Technische Universiteit
Eindhoven
University of Technology

Parameterized Timed Partial Orders with Resources: Formal Definition and Semantics

Nikola Trčka, Marc Voorhoeve and Twan Basten

ES Reports
ISSN 1574-9517

ESR-2010-01
2 June 2010

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Parameterized Timed Partial Orders with Resources: Formal Definition and Semantics

Nikola Trčka[1] Marc Voorhoeve[1] Twan Basten[1,2]

[1]Eindhoven University of Technology [2]Embedded Systems Institute

1 Introduction

In this note we define (task-level) *timed partial orders* (TPOs), as a lifting of standard timed event partial orders to the level of *tasks* (where every task consists of an enabling and a completing event). We also introduce a generic resource platform on which these tasks are to be executed, and propose a resource handling scheme. The whole mechanism is given an operational semantics. To simplify large, but structured, TPO specifications we next introduce Parameterized TPOs (PTPOs), a high-level representation of TPOs, obtained by adding index parameters to events and constraining precedence rules by boolean expressions.

2 Preliminaries

If A is a set, then $\mathcal{F}(A)$ is the set of all finite subsets of A. The set of real numbers is denoted $\mathbb{R}$, having as subset the closed unit interval $\mathcal{I} = [0, 1]$. The set of non-negative reals is denoted $\mathbb{R}_{\geq 0}$, and the set of natural numbers is denoted $\mathbb{N}$.

If A, B are sets, then A^B is the set of (total) functions from B to A. By a^B, we denote the only function from the singleton set $\{a\}^B$. We use the symbol $\hookrightarrow$ to denote partial mappings. We assume the standard comparison, and the standard operations (when applicable), for the elements of B^A, for any A and $B \subseteq \mathbb{R}$.

3 Timed Partial Orders

Engineers typically specify *precedence* (and resource) constraints for different *tasks* in their systems. The standard model of event partial orders can be used to specify such constraints, but this often becomes cumbersome as tasks need to be unfolded to individual events (typically start and end events). For this reason, we lift the notion of event partial orders to the level of tasks, adding no real expressivity but only simplifying specifications and, as shown later, making the connection with resources more natural.

The notion of precedence between tasks is more fine grained than in the case of events: Tasks typically have duration and the notion of "before" can have multiple meanings. Based on an observation from practice, we distinguish four types of task precedence rules: EE– specifying that the first task must be enabled before the second one is enabled, EC– specifying that the

first task must be enabled before the second one is completed, CE– specifying that the first task must be completed before the second one is enabled, and CC– specifying that the first task must be completed before the second one is completed. We define $\mathsf{RULE} = \{\mathsf{EE}, \mathsf{EC}, \mathsf{CE}, \mathsf{CC}\}$. Note that the last three rules do not refer to the actual completion of a task's execution (this event is resource dependent), but to the completion event being received at the task level.

We presuppose a time domain $\mathsf{TIME} \subseteq \mathbb{R}_{\geq 0} \cup \{\infty\}$ that contains 0 and ∞, and is closed under standard operations (subtraction only when applicable). A *(task-level) timed partial order* (TPO) is a pair (T, P), where T is a set of *tasks* and $P : T \times \mathsf{RULE} \times T \hookrightarrow \mathsf{TIME}$ is a partial mapping describing the *precedence relation*. Intuitively, if e.g. $P(t, \mathsf{CE}, u) = d$, then u gets enabled only after both t has completed and d time units have passed. If $d = 0$, then no real timing constraint is imposed. If $P(t, \mathsf{EC}, u)$ is undefined, then there is no precedence rule of type EC between t and u.

TPOs are visualized as (labeled) directed graphs, with nodes representing tasks and arcs indicating the precedence relation. The graph in Figure 1, e.g., represents a TPO (T, P) with $T = \{A, B, C\}$ and $P = \{(A, \mathsf{EC}, B, 0), (A, \mathsf{EE}, C, 5)\}$. Note that time constraints are not shown if zero.

