

Computing Real Numbers with Large-Population Protocols Having a Continuum of Equilibria

Xiang Huang¹ ✉ 

Department of Computer Science, University of Illinois Springfield, USA

Rachel N. Huls ✉

Department of Mathematics, University of Illinois Springfield, USA

Abstract

Bournez, Fraigniaud, and Koegler [6] defined a number in $[0,1]$ as computable by their Large-Population Protocol (LPP) model, if the proportion of agents in a set of marked states converges to said number over time as the population grows to infinity. The notion, however, restricts the ordinary differential equations (ODEs) associated with an LPP to have only finitely many equilibria. This restriction places an intrinsic limitation on the model. As a result, a number is computable by an LPP if and only if it is *algebraic*, namely, not a single transcendental number can be computed under this notion.

In this paper, we *lift* the finitary requirement on equilibria. That is, we consider systems with a continuum of equilibria. We show that essentially all numbers in $[0,1]$ that are computable by bounded general-purpose analog computers (GPACs) or chemical reaction networks (CRNs) can also be computed by LPPs under this new definition. This implies a rich series of numbers (e.g., the reciprocal of Euler's constant, $\pi/4$, Euler's γ , Catalan's constant, and Dottie number) are all computable by LPPs. Our proof is constructive: We develop an algorithm that transfers bounded GPACs/CRNs into LPPs. Our algorithm also fixes a gap in Bournez et al.'s construction of LPPs designed to compute any arbitrary algebraic number in $[0,1]$.

2012 ACM Subject Classification Theory of computation \rightarrow Interactive computation; Theory of computation \rightarrow Probabilistic computation; Theory of computation \rightarrow Abstract machines

Keywords and phrases Population protocols, Chemical reaction networks, Analog computation

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.7

Funding *Xiang Huang*: This research was supported in part by NSF grants 1545028 and 1900716. *Rachel N. Huls*: This research was supported in part by the University of Illinois Springfield Leadership Lived Experience (LLE) initiative.

Acknowledgements We thank Titus Klinge for providing the motivating example in Section 1.

1 Introduction

1.1 GPAC/CRN Computable Number

The computation of real numbers provides a theoretical benchmark for an abstract model's computational power. In 1936, Turing [32] considered computable numbers while researching a discrete model now known as the Turing machine. Then, Shannon [30] introduced the general-purpose analog computer (GPAC) to the world. The GPAC is a mathematical abstraction of Bush's differential analyzer. Bush [8] developed this machine to obtain numerical solutions of ordinary differential equations (ODEs). After much revision [29, 27, 18, 17], we can now characterize GPACs by polynomial initial value problems (PIVPs):

¹ corresponding author



$$\begin{cases} \mathbf{x}' = \mathbf{p}(\mathbf{x}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0, \end{cases} \quad t \in \mathbb{R} \quad (1)$$

where \mathbf{p} is a vector of multivariate polynomials, \mathbf{x} is a vector of variables, and \mathbf{x}_0 is its initial value. Bournez et al. [7] have shown that GPACs are provably equivalent to Turing machines in terms of computability and complexity. Chemical reaction networks (CRNs), as a restricted form of GPACs, are widely used in molecular programming applications. The deterministic semantic of CRNs usually assumes well-mixed solutions and mass-action kinetics. CRN dynamics can be modeled by polynomial systems with extra constraints [21] (see the preliminary section for more details). Although the constraints could seemingly weaken the computational power of CRNs, they are able to simulate GPACs. Hence, the CRN model is also Turing complete [13]. The key idea is to encode every variable x (possibly having negative values) in a GPAC by the difference between a pair of *positive* variables x^+ and x^- in the copycat CRN. That is, $x(t) = x^+(t) - x^-(t)$ for all $t \geq 0$.

A straightforward idea to “compute” a number α by a GPAC/CRN is to designate a variable x such that $x(t) \rightarrow \alpha$ as $t \rightarrow \infty$. Further down this line, Huang et al. defined the class of real-time computable real numbers by chemical reaction networks [24], wherein the real-time computability notion essentially requires $x(t)$ to converge exponentially fast to α . Huang et al. then demonstrate that GPACs and CRNs compute the same class of real numbers in real-time [23]. Notably, famous transcendental numbers like e and π are among this class. Later, Huang [22] added Euler’s γ and Dottie number (the unique real root of the equation $\cos(x) = x$) to the list. Note that in order to get a meaningful measure of time, in Huang et al.’s work, all variables in ODEs associated with GPACs or CRNs are *bounded*. Hence, the notion of time aligns with the time parameter t of the ODEs since no more than a linear speedup is allowed. This notion prevents the so-called Zeno phenomena [19] caused by inordinate speedup. Another, perhaps equivalent, metric is the *arc length* of $\mathbf{x}(t)$. A *complexity theory* [7] of analog computation can then be built upon this metric.

