# Parametric Verification of Weighted Systems

**Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, Julian Trier Ringsmose, Kim Guldstrand Larsen, and Radu Mardare**

**Aalborg University**
**Selma Lagerlöfs Vej 300, Denmark**
`{pchri10, mhan10, amarie10, jrings10} @student.aau.dk`
`{kgl, mardare} @cs.aau.dk`

**Abstract**

This paper addresses the problem of parametric model checking for weighted transition systems. We consider transition systems labelled with linear equations over a set of parameters and we use them to provide semantics for a parametric version of weighted CTL where the until and next operators are themselves indexed with linear equations. The parameters change the model-checking problem into a problem of computing a linear system of inequalities that characterizes the parameters that guarantee the satisfiability. To address this problem, we use *parametric* dependency graphs (PDGs) and we propose a global update function that yields an assignment to each node in a PDG. For an iterative application of the function, we prove that a fixed point assignment to PDG nodes exists and the set of assignments constitutes a well-quasi ordering, thus ensuring that the fixed point assignment can be found after finitely many iterations. To demonstrate the utility of our technique, we have implemented a prototype tool that computes the constraints on parameters for model checking problems.

**1998 ACM Subject Classification** D.2.4 Software/Program Verification

**Keywords and phrases** parametric weighted transition systems, parametric weighted CTL, parametric model checking, well-quasi ordering, tool

**Digital Object Identifier** 10.4230/OASIcs.SynCoP.2015.77

## 1 Introduction

Specification and modelling formalisms that address non-functional properties of embedded and distributed systems have been intensively studied in the last decades. In particular the modelling formalism of timed automata [3] has established itself as a very useful formalism for expressing and analysing timing constraints of systems with respect to timed logics such as TCTL [2] and MTL [20]. This has naturally sparked interest for formal verification of functionality, and model checking techniques have often been used. However, timing constraints are not the only non-functional properties of interest in applications. Often modelling resources that can be consumed and must be monitored during the evolution of a system, such as energy, is an important issue in applications. This initially led to weighted extensions of timed automata [7, 5] and more recently to energy automata [9]. Such formalisms are typically abstracted as weighted transition systems, which are transition systems having the transition labelled by real numbers (or vectors of reals) that represent the resource consumption; and weighted versions of temporal logics are used to encode and verify logical properties.

In this paper we consider a parametric extension of the concept of weighted transition system by allowing the transition labels to be not only numbers, but also linear equations over a given set of parameters. Similarly, a parametric version of weighted CTL is defined, where

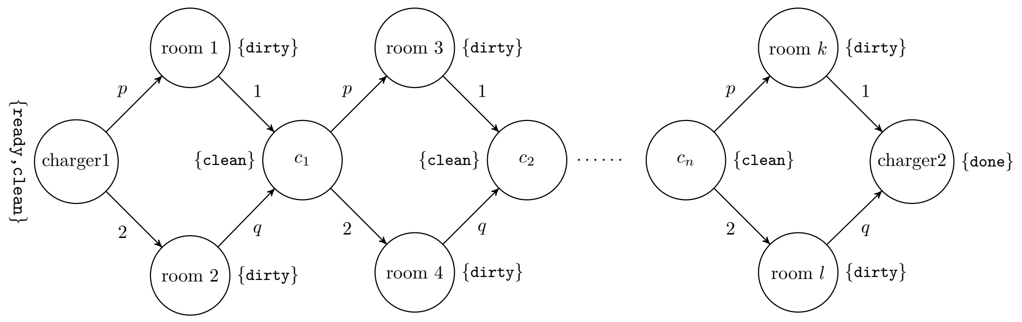


**Figure 1** Model $\mathcal{M}_{ex}$ for the robotic vacuum cleaner example.

the temporal operators are themselves indexed with linear equations representing upper bounds for the computation traces. In this context, we address the problem of parametric model checking: given a parametric model and a parametric logical property, compute a linear system of inequalities that describe the possible values of the parameters that will guarantee that the model satisfies the property.

As an example, consider a robotic vacuum cleaner. We want to know whether, in a given situation, our robot can reach and clean a dirty room within some time frame. While we know how many rooms there are, we do not have full knowledge of the time spent by the robot from room to room and neither do we know how long time a room takes to clean. The knowledge of the problem can be represented graphically as depicted in Figure 1, where the parameters $p, q$ represent our lack of knowledge. We want to check whether the robot can finish the job under some linear constraints over the parameters. This is the kind of problems we aim to address and solve in this paper.

The main contribution of this work, is the definition of a global function which iteratively computes a fixed point on the so-called *parametric dependency graphs* (PDGs). The PDGs in this paper are structures that handle parametric quantitative properties in a manner inspired by the Symbolic Dependency Graphs proposed in [19]. We use PDGs to represent problems by stating the dependencies between a given problem and its sub-problems along with their parametric quantitative constraints on the solution to the problem. We use Tarski's Fixed Point Theorem [24] to show that a fixed point exists. In this regard, we also prove that it can be reached in a finite number of steps using Dickson's Lemma [13] for well-quasi orders. Using these results, we show how to build PDGs for encoding of model checking, where parameters exist on both model and formula. Parametric model checking then entails finding constraints on parameter interpretation that makes a property satisfied. In some cases the constraints may be either trivially satisfied or not satisfiable, making it possible to give an yes/no answer. However, in the general case the answer is the set of parameter constraints obtained as a system of linear inequalities. Our technique can be further used to solve related issues such as bisimulation checking for parametric models ([10]).

In addition to the theoretical work of this paper, we have implemented a prototype tool (PVTool) that computes the fixed point for verification of parametric properties on parametric models. The input for the tool is a model that may have parametric weights on transitions, together with a state and a formula. Outputted is then a derivation of the exact constraints for interpretations of the parameters in the model and formula for the property to be satisfied by the model. Finally, we have conducted a series of preliminary performance experiments using the tool.