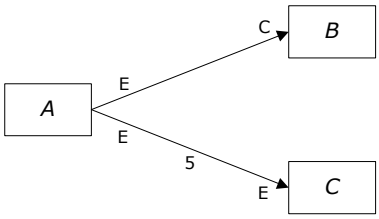

Figure 1: A TPO (T, P) with $T = \{A, B, C\}$ and $P = \{(A, \mathsf{EC}, B, 0), (A, \mathsf{EE}, C, 5)\}$

We now define when a TPO is called consistent, and we give a formal (operational) semantics to TPOs. We introduce some notation first.

Let (T, P) be a TPO. Let $E = \{e_t, c_t \mid t \in T\}$ be the set of *events* corresponding to the tasks in T: the event e_t denotes the *enabling* of t and the event c_t denotes the *completion* of t. Let $P_E : E \times E \hookrightarrow \mathsf{TIME}$ be defined as follows: i) $P_E(e_t, c_t) = 0$ for all $t \in T$, and ii) $P_E(e_t, e_u) = P(t, \mathsf{EE}, u)$, $P_E(e_t, c_u) = P(t, \mathsf{EC}, u)$, $P_E(c_t, e_u) = P(t, \mathsf{CE}, u)$, and $P_E(c_t, c_u) = P(t, \mathsf{CC}, u)$ for all $t, u \in T$. Note that P_E simply "unfolds" the precedence rules to their event-based representations, and that the pair (E, P_E) is a standard timed event partial order.

A TPO (T, P) is called *consistent* iff the transitive closure of $\mathrm{dom}(P_E)$, denoted $\mathrm{dom}(P_E)^+$, is irreflexive, i.e., if there is no $e \in E$ such that $\mathrm{dom}(P_E)^+(e, e)$. Consistency is a sanity property of every TPO, as it implies that no event occurrence is constrained by itself (i.e., (E, P_E) is an irreflexive event partial order).

We now give a formal (operational) semantics to TPOs. For this, we define a partial function $\mathsf{C} : E \hookrightarrow \mathsf{TIME}$, called a *configuration function*, that specifies which events have occurred and at which time instance. We write $TPO\langle \mathsf{C}, \mathsf{time} \rangle$ to denote that the TPO is in configuration C at time instance time. Transition labels are elements of the set $E \cup \mathsf{TIME}$, capturing event occurrences and time progress respectively. No action urgency is imposed at this moment, i.e., time is always allowed to progress. We define the following two rules:

$$\frac{e \notin \mathrm{dom}(\mathsf{C}),\ \forall f \in E : (P_E(f,e) = d) \Rightarrow (f \in \mathrm{dom}(\mathsf{C}) \text{ and } \mathsf{C}(f) + d \leq \mathsf{time})}{TPO\langle \mathsf{C}, \mathsf{time}\rangle \xrightarrow{e} TPO\langle \mathsf{C} \cup \{(e, \mathsf{time})\}, \mathsf{time}\rangle}\ 1$$

$$\frac{d \in \mathsf{TIME}}{TPO\langle \mathsf{C}, \mathsf{time}\rangle \xrightarrow{d} TPO\langle \mathsf{C}, \mathsf{time} + d\rangle}\ 2$$

Tasks can sometimes have a prespecified minimal time duration, independent of the resource that will execute them. If a task is finished before this time, the remaining time has to expire before any of the depending tasks is enabled. These durations can be specified easily in the TPO model. For example, if the minimal duration of $t \in T$ should be set to d, then we would simply put the rule $P(t, \mathsf{EC}, t) = d$.

4 Resource Handling

We first informally explain the whole resource handling mechanism and how it connects to the TPO level. Then we formally introduce all the parameters that the user is supposed to specify. Finally, we formally specify the dynamic behavior of the resource handler. A global set of tasks T is assumed.

4.1 Introduction to the overall resource handling mechanism

Tasks claim and/or release resources, where every resource has an associated cost and the relative total capacity of 1. We use relative resource amounts only to simplify the presentation; the unit-interval abstraction still allows us to express real resource amounts with any desired accuracy. In order for a task to start executing, specific resource amounts are needed. Depending on the amounts obtained, the task can proceed at a certain pace (rate of execution). A task's size can be expressed as a minimum duration (obtained when all resources are available); a task of size s proceeding at constant pace p will take s/p time units to execute. Upon termination, the task can transfer some of the resources it holds to other (successor) tasks; the remaining (amounts of) resources are freed. This feature is typically used for buffer-type resource sharing.