Also, note that the notion Huang et al. used has a *descriptive* nature: the initial values are all zero, and rate constants are integrals (rationals). Hence, one can only encode a finite amount of information in the initial value and rate constant. In this sense, a system that computes a number α also gives a finite description of α . This idea will prove crucial for later discussions.

1.2 Large Population Protocols

Population protocols (PPs) [2] can be treated as special CRNs with two inputs, two outputs, and a unit rate constant per reaction. Observe that population protocols *conserve* the population since interactions do not change agent counts. This property imposes a vital constraint on the population: the sum of all agents remains constant. After normalization, we can say that this sum is *one* without loss of generality.

Classical PP are concerned with computing predicates over their input configuration in a discrete, stochastic setting. Therefore, computing real numbers does not make sense under this setting. In a series of papers [3, 5, 6], Bournez et al. developed a setting they termed a Large-Population Protocol (LPP). LPPs are a new analytical setting rather than a new computational model. They are still population protocols, but their behaviors are analyzed as the population grows to infinity. Bournez et al. aimed for asymptotic results independent of the initial configuration. Our approach departs from theirs in that dependence on initial

values is considered. Note that we should approach with caution when defining what is meant by “initial configuration” since LPPs need to talk about different population sizes along the way.

The LPP setting provides a playground for computing real numbers by population protocols: a number $\nu \in [0, 1]$ is said to be computable by an LPP if the proportion of agents in a set of *marked states* converges to ν over time as the population grows to infinity. An additional requirement is that the ODE induced by the *balance equation* of the LPP must have finitely many equilibria. This condition enforces (exponential) stability and detaches the dependence on initial values.

Bournez et al. specifically asked[6] “... is it possible to compute solutions of trigonometric equations? E.g., can we design a protocol insuring that, asymptotically, a ratio $\frac{\pi}{4}$ of the nodes are in some prescribed state?” Their answer is *negative*. They proved that, under their definition, LPPs compute exactly the algebraic numbers in $[0,1]$. Closely related is a result by Fletcher et al. [15], where a class of numbers, called Lyapunov numbers, are found to be the set of algebraic numbers. It is not a coincidence: both works involve ODEs with either finitely many or isolated equilibria. Hence, Tarski’s quantifier elimination over real closed fields can be performed. Alternatively, one can find more modern elimination algorithms involving Gröbner bases in standard algebraic geometry texts [10, 31].

Why is there a gap between LPPs (cannot compute transcendentals) and GPAC/CRN (can compute e and π)? The following example helps explore the question.

► **Motivating Example.** Let $F(t) = \frac{1}{2}e^{e^{-t}-1}$, $E(t) = \frac{1}{2}e^{-t}$, and $G(t)$ be a function such that its derivative “cancels” with E and F ’s derivative. We have the first-order system and the corresponding chemical reaction network:

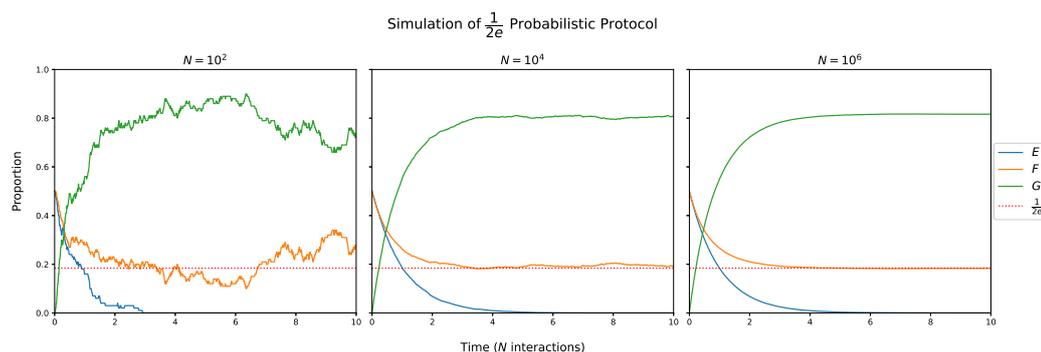
$$\text{ODE: } \begin{cases} F' = -2FE \\ E' = -E \\ G' = 2FE + E \end{cases} \quad \text{CRN/PP: } \begin{cases} F + E \xrightarrow{-2} G + E \\ E \rightarrow G. \end{cases}$$