**Related Work.** In recent years, various extensions to modeling formalisms and logics have been developed. In [23], a Parametric Kripke Structure and parametric $CTL$ is presented along with algorithms computing constraints on parameters that must be satisfied for a parametric formula to be satisfied in a state. This inspired the authors of this paper to study the behavior of parametric weighted transition systems in [8] where various notions of bisimulation on parametric systems are discussed. In the world of automata, [4] proposes an extension to Timed Automata with parametric guards and invariant called Parametric Timed Automata. For parameter synthesis [6] uses an *inverse* method relying on an initial *good* parameter valuation for each parameter used to compute a constraint on parameters. It is then shown that if some parameter valuation satisfies the constraint, the system is behaviorally equivalent (identical traces) to the system under the initial valuation. This method is in contrast to traditional methods such as [15] based on *Counterexample Guided Abstraction Refinement* (CEGAR, [12]) where the goal is to avoid a set of *bad* states given beforehand. Tool support for various formalisms have also been developed. In [16], parameter synthesis of Linear Hybrid Automata in HyTech is discussed and [18] provides an extension to UPPAAL (see [21]) capable of generating parameter constraints that are necessary and sufficient for a reachability or invariant property to hold for Linear Parametric Timed Automata. Related to our work is also [19] where Dependency Graphs, introduced by Liu and Smolka (see [22]), are extended to Symbolic Dependency Graphs suitable for model checking using Weighted CTL and Weighted Kripke Structures. They compute a fixed point on *assignments* to nodes, being simply the cost of satisfying a formula in the weighted setting. We extend this to the parametric setting where we guarantee termination by exploiting that assignments can be interpreted as a well-quasi ordering. Refer to [14, 1] for a discussion of well-quasi orders for termination of algorithms operating on infinite structures.

## 2 Parametric Weighted Models and Logic

We now introduce various concepts used throughout this paper. We let $Q$ denote either universal- or existential-quantification, i.e. $Q \in \{A, E\}$. Furthermore, we let $\min\{\emptyset\} = \infty$ and $\max\{\emptyset\} = 0$ and denote the set of natural numbers, including zero, by $\mathbb{N}$.

A *Parametric Weighted Transition System* (PTS) is an extension of weighted transition systems with linear expressions of parameters as labels on the transitions. From here on, $\mathcal{P}$ denotes a fixed finite set of *parameters*. $\mathcal{E}$ is the set of all linear expressions on the form $\sum x_i p_i + y$ where $x_i, y \in \mathbb{N}$ and $p_i \in \mathcal{P}$ for any $i \in \mathbb{N}$.

Finally, PTS label states with a set of atomic propositions from the fixed finite set $\mathcal{AP}$.

We now proceed by formally defining a PTS:

▶ **Definition 1.** A *Parametric Weighted Transition System (PTS)* $\mathcal{M}$ is a triple

$$\mathcal{M} = (M, \rightarrow, \boldsymbol{\ell}), \text{ where}$$

- $M$ is a finite non-empty set of *states*.
- $\rightarrow \subseteq M \times \mathcal{E} \times M$ is the transition relation.
- $\boldsymbol{\ell} : M \longrightarrow 2^{\mathcal{AP}}$ is a labeling function mapping states in $M$ to a set of atomic propositions

Whenever $(m, e, m') \in \rightarrow$ we use the shorthand notation $m \xrightarrow{e} m'$. We will use $\mathfrak{M}$ to denote the set of all PTSs.

A run $\rho$ in a PTS $\mathcal{M} = (M, \rightarrow, \boldsymbol{\ell})$ is a possibly infinite sequence $\rho = m_0 e_1 m_1 e_2 m_2 ...$ where $\forall i \in \mathbb{N}, m_i \in M$ and for $i > 0, e_i \in \mathcal{E}$ we have $m_{i-1} \xrightarrow{e_i} m_i$. A run is *maximal* if it is infinite or the last state in the run has no outgoing transitions. $Runs(m)$ denotes all maximal runs $\rho$ starting from $m$ and $Runs(\mathcal{M})$ denote all maximal runs in $\mathcal{M}$. $\rho(i)$ denotes the state of $\rho$ with index $i$. Finally $|\rho|$ denotes the length of a run $\rho$ given by the number of states. Runs over PTSs accumulate sums of linear expressions from the transitions in the run. Given a position $j \in \mathbb{N}$ in a run $\rho$, let $\rho(j) = m_j$. The *accumulated weight*, $\mathcal{AW}_\rho(j)$, of $\rho$ at position $j$ is then defined by $\mathcal{AW}_\rho(j) = \sum_{i=1}^{j} e_i$ if $j > 0$ and $\mathcal{AW}_\rho(j) = 0$ if $j = 0$. We denote by $\boldsymbol{out}(m)$ the set of all expression-successor pairs of outgoing transitions from $m$.

The notion of interpretations, presented next, was first introduced in [8]. We will briefly explain the concept of interpretations; for further discussions, see [8]. Interpretations on the set of parameters is a mapping of each parameter to a natural number. As in [8], we will first define the simplest form of interpretations, namely the interpretation directly on parameters.

▶ **Definition 2.** $\boldsymbol{i} : \mathcal{P} \longrightarrow \mathbb{N}$ is a function mapping each parameter to a natural number.

We denote the set of all interpretations by $\mathfrak{I}$.

Interpretations are extended to the domain $\mathcal{E}$, by letting for an arbitrary $x \in \mathbb{N}$, $\boldsymbol{i}(x) = x$ and by requiring that $\boldsymbol{i}$ preserves polynomial structures, i.e., for arbitrary $p_1, ..., p_k \in \mathcal{P}$ and $x_1, ...x_{k+1} \in \mathbb{N}$:

$$\boldsymbol{i}(x_1p_1 + ... + x_kp_k + x_{k+1}) = x_1\boldsymbol{i}(p_1) + ... + x_k\boldsymbol{i}(p_k) + x_{k+1}$$

To reason about parametric properties of PTS, we define a parametric extension of Weighted Computation Tree Logic (WTL) called *Parametric Temporal Logic* (PTL) and give a satisfiability relation of formulae w.r.t. PTS. Interpretations are used to interpret both formulae bounds and PTS edge expressions.