During the execution of a task, some amount of the resources held by this task can be reallocated, i.e. given to other tasks, possibly changing its pace. A task may even be halted, where its pace becomes 0, in order, e.g., to free resources needed for a higher priority task. To what extent resources held by a task can be reallocated depends on a user-specified function.

The resource handler consists of a receiver and a scheduler. The receiver receives task requests coming from the TPO level (via the receiving interface) and places them in the set of ready tasks. The scheduler assigns resource amounts to ready tasks based on some predefined and user-specified policy, respecting the reallocation rules. Tasks that are completed are removed from the ready set and sent back to the TPO (via the sending interface). The scheduler can be periodic, event driven, or both. In the first case, scheduling decisions are made every $\epsilon > 0$ time units. In the second case, the scheduler is invoked only when a new task arrives or when some running task is finished. In the third case, scheduling can be seen as periodic in principle, but event-driven inside the period interval. Note that this case would typically boils down to event driven scheduling.

4.2 Adding resource information to TPOs

Given the set of resources R and the set of tasks T, the user specifies the following:

- A function $c : R \times \mathcal{I} \times \mathsf{TIME} \to \mathbb{R}^n$, $n \in \mathbb{N}$, indicating the (n-dimensional) *cost* of using a certain amount of a resource for some time period.
- A *deadline* function $\partial : T \to \mathsf{TIME}$, assigning a maximum allowed delay from the time a task is enabled till it is completed. If $\partial(t) = \infty$, then t has no real deadline.
- A *duration* function $\delta : T \times \mathcal{I}^R \to \mathsf{TIME}$, indicating how much time it takes to perform some task having certain resource occupation. This function should be non-increasing (with respect to the second argument).

  Given a $t \in T$ and an $o_t \in \mathcal{I}^R$, we define $\pi(t, o_t) = \delta(t, 1^R)/\delta(t, o_t) \in \mathcal{I}$ and call it the *pace* (rate of execution) of t under resource occupation o_t.
- A *resource modification* function $\mu : T \times \mathcal{I}^R \to \mathcal{F}(\mathcal{I}^R)$, which given a task and its current resource allocation, specifies the resource modifications that are allowed in the next scheduling event. If the current allocation of $t \in T$ equals o_t, then the new occupation o'_t should satisfy $o'_t \in \mu(t, o_t)$.
- A *resource handover* function $\eta : T \times T \times \mathcal{I}^R \to \mathcal{I}^R$, indicating the handover of resources upon termination. The function η must satisfy $\Sigma_{u \in T} \eta(t, u, \bar{x}) \leq \bar{x}$, meaning that only held resources can be transferred.
- A *scheduling policy* function $\varsigma : \mathcal{F}(T \times \mathsf{TIME} \times \mathsf{TIME} \times \mathcal{I}^R) \times \mathcal{I}^R \to \mathcal{F}(T \times \mathcal{I}^R)$, which given a set of running tasks, with their arrival times, remaining durations and current resource occupations, and the total free (non-reserved) resource amounts, specifies the possibilities for new resource occupations for these tasks. The function can make use of all the previously defined elements. Note that by allowing multiple possibilities for new resource occupations, we support non-deterministic scheduling policies.

  The ς function must satisfy the following. For all $X \in \mathcal{F}(T \times \mathsf{TIME} \times \mathsf{TIME} \times \mathcal{I}^R)$, all $o \in \mathcal{I}^R$, and all $Y \in \varsigma(X, o)$, it holds that

  1. $(t, a_t, d_t, o_t) \in X$ iff $(t, o'_t) \in Y$ for some $o'_t \in \mu(t, o_t)$ – no tasks are removed/introduced and the modification constraints are met; and
  2. $\sum_{(t, o'_t) \in Y} o'_t \leq o$ – only the non-reserved resources can be granted.