It is evident that with initial values $F(0) = \frac{1}{2}$, $E(0) = \frac{1}{2}$, and $G(0) = 0$, the solution of $F(t) \rightarrow \frac{1}{2}e^{-1}$ as $t \rightarrow \infty$. Correspondingly, in the LPP setting, if we let $F(0) = \lfloor \frac{N}{2} \rfloor$, $E(0) = \lfloor \frac{N}{2} \rfloor$, and $G(0) = 0$ for a population of size N , we can observe from the experimental results in Figure 1 that the proportion of F gets very close to $\frac{1}{2}e^{-1}$ when N becomes larger. In fact, the so-called Kurtz’s Theorem [26], which says ODE solution $F(t)$ equals the proportion random variable $F^{(N)}(t)/N$ almost surely when N goes to infinity, guarantees the limit is $\frac{1}{2}e^{-1}$. Although the CRN in the above example is not technically a PP, we can translate it into a probabilistic PP (Construction 1, see also [11], Theorem 1). We maintain the protocol in the current form for simplicity. ◻

So the above LPP can “compute” $\frac{1}{2}e^{-1}$, a *transcendental* number! Is it a counter-example to Bournez et al.’s algebraic result? Not really. Since the system of ODEs in the example has the whole F - G plane (with $E = 0$) as equilibria, the system has a *continuum* of equilibria. Note that the key reason GPACs/CRNs in [23] can compute transcendentals is that they are systems with a continuum of equilibria.

1.3 Computing with a Continuum of Equilibria

The Motivating Example invites us to *extend* the notion of LPP-computable numbers: We can drop the finitary requirement on equilibria and therefore include systems with a continuum of equilibria. The seminal work [4] by Sanjay and Bernstein provides an in-depth study of this type of system.



■ **Figure 1** We use the ppsim Python package [11] to simulate our motivating example (transformed into a probabilistic PP). This figure demonstrates that the transcendental number $\frac{1}{2}e^{-1}$ can be traced by the proportion of a species as the population size grows.

The move has several consequences. An immediate one is that computation now *depends* on initial values: since the equilibria “stick together”, we need the initial values to determine which equilibrium the system converges to. Another one is stability: we lose asymptotic stability. As pointed out in [4], “... Since every neighborhood of a non-isolated equilibrium contains another equilibrium, a non-isolated equilibrium cannot be asymptotically stable. Thus asymptotic stability is not the appropriate notion of stability for systems having a continuum of equilibria.” However, it is inaccurate to immediately conclude that such systems must be fragile and sensitive to perturbations. Indeed, Sanjay and Bernstein investigated *semistability*, a more appropriate notion of stability for these systems.

We now state our new finding under the extended notion of LPP-computable numbers.

► **Main Theorem.** *LPPs compute the same set of numbers in $[0,1]$ as GPACs and CRNs.*

Our proof is constructive: We give an explicit algorithm to translate a bounded GPAC or CRN into an LPP. A critical step is the construction of a cubic form (homogeneously degree 3 polynomial) system that conserves the population. Then, we transform the system from the cubic form into the quadratic form. Finally, the system can be rewritten as a probabilistic population protocol.

The key feature of our construction is that we always guarantee the system is either *CRN-implementable* or *PP-implementable*. Bournez et al. constructed a system in [6] that claimed to compute any algebraic number. While their system is a GPAC, it is not a PP in general. As a by-product, our construction fixes this flaw.

Related work

Besides Bournez et al.’s work [6, 3, 5], Gupta, Nagda, and Devaraj [20] develop a framework for translating a subclass of differential equation systems into practical protocols for distributed systems (stochastic CRNs in most cases and, in some examples, population protocols). They also restated Kurtz’s theorem and considered a large-population setting. The subclasses they considered are *complete* and *completely partitionable*, which means the derivatives of the system sum to zero, and any term (monomial) T occurs as a pair $\{+T, -T\}$ in the system, a very nice property to have. Doty and Severson [11] provided an algorithm to translate CRNs that consist solely of unimolecular ($X \rightarrow Y$) and bimolecular reactions ($X + Y \rightarrow Z + W$) to PPs. Those reactions correspond to an ODE system’s linear terms (e.g., x) and quadratic terms (e.g., xy). Their assumption, however, restricts the application of their algorithm from

more general GPACs. Firstly, the CRNs they considered preserve populations. *Constant terms* in GPACs, which correspond to reactions like $\emptyset \rightarrow X$, *change* the population. Another consideration is the treatment of square terms. Some quadratic systems can not be *directly* written as a CRN with bimolecular reactions (i.e. if positive square terms like x^2 occur in x'). Later in this paper, we will discuss how the deep entanglement of the two (constant terms and squared terms) makes translating a general GPAC into a PP very difficult.

Our construction does not make any assumptions in the above work on GPACs, except that all variables are bounded.

The rest of this paper is organized as follows. Section 2 discusses preliminaries as well as some notations and conventions necessary to present our main result; Section 3 includes the main theorem, a four-stage constructive proof, and a few immediate corollaries. Section 4 provides some concluding remarks and discussion on future directions.

2 Preliminaries

We review some key definitions and basic facts in this section.

