling of the behaviour of a system, described by a process, a state-based diagram, etc.

(Lee and Wyner 2003) define a specialisation concept for dataflow diagrams. They distinguish minimal execution set semantics, in which adding an activity is a specialisation (as in Table 1), and the maximal execution set semantics, which is the option they choose. So obviously, their definition of specialisation differs from ours.

In (Wyner and Lee 2005), they work on a specialisation for a variant of Petri net (Workflow Process Definition). They analyse the approach of (Aalst and Basten 2002) who identify four types of inheritance of workflows, and propose an extension.

Several Petri nets refinements (node, type, subnet) were defined by (Lakos 2000), and later extended in (Choppy et al. 2013) to propose type refinement that complies with the subtype relation defined by (Liskov and Wing 1994). These refinements are useful both, for an incremental development of the specification, and for substantial gain in model checking properties.

(Wang et al. 2010) stress that design issues are important, and, in the context of component-based model-driven development, they present two refinement relations, a trace-based refinement and a state-based (data) refinement, that provide different granularity of abstractions. So they combine the data refinement and the behaviour refinement.

Along similar lines (Fayolle et al. 2015) propose to couple a dynamic behaviour specification expressed with ASTD (Algebraic State Transition Diagrams) with a data model described by means of an Event-B specification. The complementarity and consistency of refinements for both parts

are explored, and the approach is illustrated on a CBTC-like train controller.

6 Conclusions

Summary

We have presented a very general notion of specialisation and generalisation of processes which does not stick to a specific process modelling language but can be applied to all process languages where behaviour can be expressed in terms of partially ordered activities. Processes with sequential runs are a special case where all activities in runs are mutually ordered. Specialisation is interpreted as addition of features, where features can be additional tasks, additional ordering information, additional information on choices and earlier information on choices. All these features except the last one can be expressed by a specialisation relation on runs whereas the last one concerns branching points of process models that do not appear in runs and hence require a more involved definition based on runs and their possible continuations.

Specialisation versus Refinement/Generalisation versus Aggregation

One could argue that refinement of a process element is a form of specialisation because the more detailed view adds information. Conversely, what distinguishes generalisation and aggregation? According to our definition, specialisation *adds* something to a process, whereas refinement *replaces* something by something else, which should be more detailed. For example, if a task is added to the process, then the process is more special and associated activities show up in its runs. If a task is refined to two subsequent tasks, then the process is more detailed and the respective activities are refined accordingly in its runs.

Specialisation versus Instantiation

In one of our previous examples we have shown that information about the decision of choices can be viewed as a particular specialisation. The specialised process contains less alternatives. If all choices are decided, then the process is deterministic and contains no alternative at all. In other words, it only has a single run (remember that our notion of a run captures possible concurrency so that no additional runs caused by interleaving appear). Depending of the representation of the process and of its single run, both might look very similar. However, the run represents an instance of the process (and was an instance of the original process, too) whereas the process is on a lower meta-level. Whereas repeated specialisation of processes is possible and always yields new processes, instantiation of processes decreases the meta-level by one and can only occur once.

Extensions

It is obvious that there are features, that can be added, which cannot be handled by our concept so far. One example concerns concurrency. Our notion of partially ordered runs is interpreted in such a way, that activities that are not ordered can occur concurrently or in any order. However, it could be possible to specialise such a specification by demanding that activities have to occur concurrently whereas any order would be illegal. This cannot be expressed by our notion unless we add an additional "concurrent" relation. Other, similar relations are "not later than" or "at most two of three activities occur concurrently".

Another extension refers to data. Usually tasks are performed for some or several data objects. This data does not appear in runs of our processes. For a combined view of processes and data, and for a combined notion of specialisation, the process view and the data view have to be integrated. There is an obvious way to do this integration by carrying over the data attributes from tasks to activities. Then the mapping between activities constituting our specialisation relation has to be extended to these data objects; the more general process handles the more general data.

Representation

In data modelling, generalisation and other abstraction techniques such as aggregation are represented graphically in the model. While this is

nicely possible for aggregation in process models (see, for example, Desel and Merceron 2010), we do not see an elegant solution for generalisation in process models. One obvious approach is to depict additional tasks and additional relations between tasks by means of different symbols (colours or lines, respectively). However, if a single specialisation considers changes in different parts of a process then it is not obvious to express that these changes can only occur together.

The Partial Order of Specialisation

Our notion of generalisation and specialisation is not primarily given as a problem (is x a specialisation of y ?) but as a specification means. However, it is an interesting question whether the above question is decidable and, in the positive case, what is the complexity of a decision algorithm or of the problem in general (i. e., the complexity of the most efficient algorithm).

It is not difficult to see that the specialisation relation between single process runs is decidable, although any algorithm heavily depends on the data structure used to represent the respective process runs. Notice that by definition process runs are finite, i. e., have a finite set of activities. We cannot deal with isomorphism classes of process runs in algorithms but instead consider arbitrary representative process runs.

Proposition 1 *Given two process runs π and π' of two process models P and P', it is decidable whether π' specialises π.*

Proof π' can only be a specialisation of π if, for each task t, π' has at least as many activities labelled by t as π has. This condition is easy to check. Assume that it holds true.