4.3 Resource handler dynamics

The resource handler keeps track of several objects dynamically. These are:

- A partial function $\Theta : T \hookrightarrow \mathsf{TIME} \times \mathsf{TIME} \times \mathcal{I}^R$, where, for $\Theta(t) = (a_t, d_t, o_t)$, a_t holds the time at which t has arrived at the resource handler, d_t represents the current *remaining duration* of t (relative to the minimal duration with pace 1) and o_t represents the current *resource occupation* of t.
- A partial function $H : T \times T \hookrightarrow \mathcal{I}^R$ representing the resource amounts that are *handed over* from one task to another.

- A number $\Gamma \in \mathbb{R}^n$ representing the *total accumulated cost*.
- A flag $\natural \in \{0,1\}$ indicating that scheduling *is* (represented by 1), or *is not* (represented by 0), allowed to occur. Given $\natural_1, \natural_2 \in \{0,1\}$ we introduce a shorthand notation: $(\natural_1 : \natural_2) = \begin{cases} \natural_1, & \text{if event-driven scheduling} \\ \natural_2, & \text{if periodic scheduling} \\ \max(\natural_1, \natural_2), & \text{if both event-driven and periodic scheduling} \end{cases}$

When a new task is received, its initial remaining duration is its minimal duration, and its initial resource occupation is the whole amount of resources that are possibly kept for this task by the previous tasks (H). When a task is finished (its remaining time is 0), it is removed from Θ, and the function H is updated to include the resources that this task possibly hands over to some successor tasks.

The periodic scheduler is invoked every $\epsilon \in \mathsf{TIME}$ time units. When it wakes up, it computes the new remaining duration of every task by putting $d'_t = \max(0, d_t - \epsilon \cdot \pi(t, o_t))$ where $\pi(t, o_t)$ is the current pace of t. Similarly, the total cost is updated by $\Gamma' = \Gamma + \sum_{r \in R} c(r, \sum_{t \in \mathrm{dom}(\Theta)} o_t(r), \epsilon)$. The new resource configuration is determined from one of the possibilities specified by ς. Each of the running tasks obtains a new pace with which it is running till the next scheduling period.

The event-driven scheduler is essentially the same as the periodic one but it is only invoked after a new task has arrived or when some running task has finished.

We now give formal semantics of the resource handler. The state of the resource handler is determined by the functions Θ and H, the numbers Γ and $\natural$, and the current time instance. The set of transition labels is $\{?t, !t \mid t \in T\} \cup \{\mathsf{sch}\} \cup \mathsf{TIME}$, corresponding respectively to a receipt of a new task, sending of a finished task, a scheduling activity and time progress.

$$\frac{(t, a_t, 0, o_t) \in \Theta}{RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{!t} RH\langle \Theta \setminus \{(t, a_t, 0, o_t)\}, H \cup \{((t,u), \eta(t,u,o_t)) \mid u \in T\}, \Gamma, (1 : \natural), \mathsf{time}\rangle} \; 3$$

$$\frac{t \in T}{\begin{array}{l} RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{?t} \\ RH\langle \Theta \cup \{(t, \mathsf{time}, \delta_t, \Sigma_{(u,t) \in \mathrm{dom}(H)} H(u,t))\}, H \setminus \cup_{u \in T}(u, t, H(u,t)), \Gamma, (1 : \natural), \mathsf{time}\rangle \end{array}} \; 4$$

$$\frac{d \in \mathsf{TIME}}{\begin{array}{l} RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{d} \\ RH\langle \{(t, a_t, \max(0, d_t - d \cdot \pi(t, o_t)), o_t) \mid (t, a_t, d_t, o_t) \in \Theta\}, H \\ , \Gamma + \sum_{r \in R} c(r, \sum_{t \in \mathrm{dom}(\Theta)} o_t(r), d), (\natural : \begin{cases} 1, & (\mathsf{time} + d)/\epsilon \in \mathbb{N} \\ 0, & \text{else} \end{cases}), \mathsf{time} + d\rangle \end{array}} \; 5$$

$$\frac{\natural = 1, \; Y \in \varsigma(\Theta, \mathcal{I}^R - \sum_{(u,v) \in \mathrm{dom}(H)} H(u,v))}{RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\mathsf{sch}} RH\langle \{(t, a_t, d_t, o'_t) \mid (t, o'_t) \in Y\}, H, \Gamma, 0, \mathsf{time}\rangle} \; 6$$