2.1 GPACs, CRNs, Population Protocols, and Their ODE Characterization

We have discussed that GPACs can be characterized by polynomial initial value problems (PIVPs) in the previous section. To align with the ODE descriptions of GPACs, we will use the ODE characterization of CRNs and PPs throughout this paper. This choice clarifies the discussion of our construction and proof in the coming sections.

We start with a few notations and conventions in this paper.

► **Notation (Variable Vector).** We use the notation $\mathbf{x}(t) = (x_1(t), x_2(t), \dots, x_n(t))$ to denote a vector of variables in (usually the solution of) an ODE system, where $n \in \mathbb{N}$ is the vector's dimension and the parameter t is regarded as “time” in the system. The first component x_1 is usually designated for tracing the number we wish to compute.

A typical definition of a CRN or PP specifies a pair (S, R) , with S being the set of species or states and R being the set of reactions or interactions between states. In general, a reaction for an abstract CRN looks like $X + Y \xrightarrow{k} X + W + Z$, where k is the *reaction constant*. There is no restriction on the number of reactants (input) or products (output). Reactions in population protocols, however, are strictly limited to those with two inputs and two outputs: $X + Y \rightarrow W + Z$.

The so-called *deterministic mass-action* semantic of a CRN defines a polynomial differential system, which governs the CRN's dynamics. One can find descriptions of this semantic in many works of literature (e.g., [23]). We will use these polynomial differential systems directly throughout the paper.

► **Convention.** We manipulate our use of the variable vector: when the context is clear, we simply say $(\mathbf{x}, \mathbf{x}')$, or \mathbf{x} itself is a GPAC (CRN, PP), without specifying the associated polynomial differential system $\mathbf{x}'(t) = (x'_1(t), \dots, x'_n(t))$. If we don't specify the initial values that means we assume them to be zero. We would assume the correspondence between a variable x and the species X that it describes. In some cases, we will call a variable x a species and $\mathbf{x} = (x_1, x_2, \dots, x_n)$ a set of species.

Now, we introduce notation and terminology for polynomials.

► **Notation** (Positive Polynomials). We denote $\mathbb{P}^+ \subset \mathbb{Q}[\mathbf{x}] = \mathbb{Q}[x_1, \dots, x_n]$ the set of multivariable polynomials with positive (rational) coefficients for each monomial (terms). For instance, $x_1x_2 + x_3 + 1 \in \mathbb{P}^+$, while $x_1x_2 - x_3 + 1 \notin \mathbb{P}^+$.

► **Notation** (Variable Occurrence in a Polynomial or Monomial). Let p be a polynomial (or monomial) and x be a variable, we use the convenient notation $x \in p$ to express x occurs in p .

Homogeneous polynomials play a fundamental role in our construction. We call them forms for short.

► **Terminology** (Forms). In mathematics, “form” is another name for homogeneous polynomials. A quadratic form is a homogeneous polynomial of degree two, for example, $4x^2 + 2xy - 3y^2$; a cubic form is a homogeneous polynomial of degree three, for example, $x^3 + 3x^2y + 3xy^2 + y^3$.

► **Terminology** (Conservative Systems). A conservative system has a (non-trivial) quantity that is constant along each solution. In this paper, we call a system that maintains constant total mass, i.e., a system $\mathbf{x} = (x_1, x_2, \dots, x_n)$ that satisfies

$$\sum_{i=1}^n x'_i = 0,$$

a conservative system.

We say a function $f(t)$ is implementable (generable) by a GPAC (CRN, LPP) if there exist a GPAC (CRN, LPP) and a variable (species, state) x in it, such that $f(t) = x(t)$ for all t . We choose the word *implementable* over *generable* (as in [18]) to reflect the idea that we *program* a GPAC (CRN, LPP) to *implement* a function.

► **Definition 1** (GPAC-implementable Functions [18]. See also Definition 3 in [13].). A function $f : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ is GPAC-implementable if it is the first component i.e. x_1 , of the $\mathbf{x}(t) = (x_1, x_2, \dots, x_n)$ solution for the following differential equation:

$$\begin{cases} \mathbf{x}' = \mathbf{p}(\mathbf{x}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0, \end{cases} \quad t \in \mathbb{R} \quad (2)$$

where \mathbf{p} is a vector of multivariate polynomials and \mathbf{x} is a variable vector with \mathbf{x}_0 as its initial value.

Note that if a variable x has initial value a , we can introduce a new variable $y = x - a$ to make $y(0) = 0$. Therefore, without loss of generality, we often assume $\mathbf{x}(0) = \mathbf{0}$ in this paper.

► **Theorem 2** (CRN-implementable Functions [21], Theorem 3.1 and 3.2). A function is CRN-implementable if and only if it is GPAC-implementable with further restriction such that the derivative of each component x_i in its variable vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is of the form

$$x'_i = p - qx_i, \quad \text{where } p, q \in \mathbb{P}^+. \quad (3)$$

The restriction is rooted in reaction modeling: a negative term in x' means to *destroy* some molecular x at some rate. However, a reaction cannot destroy a non-reactant, so x must appear as a reactant in the reaction. Such restriction extends to Population Protocols together with some extra constraints.