▶ **Definition 3.** The set of PTL *state formulae* are given by the abstract syntax:

$$\Phi, \Psi ::= \top \mid \bot \mid \mathtt{a} \mid \Phi \wedge \Psi \mid \Phi \vee \Psi \mid E\varphi \mid A\varphi$$

and the set of PTL *path formulae* are given by the abstract syntax:

$$\varphi ::= X_{\leq e}\Phi \mid \Phi U_{\leq e}\Psi$$

where $\mathtt{a} \in \mathcal{AP}$ and $e \in \mathcal{E}$.

Whether a PTL formula is *satisfied* by a state $m$ or a run $\rho$ of some PTS $\mathcal{M}$ with an interpretation $\boldsymbol{i}$, is given by the satisfiability relation $\models_{\boldsymbol{i}}$. We denote this by $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$ and $\mathcal{M}, \rho \models_{\boldsymbol{i}} \varphi$, respectively.

▶ **Definition 4.** Given a PTL formula, a PTS $\mathcal{M} = (M, \rightarrow, \boldsymbol{\ell})$, a state $m \in M$ or a run $\rho \in Runs(\mathcal{M})$ and an interpretation $\boldsymbol{i} \in \mathfrak{I}$, the *satisfiability relation* $\models_{\boldsymbol{i}}$ is inductively defined as:

| | | |
|---|---|---|
| $\mathcal{M}, m \models_{\boldsymbol{i}} \top$ | | always |
| $\mathcal{M}, m \models_{\boldsymbol{i}} \bot$ | | never |
| $\mathcal{M}, m \models_{\boldsymbol{i}} \mathtt{a}$ | iff | $\mathtt{a} \in \boldsymbol{\ell}(m)$ |
| $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi \wedge \Psi$ | iff | $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$ and $\mathcal{M}, m \models_{\boldsymbol{i}} \Psi$ |
| $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi \vee \Psi$ | iff | $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$ or $\mathcal{M}, m \models_{\boldsymbol{i}} \Psi$ |
| $\mathcal{M}, m \models_{\boldsymbol{i}} E\varphi$ | iff | there exists $\rho \in Runs(m)$, such that $\mathcal{M}, \rho \models_{\boldsymbol{i}} \varphi$ |
| $\mathcal{M}, m \models_{\boldsymbol{i}} A\varphi$ | iff | for all $\rho \in Runs(m)$, it is the case that $\mathcal{M}, \rho \models_{\boldsymbol{i}} \varphi$ |
| $\mathcal{M}, \rho \models_{\boldsymbol{i}} X_{\leq e}\Phi$ | iff | $\|\rho\| > 0$, $\boldsymbol{i}(\mathcal{AW}_\rho(1)) \leq \boldsymbol{i}(e)$ and $\mathcal{M}, \rho(1) \models_{\boldsymbol{i}} \Phi$ |
| $\mathcal{M}, \rho \models_{\boldsymbol{i}} \Phi U_{\leq e}\Psi$ | iff | there exists $j \in \mathbb{N}$ s.t. $\mathcal{M}, \rho(j) \models_{\boldsymbol{i}} \Psi$ where $\boldsymbol{i}(\mathcal{AW}_\rho(j)) \leq \boldsymbol{i}(e)$ and $\mathcal{M}, \rho(j') \models_{\boldsymbol{i}} \Phi$ for all $j \in \mathbb{N}, j' < j$ |

## 3 Analysis of Parametric Properties

In this section we present a global method for analysis of various parametric problems using fixed point computations on Parametric Dependency Graphs (PDG). In this work we show how to abstract model checking problems into finding minimal fixed point assignments on PDGs specifically built for model checking. Refer to the technical report [10] for missing proofs.

### 3.1 Fixed Point Computations on Parametric Dependency Graphs

As presented in [19], Symbolic Dependency Graphs (SDG) can be used as an abstraction of problems with quantitative dependencies. We give a parametric extension to SDGs called *Parametric Dependency Graph* (PDG). This section will introduce PDGs in a general and formal fashion without context. Intuition and example of application is given in Section 3.2, where we show how to encode the model checking problem.

As the name suggests, Dependency Graphs encode dependencies. These dependencies may arise from the optimal substructure of a given problem. We encode this through the notion of *hyper-edges* with multiple targets, one for each dependency. In the most general sense the notion of a *cover-edge* states an upper bound constraint for the *cost* of some sub-property to be true, where cost is encoded as expression weights on the hyper-edges.

▶ **Definition 5.** A *Parametric Dependency Graph* (PDG) is a tuple $\mathcal{G} = (N, H, C)$, where:

- $N$ is a finite set of *nodes*,
- $H \subseteq N \times 2^{\mathcal{E} \times N}$ is a finite set of *hyper-edges*.
- $C \subseteq N \times \mathcal{E} \times N$ is a finite set of *cover-edges*.

Whenever $(n, T) \in H$ we refer to $n$ as the *source* node and $T$ as the *target-set*. For each $n' \in T$ we refer to $n'$ as a *target* node, or simply *target*. We will use $n \overset{e}{\dashrightarrow} n'$ whenever $(n, e, n') \in C$.

The set of all PDGs will be denoted by $\mathfrak{G}$. We will now proceed to define assignments which are used to encode the parametric cost for reaching a truth value in PDGs and note that the assignments form a complete lattice.

▶ **Definition 6.** Given a PDG $\mathcal{G} = (N, H, C)$, an assignment

$$A : N \longrightarrow (\mathfrak{I} \longrightarrow \mathbb{N} \cup \{\infty\})$$

on $\mathcal{G}$ is a mapping from each node $n \in N$ to a function that, given a parameter interpretation, yields a natural number or $\infty$.

We denote the set of all assignments $\mathfrak{A}$. The partial ordering over $\mathfrak{A}$ is defined as follows:

▶ **Definition 7.** $(\mathfrak{A}, \sqsubseteq)$ is a poset such that for $A_1, A_2 \in \mathfrak{A}$:

$$A_1 \sqsubseteq A_2 \quad \text{iff} \quad \forall n \in N \, \forall \boldsymbol{i} \in \mathfrak{I} : A_1(n)(\boldsymbol{i}) \geq A_2(n)(\boldsymbol{i})$$

$(\mathfrak{A}, \sqsubseteq)$ is clearly a complete lattice with $A^0$ and $A^\infty$ as top and bottom element, respectively.