4.4 Putting the whole system together

The following rules define the (asynchronous) communication interface between the TPO level and the resource handler, and the dynamics of the complete system. We use two interface variables $\mathsf{PR}, \mathsf{RP} \subseteq T$, to respectively hold the tasks traveling from the TPO to the resource handler and the tasks traveling from the resource handler to the TPO. The set of transition labels is $\{\mathsf{po!}(t), \mathsf{po?}(t), \mathsf{rh!}(t), \mathsf{rh?}(t) \mid t \in T\} \cup \{\mathsf{sch}\} \cup \mathsf{TIME}$. The labels $\mathsf{po?}(t)$ and $\mathsf{po!}(t)$ correspond to task t being received from, respectively sent to, the TPO level. Similarly, the labels $\mathsf{rh!}(t)$ and $\mathsf{rh?}(t)$ correspond to task t being sent to, respectively received from, the resource handler. Label sch denotes the scheduling activity and the rest of the labels are for time progress.

$$\frac{TPO\langle \mathsf{C}, \mathsf{time}\rangle \xrightarrow{s_t} TPO\langle \mathsf{C}', \mathsf{time}\rangle}{SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\mathsf{po!}(t)} SYS\langle \mathsf{C}', \mathsf{PR} \cup \{t\}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle} \; 7$$

$$\frac{TPO\langle \mathsf{C}, \mathsf{time}\rangle \xrightarrow{e_t} TPO\langle \mathsf{C}', \mathsf{time}\rangle, \;\; t \in \mathsf{RP}}{SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\mathsf{po?}(t)} SYS\langle \mathsf{C}', \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle} \; 8$$

$$\frac{RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{!t} RH\langle \Theta', H', \Gamma', \natural', \mathsf{time}\rangle}{SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\mathsf{rh!}(t)} SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP} \cup \{t\}, \Theta', H', \Gamma', \natural', \mathsf{time}\rangle} \; 9$$

$$\frac{RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{?t} RH\langle \Theta', H', \Gamma', \natural', \mathsf{time}\rangle, \;\; t \in \mathsf{PR}}{SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\mathsf{rh?}(t)} SYS\langle \mathsf{C}, \mathsf{PR} \setminus \{t\}, \mathsf{RP}, \Theta', H', \Gamma', \natural', \mathsf{time}\rangle} \; 10$$

$$\frac{TPO\langle \mathsf{C}, \mathsf{time}\rangle \xrightarrow{d} TPO\langle \mathsf{C}', \mathsf{time}'\rangle, RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{d} RH\langle \Theta', H', \Gamma', \natural', \mathsf{time}'\rangle}{SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{d} SYS\langle \mathsf{C}', \mathsf{PR}, \mathsf{RP}, \Theta', H', \Gamma', \natural', \mathsf{time}'\rangle} \; 11$$

$$\frac{RH\langle \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\mathsf{sch}} RH\langle \Theta', H', \Gamma', \natural', \mathsf{time}\rangle}{SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\mathsf{sch}} SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta', H', \Gamma', \natural', \mathsf{time}\rangle} \; 12$$

To reduce the level of non-determinism in the model, we assume that scheduling does not take place until every task that is either already enabled by the TPO or will be enabled when a traveling task-completion information arrives, has sent its request and that request has arrived to the queue. Moreover, we assume that all events are *eager*, i.e. that time is allowed to pass only if no event can occur. Time progress is maximal: if the system can let an amount d of time pass, then it cannot let a smaller amount pass. To achieve all this we introduce a partial order $\prec$ on transition labels, and set $d_1 \prec d_2 \prec \mathsf{sch} \prec x$ for every $x \in \{\mathsf{po!}(t), \mathsf{po?}(t), \mathsf{rh?}(t), \mathsf{rh!}(t) \mid t \in T\}$ and every $d_1, d_2 \in \mathsf{TIME}$ such that $d_1 < d_2$. The first then rule imposes the priority restrictions; the second rule is needed to support the maximal progress assumption.