- **Corollary 3** (PP-implementable Functions). *A function is PP-implementable, if and only if*
- i. *it is CRN-implementable, i.e., there is a CRN $(\mathbf{x}, \mathbf{x}')$ computing it.*
 - ii. *for all x_i in $\mathbf{x} = (x_1, x_2, \dots, x_n)$, x_i' does not possess a positive x_i^2 term.*
 - iii. *\mathbf{x}' is a quadratic form system, and*
 - iv. *\mathbf{x}' is conservative.*

Proof. For the “only if” direction, all conditions other than condition (ii) are clear. To see that x_i' can not have an x_i^2 , we consider two X_i ’s on the left-hand side of a reaction. The reaction can not increase the amount of X_i , since that would require at least $k + 1$ X_i ’s as products on the right-hand side. As a result, a positive x_i^2 term can not occur in x_i' . See Theorem 16 for the “if” direction of the proof. ◀

In this paper, we will also call a CRN containing only unimolecular reactions of the form $X \rightarrow Y$ a *unimolecular* population protocol (UPP). Similarly, a CRN containing only termolecular reactions of the form $X + Y + Z \rightarrow U + V + W$ is called a *termolecular* population protocol (TPP). In general, a k -PP is a CRN that contains only reactions of the form $X_1 + \dots + X_k \rightarrow Y_1 + \dots + Y_k$. Generally, we have the following characterization.

- **Corollary 4** (k -PP-implementable Functions). *A function is k -PP-implementable, if and only if*
- i. *it is CRN-implementable, i.e., there is a CRN $(\mathbf{x}, \mathbf{x}')$ computing it.*
 - ii. *for all x_i in \mathbf{x} , x_i' does not possess a positive x_i^k term.*
 - iii. *\mathbf{x}' is homogeneously degree k , and*
 - iv. *\mathbf{x}' is conservative.*

From above, we see that turning a GPAC into a PP means having to dance with more and more chains. We must carefully appease all restrictions, which makes the translation process difficult.

We first revisit some useful concepts regarding PPs.

2.2 Probabilistic Large-Population Protocols

► **Definition 5** (Balance Equations [6]). *Let (Q, R) be a large-population protocol, where Q is the state (species, variables) set and R is a set of reaction rules, each with two reactants and two products. The balance equation of a deterministic LPP is a function $b : \mathbb{R}^{|Q|} \rightarrow \mathbb{R}^{|Q|}$ such that*

$$b(x) = \sum_{(q_1, q_2) \in Q^2} \left(x_{q_1} x_{q_2} (-e_{q_1} - e_{q_2} + \sum_{(q_3, q_4) \in Q^2} \delta_{q_1, q_2, q_3, q_4} (e_{q_3} + e_{q_4})) \right) \quad (4)$$

where $\delta_{q_1, q_2, q_3, q_4} = 1$ if $(q_1, q_2) \rightarrow (q_3, q_4)$, and 0 otherwise; $(e_q)_{q \in Q}$ is the canonical base of $\mathbb{R}^{|Q|}$ and x_{q_i} is the proportion of the population in state q_i .

The balance equation is a description of the system’s dynamics. Intuitively, it says whenever q_1 and q_2 bump into each other, if there is a reaction $(q_1, q_2) \rightarrow (q_3, q_4)$ in the PP, then the pair (q_1, q_2) turns into (q_3, q_4) ; otherwise, we interpret (q_1, q_2) as a *null reaction* and the agents remain unchanged. We adopt an *asymmetric* interpretation of reactions in PPs throughout the paper. That is, the ordered pairs (q_i, q_j) and (q_j, q_i) , when used as reactants, do not necessarily result in the same products. The assumption is not essential: the choice is to align with the existing literature (e.g., [6]) for convenience.

The above balance equation is for deterministic PPs, where $\delta_{q_1, q_2, q_3, q_4}$ is either 0 or 1. It is more convenient to use probabilistic transition rules in many scenarios. We introduce probabilistic LPPs (PLPPs).

7:8 Computing Real Numbers with LPPs Having a Continuum of Equilibria

► **Definition 6** (Probabilistic LPP (PLPP) [6]). *A PLPP is an LPP with transition rules in the form*

$$q_i q_j \rightarrow \alpha_{i,j,k,l} q_k q_l$$

and for every $(q_i, q_j) \in Q^2$, we have

- for every $(q_k, q_l) \in Q^2$, $\alpha_{i,j,k,l} \in \mathbb{Q}$ and $\alpha_{i,j,k,l} > 0$, and
- $\sum_{(q_k, q_l)} \alpha_{i,j,k,l} = 1$.