There are finitely many injective label-preserving mappings from the activities of π to the activities of π', and it is not difficult to construct them in a systematic way. For each pair of activities of π, which are in the immediate precedence relation, we check whether the respective target activities are ordered in π' (they do not necessarily have to be immediate successors!). If this is true for at least one mapping, then π' is a specialisation of π.

Things become more difficult when process models are compared because each process model can have infinitely many runs. Since we do not consider any particular process model, nothing can be said about decidability in general. Clearly, there is only a chance for decidability of specialisation if a process model cannot generate infinitely many arbitrary runs.

It is not difficult to prove that "being a specialisation" is a partial order on process models. Its minimal element is the empty process model which is a (useless, though) generalisation of all process models. Using this order one might ask, for a given set of process models, whether there is a "largest" generalisation, whether this is unique, and whether it can be constructed algorithmically. Similarly, it would be interesting to find the "smallest" specialisation of a set of process models. There are no obvious answers to these questions, and hence the study of the partial order of specialisation is subject to future work.

References

van der Aalst W. M. P., Basten T. (2002) Inheritance of workflows: an approach to tackling problems related to change. In: Theoretical Computer Science 270(1-2), pp. 125–203

Bhat G., Peled D. (1998) Adding Partial Orders to Linear Temporal Logic. In: Fundamenta Informaticae 36(1), pp. 1–21

Choppy C., Petrucci L., Sanogo A. (2013) Coloured Petri Nets Refinements. In: Proceedings of the workshop on Petri Nets and Software Engineering (PNSE'13), Workshop Proceedings 989, CEUR, pp. 187–201

Derrick J., Boiten E. A., Reeves S. (eds.) Proceedings 17th International Workshop on Refinement, RefineFM 2015. EPTCS Vol. 209 https://doi.org/10.4204/EPTCS.209

Desel J., Merceron A. (2010) Vicinity Respecting Homomorphisms for Abstracting System Requirements. In: Transactions on Petri Nets and Other Models of Concurrency Lecture Notes in Computer Science 4 Jensen K., Donatelli S., Koutny M. (eds.), pp. 1–20

Diallo N., Ghardallou W., Desharnais J., Mili A. (2015) Program Derivation by Correctness Enhacements. In: Derrick J., Boiten E. A., Reeves S. (eds.) Proceedings 17th International Workshop on Refinement, RefineFM 2015, Oslo, Norway, 22nd June 2015.. EPTCS Vol. 209, pp. 57–70 https://doi.org/10.4204/EPTCS.209.5

Fayolle T., Frappier M., Laleau R., Gervais F. (2015) Formal refinement of extended state machines. In: Derrick J., Boiten E. A., Reeves S. (eds.) Proceedings 17th International Workshop on Refinement, RefineFM 2015, Oslo, Norway, 22nd June 2015.. EPTCS Vol. 209, pp. 1–16 https://doi.org/10.4204/EPTCS.209.1

Frank U., Laak B. V. (2002) A Method for the Multi-Perspective Design of Versatile E-Business Systems. In: Proceedings of the 8th Americas Conference on Information Systems, pp. 621–627

Grabowski J. (1981) On partial languages. In: Fundamenta Informaticae 4(2), pp. 428–498

Guttag J. V., Horowitz E., Musser D. R. (1978) Abstract data types and software validation. In: Commun. ACM 21(12), pp. 1048–1064

Hoare C. A. R. (1972) Proof of Correctness of Data Representations. In: Acta Informatica 1, pp. 271–281

Lakos C. (2000) Composing Abstractions of Coloured Petri Nets.. In: Nielsen, M., Simpson, D. (eds.) 21st Int. Conf. on Application and Theory of Petri Nets (ICATPN). Lecture Notes in Computer Science Vol. 1825. Springer-Verlag, pp. 323–345

Lee J., Wyner G. M. (2003) Defining specialization for dataflow diagrams. In: Information Systems 28(6), pp. 651–671

Liskov B. H., Wing J. M. (1994) A Behavioral Notion of Subtyping. In: ACM Trans. on Programming Languages and Systems 16(6), pp. 1811–1841

Mayr H. C., Kop C., Esberger D. (2007) Business Process Modeling and Requirements Modeling. In: First International Conference on the Digital Society (ICDS'07), pp. 8–8

Mayr H. C., Michael J. (2012) Control pattern based analysis of HCM-L, a language for cognitive modeling. In: International Conference on Advances in ICT for Emerging Regions (ICTer2012). IEEE, pp. 169–175

Pratt V. (1986) Modelling Concurrency with Partial Orders. In: Int. Journal of Parallel Programming 15, pp. 33–71

Smith J. M., Smith D. C. P. (1977) Database abstractions: aggregation and generalization. In: ACM Trans. Database Systems 2(2), pp. 105–133

The Scala Programming Language (n.d.) http://www.scala-lang.org/. Last Access: Accessed on: 29.1.2018

Wang Z., Wang H., Zhan N. (2010) Refinement of Models of Software Components. In: Proceedings of SAC'10, pp. 2311–2318

Workflow Management Coalition (n.d.) http://www.wfmc.org/. Last Access: Accessed on: 29.1.2018

Wyner G. M., Lee J. (2005) Applying Specialization to Petri Nets: Implications for Workflow Design. In: Bussler C., Haller A. (eds.) Business Process Management Workshops Vol. 3812, pp. 432–443

This work is licensed under a Creative Commons 'Attribution-ShareAlike 4.0 International' licence.