Let $A^0$ denote the assignment that maps to each node a function that assigns the value 0 regardless of parameter interpretations, i.e. $\forall n \in N \, \forall \boldsymbol{i} \in \mathfrak{I} : A^0(n)(\boldsymbol{i}) = 0$. Similarly $A^\infty$ denotes the assignment that maps to each node a function that assigns the value $\infty$ regardless of parameter interpretations.

Generally when computing assignments, we use $\infty$ to represent negative results (infinite cost) and 0 (no cost) to represent positive results. As usual, when computing a minimal fixed point, we start from the bottom element, $A^\infty$ and similarly from the top element, $A^0$, for maximal fixed points.

We are now ready to define the global update function, which applied iteratively, updates PDG node assignments. Note that this function only considers PDG where for any node there is at most one outgoing *cover*-edge. This is not a limitation for this work as we only consider problems where one *cover*-edge is sufficient and one can easily extend the function to consider multiple *cover*-edges while preserving all results.

▶ **Definition 8.** Given a PDG $\mathcal{G} = (N, H, C)$, $\boldsymbol{F} : \mathfrak{A} \longrightarrow \mathfrak{A}$ is a function that, given an assignment $A$ on $\mathcal{G}$, produces a new assignment on $\mathcal{G}$ for any $n \in N, \boldsymbol{i} \in \mathfrak{I}$, as follows:

$$\boldsymbol{F}(A)(n)(\boldsymbol{i}) = \begin{cases} \begin{cases} 0 & \text{if } A(n')(\boldsymbol{i}) \leq \boldsymbol{i}(e) \\ \infty & \text{otherwise} \end{cases} & \text{if } n \overset{e}{\dashrightarrow} n' \\ \\ \min\limits_{(n,T)\in H} \{ \max\limits_{(e,n')\in T} \{A(n')(\boldsymbol{i}) + \boldsymbol{i}(e)\}\} & \text{otherwise} \end{cases}$$

We use $\boldsymbol{F}^i(A)$ to denote $i$ repeated applications of the function $\boldsymbol{F}$ on $A$, i.e
$\boldsymbol{F}^i(A) = \boldsymbol{F}(\boldsymbol{F}^{i-1}(\dots \boldsymbol{F}^1(A)))$ for $i > 0$ and $\boldsymbol{F}^0(A) = A$.

To show that $\boldsymbol{F}$ has a minimal fixed point, we observe by case inspection of Definition 8 that $\boldsymbol{F}$ is monotone with respect to the complete lattice $(\mathfrak{A}, \sqsubseteq)$ as stated by the following lemma.

▶ **Lemma 9.** *The update function $\boldsymbol{F}$ is monotone on the complete lattice $(\mathfrak{A}, \sqsubseteq)$.*

By Tarski's Fixed Point Theorem, we can conclude that $\boldsymbol{F}$ has a minimal and maximal fixed point. The assignment corresponding to these fixed points are denoted by $A^{min}$ and $A^{max}$. We now proceed to show that $A^{min}$ corresponds to $\boldsymbol{F}^i(A)$ for some $i \in \mathbb{N}$, ensuring that the minimal fixed point is computable in a finite number of steps. The key to this is the notion of well-quasi orders [14].

In general, assignments are functions that, given a node and an interpretation, compute a number. As our function is monotone w.r.t. $\sqsubseteq$ we know that the sequence of assignments computed by our function, given an interpretation, is decreasing for any node. We can therefore pick any interpretation and interpret an assignment $A$ as a tuple $(x_0, x_1, \dots, x_k)$ where $k+1$ is the number of nodes in the PDG and $x_i \in \mathbb{N} \cup \{\infty\}$ for all $0 \leq i \leq k$. Each iteration can then be interpreted as a function computing the next tuple in a (possibly) infinite sequence of tuples. Let the set of all such tuples be denoted by $\mathfrak{A}^T$ and let $A_i^T \in \mathfrak{A}^T$ denote the tuple computed by the $i$'th iteration and $\leq$ the component-wise ordering of tuples in $\mathfrak{A}^T$.

As $((\mathbb{N} \cup \{\infty\}), \leq)$ is a well-quasi order, we can use [17] (Theorem 2.3) saying that the Cartesian product of a finite number of well-quasi-ordered spaces is well-quasi-ordered, to state the following lemma.

▶ **Lemma 10.** *$(\mathfrak{A}^T, \leq)$ is a well-quasi-order.*

By the well quasi-ordering of $(\mathfrak{A}^T, \leq)$, we can now state that the fixed point computation ends in a finite number of steps, when applied iteratively on the bottom element $A^\infty$.

▶ **Theorem 11.** *There exists a natural number $i$ such that $A^{min} = \boldsymbol{F}^i(A^\infty)$.*

A similar result for the maximal fixed point does not exist. Consider a PDG consisting of only one node with a non-zero self loop. In this case, the next assignment is the old one plus the loop expression weight, thereby never reaching a fixed point.

▶ **Lemma 12.** *There exists a PDG such that $A^{max} \neq \boldsymbol{F}^i(A^0)$ for any $i \in \mathbb{N}$.*

## 3.2 Parametric Dependency Graphs For Model Checking

The verification of PTL properties differs from the usual notion of model checking in that it may not always be possible to give a Boolean answer. Instead we seek constraints on parameter interpretations that makes a PTL formula satisfiable by a given PTS state.

The construction rules in Figure 2 define the PDG construction for model checking.

As PDGs in this context are used to encode constraints on satisfiability of PTL formulae, they are constructed over a PTS model $\mathcal{M}$, a state $m$ in the model and a PTL formula $\Phi$. Each node $n \in N$ of a PDG $\mathcal{G}$ will be a pair $\langle m, \Phi\rangle$, where $\Phi$ is interpreted as a formula that may be satisfied in state $m$. A node $n$ therefore represents $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$. We notice that the problem may depend on sub-formulae of $\Phi$ and successors of $m$. We therefore use hyper-edges to connect the root $\langle m, \Phi\rangle$ to nodes representing these dependencies. For PTL path formulae we encode the parametric upper bounds as cover-edges in the PDG.