The balance equation remains mostly the same as Equation (4). To simplify the notations, we often take $Q = [n] = \{1, 2, \dots, n\}$ and will still use q_i or use terms like “state i ” when referring to a state.

$$b(x) = \sum_{(i,j) \in [n]^2} \left(x_i x_j (-e_i - e_j + \sum_{(k,l) \in [n]^2} \alpha_{i,j,k,l} (e_k + e_l)) \right) \quad (5)$$

The ODE associated with a PLPP can be written as

$$\frac{d\mathbf{x}}{dt} = b(\mathbf{x}), \quad (6)$$

where $\mathbf{x} = (x_1(t), \dots, x_n(t)) \in \mathbb{R}^n$ is a variable vector and its i -th component, x_i , tracks the proportion of the total population in state i over time t . We will often call $\mathbf{x} = (x_1, x_2, \dots, x_n)$ or \mathbf{x} an LPP (PLPP). ┘

A simple observation results directly from the definition of LPP or PLPP.

► **Observation 7** (“The one trick”). *Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be an LPP or PLPP, then*

$$\sum_{i \in [n]} x_i = 1.$$

Although obvious, the above is the single *most important* observation in this paper. Many constructions and proofs rely on it. This observation is frequently used to re-write the constant term in an ODE. We will rewrite a constant c in either of the following ways, depending on our needs:

- $c = c \cdot 1 = c(x_1 + \dots + x_n)$, or
- $c = c \cdot 1 \cdot 1 = c(x_1 + \dots + x_n)(x_1 + \dots + x_n)$. ┘

► **Observation 8.** *The r -th component of Equation (6) has the form*

$$\frac{dx_r}{dt} = f(\mathbf{x}) - 2x_r, \quad (7)$$

where $f(\mathbf{x})$ is a quadratic form in \mathbb{P}^+ .

An intuitive way to think about equation 7:

- The term $-2x_r$ represents all reactions that *consume* x_r . A complete set of rules for a PP must include all combinations of pairs (x_r, x_k) , or (x_k, x_r) , where k iterates through all $[n]$. These combinations sum to $-2x_r$.
- The $f(x)$ represents all reactions that *produce* x_r . This observation is critical in the constructive proof of our main theorem.

We now give a formal definition of LPP-computable numbers.

2.3 Large-Population Protocol Computable Numbers: Revisited

We extend the definition from [6]. Note that the definition also applies to large-population *unimolecular* protocols (LPUPs).

► **Definition 9.** A real number ν is said to be computable by an LPP (PLPP, LPUP) if there exists an LPP (PLPP, LPUP) such that $\mathbf{x}(t) = (x_1(t), x_2(t), \dots, x_n(t)) \in [0, 1]^n$ and

$$\lim_{t \rightarrow \infty} \sum_{i \in M} x_i(t) = \nu,$$

where $M \subseteq \{1, \dots, n\}$ represents the subset of states marked in \mathbf{x} . Moreover, all the states x_i must be initialized to some positive rational $r_i \in \mathbb{Q} \cap [0, 1]$, in the sense that $\lim_{N \rightarrow \infty} x_i^{(N)}(0) = r_i$, when $x_i^{(N)}(0)$ is the initial fraction of state i at the stage when the population is N .

A major change is our removal of the finitary requirement on equilibria. A note originally in [6] discusses the rationale for the requirement of finitely many equilibria. This assumption is needed to “avoid pathological cases, in particular the case of idle systems $q \ q' \rightarrow q \ q'$ for all q and q' . Indeed, in idle systems, all initial states are equilibria, and such a system would compute any real of $[0, 1]$, depending on the initial configuration.”

This pathological behavior could occur, if one is allowed to initialize the system with any real numbers. However, we can prevent this case by restricting the initial configuration to rational numbers. Also note that by forcing the initial counts of a species X to be a fraction of the total population N , we exclude those systems (e.g., [12, 1]) that crucially depend on small counts in a very large population. These systems are not modeled correctly by ODEs. For instance, one can not initialize X to have 100 molecules. Any state initialized to a constant amount will be a fraction that approaches zero, as N grows to infinity. In our definition, such X will be initialized to have *zero* molecules for a fixed N .

► **Remark.** It is worth noting the above setup satisfies the assumptions of Kurtz’s Theorem [26]. Therefore, the long-term behavior of an LPP is governed by an ODE, which is induced by its *balance equation*. As a result, in this paper, we often blur the boundary between the continuous semantics (by ODE) and stochastic semantics when discussing limiting behavior. See Section 4 for initial discussion on further investigation on the two semantics.

For the rest of the paper, whenever we mention LPP computable number, we mean the extended notion defined above, unless specified otherwise.

2.4 Basic Results

2.4.1 Large-Population UPPs Compute Only Rationals

As a toy case study, we investigate the class of numbers computable by large-population unimolecular protocols (LPUP). We find that unimolecular protocols have limited power.