$$\frac{\begin{array}{l} SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\ell} SYS\langle \mathsf{C}', \mathsf{PR}', \mathsf{RP}', \Theta', H', \Gamma', \natural', \mathsf{time}'\rangle, \\ \forall \ell' \succ \ell : SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\ell'} \end{array}}{SYS_{\prec}\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{\ell} SYS_{\prec}\langle \mathsf{C}', \mathsf{PR}', \mathsf{RP}', \Theta', H', \Gamma', \natural', \mathsf{time}'\rangle} \; 13$$

$$\frac{\begin{array}{l} SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{d} SYS\langle \mathsf{C}', \mathsf{PR}', \mathsf{RP}', \Theta', H', \Gamma', \natural', \mathsf{time}'\rangle, \\ \forall d' : 0 < d' < d : \forall \ell : SYS\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time} + d'\rangle \xrightarrow{\ell} \end{array}}{SYS_{\prec}\langle \mathsf{C}, \mathsf{PR}, \mathsf{RP}, \Theta, H, \Gamma, \natural, \mathsf{time}\rangle \xrightarrow{d} SYS_{\prec}\langle \mathsf{C}', \mathsf{PR}', \mathsf{RP}', \Theta', H', \Gamma', \natural', \mathsf{time}'\rangle} \; 14$$

5 Adding parameters

To facilitate the specification of large (even infinite) TPOs where the set of tasks follows a certain syntactical pattern, we introduce the concept of (typed) parameters. Every task is associated a set of index variables (i.e., the parameters) which can be used to compactly represent a set of similar precedence rules. We formalize this structure now.

Let TYPE be the set of all *types*, where every type is a set of *values*. The set of all values, i.e. the set $\bigcup_{T \in \mathsf{TYPE}}$, is denoted VAL. We assume that TYPE contains standard types like Bool, Nat and Int, as well as type TIME.

Let UVAR be a set of *"undecorated" variables*, with a function $\mathsf{type} \in \mathsf{UVAR} \to \mathsf{TYPE}$ assigning every variable a type. Every variable $x \in \mathsf{UVAR}$ can be decorated with a prime. The set DVAR of *decorated variables* is defined as $\mathsf{DVAR} = \{x' \mid x \in \mathsf{UVAR}\}$ and we set $\mathsf{VAR} = \mathsf{UVAR} \cup \mathsf{DVAR}$. The function type is extended to VAR by setting $\mathsf{type}(x') = \mathsf{type}(x)$ for every $x \in \mathsf{UVAR}$.

Let EXP be a set of *expressions*, built over values and variables. We assume EXP to be strongly typed, i.e. that we can extend type to EXP and determine $\mathsf{type}(e) \in \mathsf{TYPE}$ for every $e \in \mathsf{EXP}$. We define $\mathsf{BoolEXP} = \{e \in \mathsf{EXP} \mid \mathsf{type}(e) = \mathsf{Bool}\}$ and similarly NatEXP and $\mathsf{TimeEXP}$. The function $\mathsf{var} : \mathsf{EXP} \to \mathcal{F}(\mathsf{VAR})$ returns the set of (free) variables in an expression. If $\mathsf{var}(e) \subseteq \mathsf{UVAR}$ then e' is the expression obtained from e by decorating each variable.

A *valuation* is a partial function $\sigma : \mathsf{VAR} \hookrightarrow \mathsf{VAL}$ satisfying $\sigma(x) \in \mathsf{type}(x)$ for every $x \in \mathrm{dom}(\sigma)$. The set of all valuations is Σ. If $e \in \mathsf{EXP}$ and $\mathsf{var}(e) \subseteq \mathrm{dom}(\sigma)$, then $\sigma(e)$ can be evaluated to a value in VAL.

Given a valuation $\sigma \in \Sigma$ with $\mathrm{dom}(\sigma) \subseteq \mathsf{UVAR}$, we define $\sigma' \in \mathsf{DVAR} \hookrightarrow \mathsf{VAL}$ by $\sigma'(x') = \sigma(x)$ for each $x \in \mathrm{dom}(\sigma)$. Moreover, we set $\sigma'(e') = \sigma(e)$ whenever applicable.