It should be clear from the PDG construction that by applying the update function $\boldsymbol{F}$ on an assignment to the PDG, any node that has an outgoing hyper-edge with an empty target set, i.e. nodes representing $\mathcal{M}, m \models_{\boldsymbol{i}} \top$ or $\mathcal{M}, m \models_{\boldsymbol{i}} \mathtt{a}$ where $\mathtt{a} \in \boldsymbol{\ell}(m)$, gets the value $\min\{\max\{\emptyset\}\} = 0$ thus representing satisfiability whereas nodes with no outgoing edges gets the value $\min\{\emptyset\} = \infty$ thus representing non-satisfiability. This shows the intuition behind the semantics of the values, that 0 represents satisfiability and $\infty$ represents non-satisfiability. In the context of model checking we compute a minimal fixed point, meaning that we start from the bottom element, $A^\infty$.

We now state the correctness theorem of the developed model-checking algorithm. The proof follows the correctness proof in [19], as models and formulae can be converted to a non-parametric setting using an interpretation. We denote by $\boldsymbol{B}(\mathcal{M}, m, \Phi)$ the PDG constructed by the rules in Figure 2 for the model checking problem $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$.

▶ **Theorem 13** (Correctness of Model-Checking Algorithm). *Let $\mathcal{M} = (M, \rightarrow, \ell)$ be a PTS, $m \in M$ a state, $\boldsymbol{i} \in \mathfrak{I}$ an interpretation and let $\mathcal{G} = \boldsymbol{B}(\mathcal{M}, m, \Phi) = (N, H, C)$ where $\langle m, \Phi\rangle \in N$. Then*

$$\mathcal{M}, m \models_{\boldsymbol{i}} \Phi \textit{ iff } A^{min}(\langle m, \Phi\rangle)(\boldsymbol{i}) = 0$$

The following example shows the construction of a PDG given a PTL formula and a PTS model and the application of Theorem 13 to derive parameter constraints.

▶ **Example 14.** Given the PTS $\mathcal{M}$ (Figure 3) and the formula

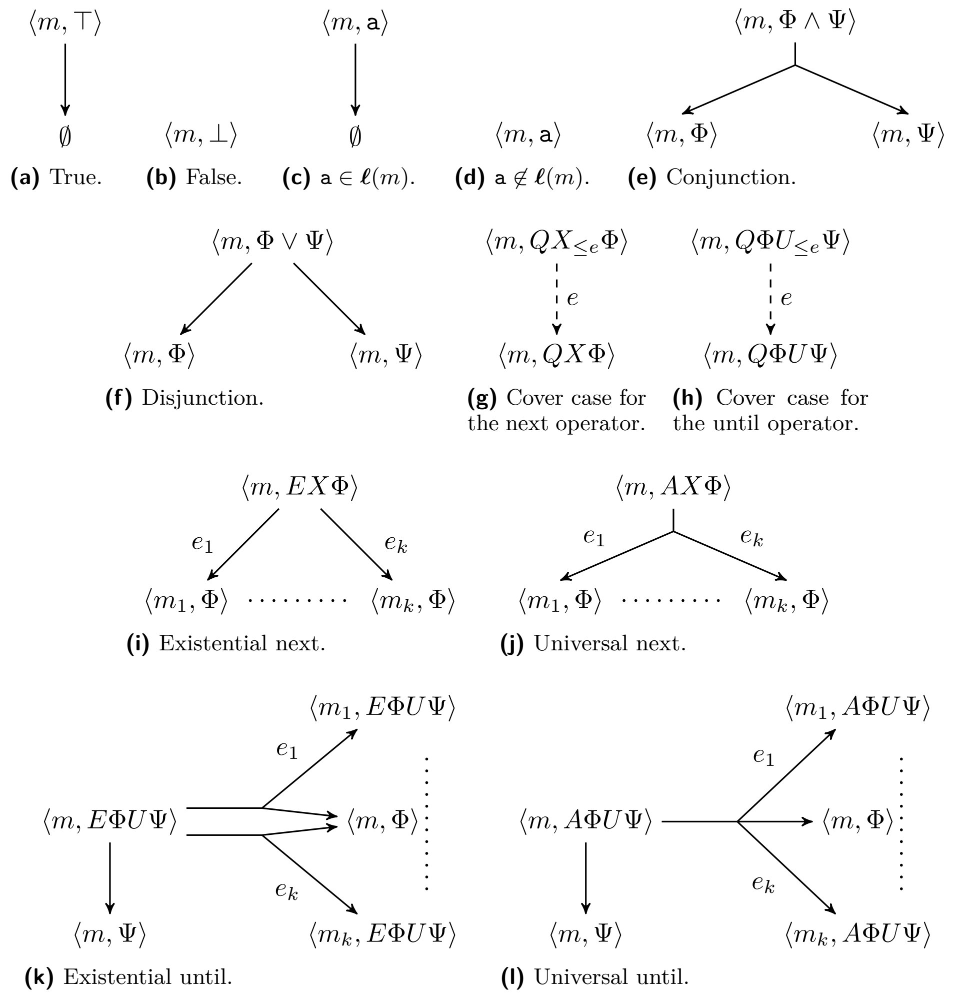
$$\Phi_{ex} = E\mathtt{b}U_{\leq 5}(\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}),$$

we now apply the construction rules in Figure 2 to encode the query $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi_{ex}$ as the PDG $\mathcal{G}$ in Figure 4.

By repeated application of the monotone function $\boldsymbol{F}$, we arrive at a fixed point after 7 iterations. Using the correctness theorem (and a few trivial simplifications) we get the exact constraints $\boldsymbol{i}(p) \leq 7 + \boldsymbol{i}(q)$ and $\boldsymbol{i}(q) \leq 5$ for $\boldsymbol{i}$ to be a valid interpretation. A quick look at the PTS should convince the reader of the correctness of these constraints.