► **Lemma 10.** A number is LPUP-computable if and only if it is rational.

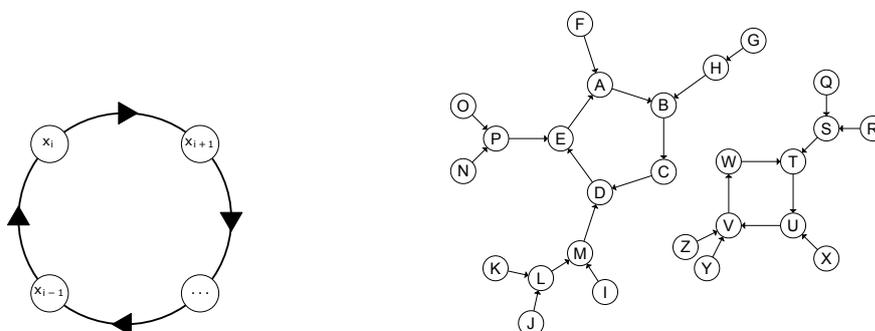
Proof. Suppose $\nu = \frac{p}{q} \in [0, 1]$ is a rational number. Lemma 3 of Bournez et al. [6] constructs an LPP with q states such that each state converges $\frac{1}{q}$ for an exponentially stable equilibrium. We modify their construction to obtain a unimolecular LPP. Let $X = \{X_1, \dots, X_q\}$ be the state set and the transitions be of the form $X_i \rightarrow X_{(i \bmod q)+1}$ (see Figure 2 for a

7:10 Computing Real Numbers with LPPs Having a Continuum of Equilibria

visualization). Then each state will converge to $\frac{1}{q}$. To compute ν , mark states X_1, \dots, X_p . Thus, there exists a deterministic unimolecular LPP that computes ν . (Alternatively, one could initialize an idle system with the desired rationals, since now the extended notion of LPP-computable number allows a continuum of equilibria and rational initial values.)

Suppose a deterministic unimolecular protocol \mathcal{P} computes some number $\nu \in [0, 1]$. Then there is a graph G corresponding to the reactions of \mathcal{P} such that nodes are states and directed edges are transitions. The deterministic nature ensures that self-loops exist only if a node is isolated from all other nodes. Since each node has the outdegree 1, G is a functional graph. Functional graphs are a set composed of rooted trees attached to a cycle ([14], page 129; or see Figure 3 for an example). Traversing the graph from any starting point will eventually lead to being stuck in a cycle. The proportion of mass that flows into a cycle will become uniformly distributed among the cycle's finite number of nodes, computing a rational number. Thus, deterministic unimolecular protocols compute rational numbers. ◀

Note that the above lemma can be extended to probabilistic unimolecular protocols, but we omit this work for the sake of simplicity.



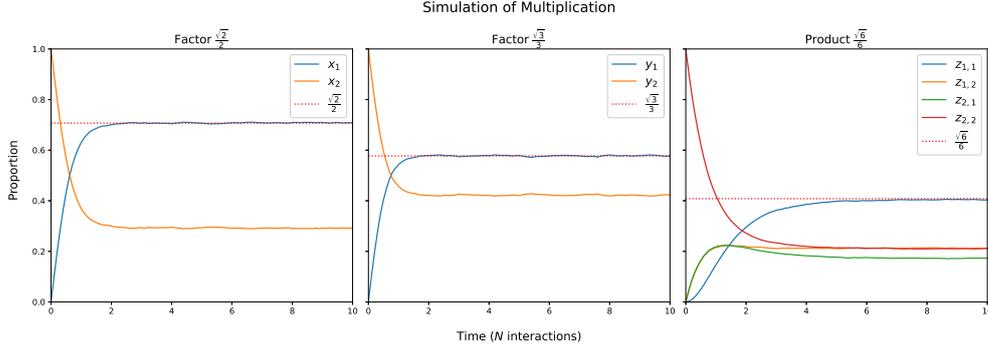
■ **Figure 2** Visualization of transitions in a unimolecular protocol. ■ **Figure 3** Example of a Functional Graph.

2.4.2 Product Protocols and Multiplication

We will show that LPP-computable numbers are closed under multiplication. The proof is done through product protocols. Since our Main Theorem shows that LPP-computable numbers are essentially GPAC-computable, many other arithmetic operations can be done by taking a detour via GPACs. However, the resulting protocols are not so intuitive. In addition, the product protocol is an important technique for our later construction in Section 3.