Definition 1 (Parameterized Timed Partial Order (PTPO))
A PTPO is a tuple (T, V, S, P, δ), where

- *T is a finite set of* task classes*,*
- *$V : T \to \mathcal{F}(\mathsf{UVAR})$ associates a finite set of (index) variables to every task class,*
- *$S : T \to \mathsf{BoolEXP}$, with $\mathsf{var}(S(t)) \subseteq V(t)$ for all $t \in T$, determines the scope of parameters for every task class (in form of a condition),*

$\mathsf{UVAR} = \{n, m, k\}$
$\mathsf{type}(n) = \mathsf{type}(m) = \mathsf{type}(k) = \mathsf{Nat}$
$V(A) = \{n\}$, $V(B) = \{n, k\}$, $V(C) = \{m, n\}$
$G(n) \equiv 1 \leq n \leq 10$
$S(A) \equiv G(n)$
$S(B) \equiv G(n) \wedge k \leq 7 \wedge (n - k)\%3 = 0$
$S(C) \equiv G(n) \wedge m + n < 20$
$P(A, \mathsf{EC}, B) \equiv n' = n + k$
$P(A, \mathsf{EE}, C) \equiv m \geq n$
$\delta(A, \mathsf{EE}, C) \equiv m/2$

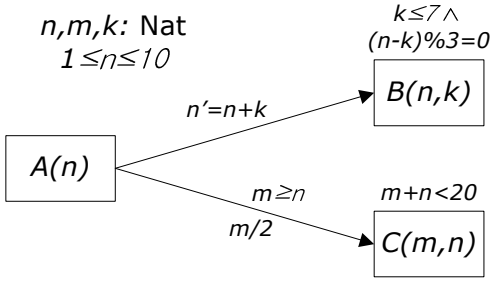

Figure 2: A PTPO and its visualization

- $P : (T \times \mathsf{RULE} \times T) \to \mathsf{BoolEXP}$, *with* $\mathsf{var}(P(t, u)) \subseteq V(t) \cup V(u) \cup \{x' \mid x \in V(t) \cap V(u)\}$ *for all* $(t, u) \in \mathrm{dom}(P)$, *gives a condition under which the precedence relation between two task class instances should exist (shared variables must be decorated in u to avoid ambiguity), and finally,*
- $\delta : (T \times \mathsf{RULE} \times T) \to \mathsf{TimeEXP}$, *with* $\mathrm{dom}(\delta) \subseteq \mathrm{dom}(P)$, *assigns a time constraint to every precedence rule.*

Figure 2 gives an example of a PTPO and shows its visualization. Parameterized TPOs are visualized similarly to regular TPOs but with several extensions. All the variables that are connected to a certain task class by means of function V are shown as (functional) parameters of this class. The scope condition of a class (specified by S) is shown above the node representing this class (omitted if constant true). If every event class contains the same boolean subcondition appearing in S, then this subcondition is shown separately (condition $G(n)$ in Figure 2), outside the graph, together with the set of variable declarations UVAR. Arc labels contain both the conditional expression determined by P (the label is omitted if constant true, and the arc is not shown if P is false) and the numeric expression determined by δ (omitted if constant 0).

PTPOs are only used to define large (possible infinite) TPO's concisely; they add no expressive power to regular TPOs or, i.e., every PTPO can be converted to a TPO. Given a PTPO (T, V, S, P, δ) and a $t \in T$, $t_V = \{(t, \sigma) \mid \sigma \in \Sigma \wedge \mathrm{dom}(\sigma) = \mathsf{UVAR} \wedge \sigma(S(t)) = \mathsf{true}\}$ is the set of all *instances* of t. We define $\bar{T} = \bigcup\{t_V \mid t \in T\}$ and $\bar{P} = \{((t, \sigma_t), r, (u, \sigma_u), (\sigma_t \cup \sigma'_u)(\delta(t, r, u))) \in \bar{T} \times \mathsf{RULE} \times \bar{T} \times \mathsf{TIME} \mid (\sigma_t \cup \sigma'_u)(P(t, r, u)) = \mathsf{true}\}$. The pair $(\bar{T}, \bar{P})$ is called the *parameter-unfolding* of (T, V, S, P, δ). A PTPO is called *consistent* iff its parameter-unfolding is consistent.