## 4 Prototype Tool (PVTool)

We now present a proof of concept tool (PVTool [11]), being entirely web-based, for verification of parametric properties using the technique developed in this work. In Section 1, Figure 1 we presented an example of a parametric system which depicts a robotic vacuum cleaner. We now want to verify whether or not the cleaner can move from the starting location to



**(a)** True. **(b)** False. **(c)** $\mathsf{a} \in \ell(m)$. **(d)** $\mathsf{a} \notin \ell(m)$. **(e)** Conjunction.

**(f)** Disjunction. **(g)** Cover case for the next operator. **(h)** Cover case for the until operator.

**(i)** Existential next. **(j)** Universal next.

**(k)** Existential until. **(l)** Universal until.

**Figure 2** Let $\{(e_1, m_1), (e_2, m_2) \ldots (e_k, m_k)\} \in \boldsymbol{out}(m)$ and $Q \in \{A, E\}$.

one of the next rooms within 10 minutes units and clean it within 20 minutes. This question can be stated as follows:

$$\mathcal{M}_{ex}, charger1 \models_{\boldsymbol{i}} EX_{\leq 10}[A\ \texttt{dirty}\ U_{\leq 20}\ \texttt{clean}]$$

In the online PVTool we can now construct a model of the vacuum cleaner problem and state the PTL query and in return get the parameter constraints for the property to be satisfied in the model as depicted in Figure 5. We will briefly explain the model syntax used in the tool. The line "`charger1 := {clean, ready} <p>room1 + <2>room2;`" declares a state with the symbolic name *charger1* and two atomic propositions: *clean* and *ready*. The state has two outgoing transitions; one going to the state *room1* with the weight $p$ and one going to the state *room2* with the weight 2. Subsequent lines declare states in a similar manner. The formulae syntax is almost identical to the syntax used in the paper, with the exceptions that all sub-formulae must be enclosed in square brackets, bounds on path formulae must be enclosed on curly brackets and we do not write the $\leq$ sign. A thorough explanation of the syntax can be found on PVTool's web page.

To examine performance of the implementation we experiment with models that have a variable number of states. All experiments use models with a structure similar to the

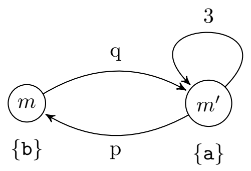
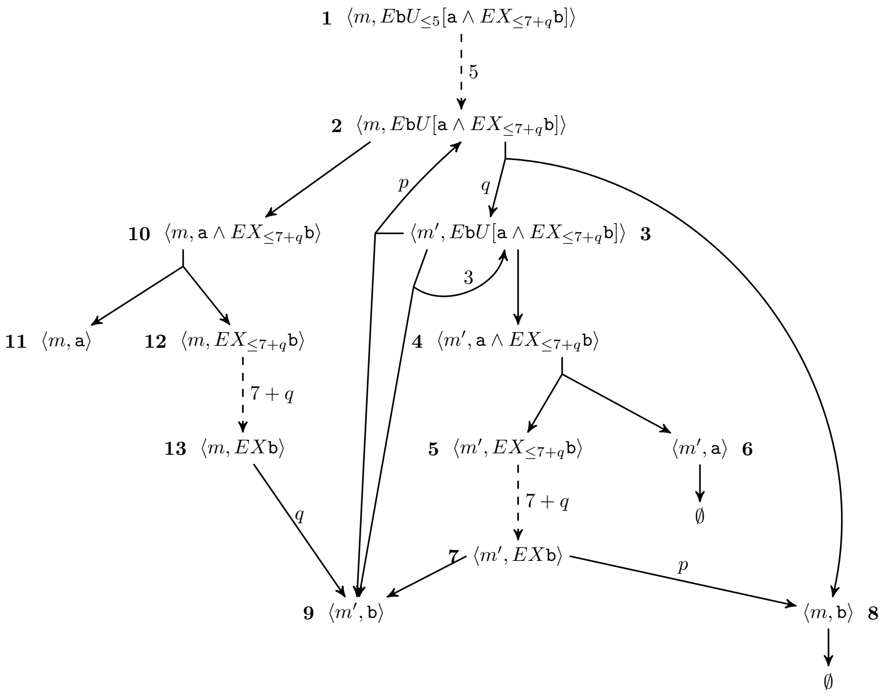


**Figure 3** PTS $\mathcal{M}$. $\boldsymbol{\ell}(m) = b, \boldsymbol{\ell}(m') = a$.



1 $\langle m, E\mathtt{b}U_{\leq 5}[\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}]\rangle$
5
2 $\langle m, E\mathtt{b}U[\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}]\rangle$
$p$
$q$
10 $\langle m, \mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}\rangle$
$\langle m', E\mathtt{b}U[\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}]\rangle$ 3
3
11 $\langle m, \mathtt{a}\rangle$
12 $\langle m, EX_{\leq 7+q}\mathtt{b}\rangle$
4 $\langle m', \mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}\rangle$
$7+q$
13 $\langle m, EX\mathtt{b}\rangle$
5 $\langle m', EX_{\leq 7+q}\mathtt{b}\rangle$
$\langle m', \mathtt{a}\rangle$ 6
$\emptyset$
$7+q$
$q$
7 $\langle m', EX\mathtt{b}\rangle$
$p$
9 $\langle m', \mathtt{b}\rangle$
$\langle m, \mathtt{b}\rangle$ 8
$\emptyset$

**Figure 4** PDG $\mathcal{G}$.

model depicted in Figure 1. We simply change the amount of rooms and intermediate “clean” states. For all experiments we use the PTL query

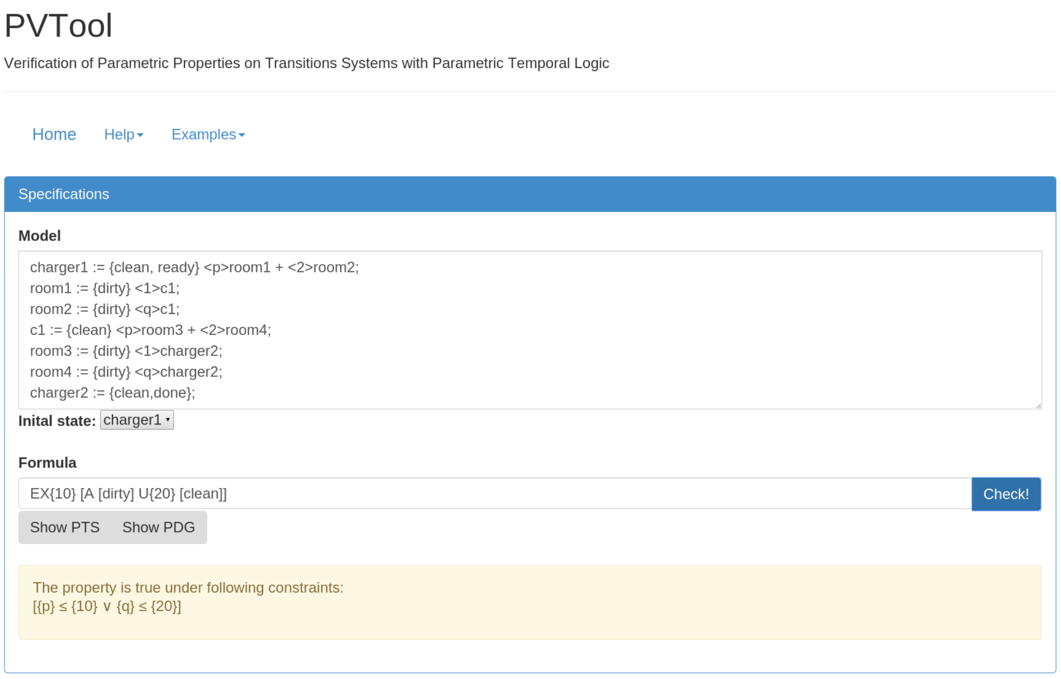
$$\mathcal{M}_{ex}, charger1 \models_{\boldsymbol{i}} E\ [A\ \mathtt{dirty}\ U_{\leq s}\ \mathtt{clean}]\ U_{\leq r}\ \mathtt{done}$$

Refer to Figure 6 to see the performance results. Memory and time consumption increase exponentially in the number of PTS states, reflecting the exponential increase in number of possible paths when scaling the PTS.

## 5 Conclusion and Future Work

We have defined a variant of Weighted CTL which uses parametric linear expressions as upper bounds on transition weights to allow reasoning about unknown behavior. We call this logic Parametric Temporal Logic (PTL). The semantics of PTL is defined for Parametric Weighted Transition Systems (PTS) that allow parametric linear expressions as weights on transitions.

For encoding of parametric propositional dependencies we present Parametric Dependency Graphs (PDGs). We associate to each node in a PDG a parametric cost and demonstrate



PVTool

Verification of Parametric Properties on Transitions Systems with Parametric Temporal Logic

Model

```
charger1 := {clean, ready} <p>room1 + <2>room2;
room1 := {dirty} <1>c1;
room2 := {dirty} <q>c1;
c1 := {clean} <p>room3 + <2>room4;
room3 := {dirty} <1>charger2;
room4 := {dirty} <q>charger2;
charger2 := {clean,done};
```

Inital state: charger1

Formula

EX{10} [A [dirty] U{20} [clean]]

The property is true under following constraints:
[{p} ≤ {10} ∨ {q} ≤ {20}]

**Figure 5** Screen shot of verification of PTL property on the model in Figure 1 using PVTool.

| PTS sts | PDG nds | Iter. | Mem (kB) | Time (s) |
|---|---|---|---|---|
| 7 | 41 | 9 | 1,004 | 0.0015 |
| 13 | 77 | 13 | 1,504 | 0.017 |
| 19 | 113 | 17 | 3,808 | 0.19 |
| 25 | 149 | 21 | 14,264 | 2.5 |
| 31 | 185 | 25 | 60,548 | 35 |
| 34 | 221 | 29 | 263,832 | 781 |

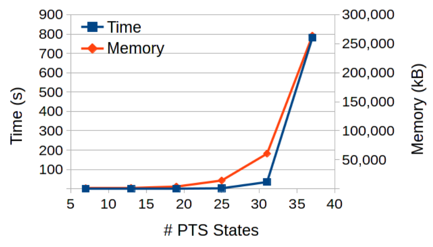


**Figure 6** Experiments on a 4 core Intel Xeon E3-1245 v2 3.4 GHz processor with 16 GB RAM.

that a maximal and minimal fixed point on these costs exists. We also show that the minimal fixed point can always be computed in a finite number of steps. Furthermore we show that deciding model checking PTS with PTL properties corresponds to finding the minimal fixed point in a PDG abstraction of model checking problems.

For verification of PTL properties on PTS we have implemented a model checking tool, PVTool ([11]) - available online, in which it is possible to define a PTS, give a PTL formula and get as output the constraints on parameters for a state in the PTS to satisfy the formula. In this context, preliminary experiments were made using an easily scalable PTS model to assess memory and time consumption which scaled exponentially in the size of the PTS, as expected.

As extension on this work, we propose to further look into other methods for updates on assignments to PDG nodes. In [19], the authors present a local fixed point algorithm for dependency graphs. If something similar can be done in the parametric setting we expect significant performance improvements. It would also be interesting to investigate whether a complete characterisation of PDGs for which the maximal fixed point can be computed in a finite number of steps, can be made. Finally one could look for new or existing problem domains where our method can be applied. A first step in this direction was done in the draft [10], for a variant of bisimilarity checking.

## References

**1** Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 313–321. IEEE, 1996.

**2** Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.

**3** Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *ICALP*, pages 322–335, 1990.

**4** Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601, 1993.

**5** Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *HSCC*, pages 49–62, 2001.

**6** Étienne André, Thomas Chatain, Laurent Fribourg, and Emmanuelle Encrenaz. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(05):819–836, 2009.

**7** Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*, pages 147–161, 2001.

**8** Sine Viesmose Birch, Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, and Julian T. Ringsmose. Weighted transition system: From boolean to parametric analysis. 8th semester project at Aalborg University, Department of Computer Science, 2014.

**9** Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In *FORMATS 2008. Proceedings*, pages 33–47, 2008.

**10** Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, and Julian T. Ringsmose. Parametric verification of weighted systems. Unpublished Technical Report. `http://pvtool.dk/tech_draft.pdf`.

**11** Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, and Julian T. Ringsmose. Prototype tool. `http://pvtool.dk/`.

**12** Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample guided abstraction refinement. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 154–169, 2000.

**13** Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):pp. 413–422, 1913.

**14** Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1):63–92, 2001.

**15** Goran Frehse, Sumit Kumar Jha, and Bruce H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 187–200. Springer Berlin Heidelberg, 2008.

**16** Thomas A. Henzinger and Howard Wong-Toi. Using hytech to synthesize control parameters for a steam boiler. In Jean-Raymond Abrial, Egon Börger, and Hans Langmaack, editors, *Formal Methods for Industrial Applications*, volume 1165 of *Lecture Notes in Computer Science*, pages 265–282. Springer Berlin Heidelberg, 1996.

17 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.

18 Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *J. Log. Algebr. Program.*, 52-53:183–220, 2002.

19 Jonas Finnemann Jensen, Kim Guldstrand Larsen, Jiří Srba, and Lars Kaerlund Oestergaard. Local model checking of weighted CTL with upper-bound constraints. In *Model Checking Software*, pages 178–195. Springer, 2013.

20 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

21 Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

22 Xinxin Liu and Scott A. Smolka. Simple linear-time algorithms for minimal fixed points. In *Automata, Languages and Programming*, pages 53–66. Springer, 1998.

23 Chaiwat Sathawornwichit and Takuya Katayama. A parametric model checking approach for real-time systems design. In *Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific*, pages 8–pp. IEEE, 2005.

24 Alfred Tarski et al. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

## A Fixed point computation results

In this appendix we show the fixed point computation from Example 14 in Table 1. For readability we have taken the liberty of simplifying the assignments based on properties of expressions and min / max functions valid under any interpretation of parameters. If any assignment to a node does not change (modulus our simplification rules) during the entire computation, we have omitted the node from the table. Let $A(n)$ be the assignment to some node $n$ and $e \in \mathcal{E}$ an expression.

- $\max\{\emptyset\} = \min\{0, \ldots\} = 0$
- $\min\{\emptyset\} = \max\{\infty, \ldots\} = \infty + A(n) = e + \infty = \infty$
- $0 + A(n) = A(n)$
- $e + 0 = e$
- Min/max of a single element is the element itself

Finally, let $\boldsymbol{F}^i(n)$ be a shorthand for $\boldsymbol{F}^i(A^\infty)(n)$. Using the correctness theorem we can now derive constraints that must be satisfied by an interpretation $\boldsymbol{i}$ for $\mathcal{M}, m \models_{\boldsymbol{i}} E\mathtt{b}U_{\leq 5}[\mathtt{a} \wedge EX_{\leq 7+\mathtt{q}}\mathtt{b}]$ to be true. From our simplified assignments it is easy to see that the constraints for $\boldsymbol{F}^7(\mathbf{1})(\boldsymbol{i}) = \boldsymbol{F}^8(\mathbf{1})(\boldsymbol{i}) = 0$ given some interpretation $\boldsymbol{i}$ must be $\boldsymbol{i}(p) \leq 7 + \boldsymbol{i}(q)$ and $\boldsymbol{i}(q) \leq 5$.

**Table 1** Updating assignments.

| $n$ | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| $A^\infty(n)(\boldsymbol{i})$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $\boldsymbol{F}^1(n)(\boldsymbol{i})$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $0$ | $\infty$ | $0$ |
| $\boldsymbol{F}^2(n)(\boldsymbol{i})$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $0$ | $\boldsymbol{i}(p)$ | $0$ |
| $\boldsymbol{F}^3(n)(\boldsymbol{i})$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\begin{cases} 0 & \text{if } \boldsymbol{i}(p) \leq 7 + \boldsymbol{i}(q) \\ \infty & \text{otherwise} \end{cases}$ | $0$ | $\boldsymbol{i}(p)$ | $0$ |
| $\boldsymbol{F}^4(n)(\boldsymbol{i})$ | $\infty$ | $\infty$ | $\infty$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $0$ | $\boldsymbol{i}(p)$ | $0$ |
| $\boldsymbol{F}^5(n)(\boldsymbol{i})$ | $\infty$ | $\infty$ | $\boldsymbol{F}^4(4)(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $0$ | $\boldsymbol{i}(p)$ | $0$ |
| $\boldsymbol{F}^6(n)(\boldsymbol{i})$ | $\infty$ | $\boldsymbol{i}(q) + \boldsymbol{F}^5(\mathbf{3})(\boldsymbol{i})$ | $\boldsymbol{F}^4(4)(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $0$ | $\boldsymbol{i}(p)$ | $0$ |
| $\boldsymbol{F}^7(n)(\boldsymbol{i})$ | $\begin{cases} 0 & \text{if } \boldsymbol{F}^6(\mathbf{2})(\boldsymbol{i}) \leq 5 \\ \infty & \text{otherwise} \end{cases}$ | $\boldsymbol{i}(q) + \boldsymbol{F}^5(\mathbf{3})(\boldsymbol{i})$ | $\boldsymbol{F}^4(4)(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $0$ | $\boldsymbol{i}(p)$ | $0$ |
| $\boldsymbol{F}^8(n)(\boldsymbol{i})$ | $\begin{cases} 0 & \text{if } \boldsymbol{F}^6(\mathbf{2})(\boldsymbol{i}) \leq 5 \\ \infty & \text{otherwise} \end{cases}$ | $\boldsymbol{i}(q) + \boldsymbol{F}^5(\mathbf{3})(\boldsymbol{i})$ | $\boldsymbol{F}^4(4)(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$ | $0$ | $\boldsymbol{i}(p)$ | $0$ |