► **Lemma 11** (Product of LPP-computable Numbers is LPP-computable). *Let P_x and P_y be LPPs that compute α and β respectively. Denote their respective state vectors by $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and $\mathbf{y} = (y_1, y_2, \dots, y_m)$, initial configurations by $\mathbf{x}(0)$ and $\mathbf{y}(0)$, and differential systems by \mathbf{x}' and \mathbf{y}' . Assume any differential equation in \mathbf{x}' or \mathbf{y}' is a quadratic form. Then there exists an LPP, P_z , that computes the product $\alpha\beta$.*

Proof. Let $p_{x_i}, q_{x_i}, p_{y_j}, q_{y_j} \in \mathbb{P}^+$. Then we can write $x'_i = p_{x_i} - x_i q_{x_i}$ for all $x'_i \in \mathbf{x}'$. Since x'_i is a quadratic form, p_{x_i} is a quadratic form and q_{x_i} is a linear form. This notation generalizes to $y'_j \in \mathbf{y}'$ as well. Introduce new state variables $z_{i,j} := x_i y_j$ for all $(x_i, y_j) \in \mathbf{x} \times \mathbf{y}$ where \times is the Cartesian product (view \mathbf{x} and \mathbf{y} as sets). Then we have the state vector $\mathbf{z} = (z_{1,1}, z_{1,2}, z_{2,1}, \dots, z_{n,m})$. Suppose $M_{\mathbf{x}} \subset \mathbf{x}$ and $M_{\mathbf{y}} \subset \mathbf{y}$ are the marked states. For each $z_{i,j} = x_i y_j \in \mathbf{z}$, mark $z_{i,j}$ if $x_i \in M_{\mathbf{x}}$ and $y_j \in M_{\mathbf{y}}$. For the initial values, let $\mathbf{z}(0) = \mathbf{x}(0) \cdot \mathbf{y}(0)$.



■ **Figure 4** With ppsim, we can simulate a product protocol with input LPPs \mathbf{x} and \mathbf{y} that compute $\frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{3}}$. Note that the factor LPPs originate from [5], and the product LPP is a result of applying Lemma 11. These simulations affirm the closure of LPPs under multiplication.

To obtain the differential system \mathbf{z}' , perform the following steps. First, differentiate $z_{i,j} = (x_i y_j)$. By the chain rule, we have

$$z'_{i,j} = (x_i y_j)' = x'_i y_j + x_i y'_j.$$

Next, substitute the equations for x'_i and y'_j . Then, we have the cubic form,

$$z'_{i,j} = y_j (p_{x_i} - x_i q_{x_i}) + x_i (p_{y_j} - y_j q_{y_j}).$$

Apply the one-trick (Proposition 7) to obtain a homogeneously degree four polynomial such that every term is exactly degree two in variable x and exactly degree two in variable y . We have,

$$z'_{i,j} = y_j \left(\sum_{k=1}^m y_k \right) (p_{x_i} - x_i q_{x_i}) + x_i \left(\sum_{k=1}^n x_k \right) (p_{y_j} - y_j q_{y_j}).$$

After expanding the polynomial, group the positive and negative terms. Note that we can factor $x_i y_j$ from the collection of negative terms, which yields

$$z'_{i,j} = \left(\sum_{k=1}^m y_k y_j p_{x_i} + \sum_{k=1}^n x_k x_i p_{y_j} \right) - (x_i y_j) \left(q_{x_i} \sum_{k=1}^m y_k + q_{y_j} \sum_{k=1}^n x_k \right).$$

By substituting $z_{i,j}$ for the factored $x_i y_j$, we have

$$z'_{i,j} = \left(\sum_{k=1}^m y_k y_j p_{x_i} + \sum_{k=1}^n x_k x_i p_{y_j} \right) - z_{i,j} \left(q_{x_i} \sum_{k=1}^m y_k + q_{y_j} \sum_{k=1}^n x_k \right),$$

which satisfies condition (i) from Corollary 3.

Let $p_{z_{i,j}} = \left(\sum_{k=1}^m y_k y_j p_{x_i} + \sum_{k=1}^n x_k x_i p_{y_j} \right)$. Without loss of generality, consider $p_{z_{i,j}}^* = \sum_{k=1}^m y_k y_j p_{x_i}$. Note that every monomial in $p_{z_{i,j}}^*$ is of the form $x_u x_v y_j y_k$ where $x_u x_v$ is a monomial in p_{x_i} . Then we can substitute either $z_{u,j} z_{v,k}$ or $z_{u,k} z_{v,j}$ for each $x_u x_v y_j y_k$ in $p_{z_{i,j}}^*$. Since x_i^2 is not a valid monomial of p_{x_i} , we will not have $z_{i,j}^2$ in $p_{z_{i,j}}^*$. Thus, $p_{z_{i,j}}$ is a quadratic form in variable z that satisfies condition (ii) of Corollary 3.

Let $q_{z_{i,j}} = \left(q_{x_i} \sum_{k=1}^m y_k + q_{y_j} \sum_{k=1}^n x_k \right)$. Without loss of generality, consider $q_{z_{i,j}}^* = q_{x_i} \sum_{k=1}^m y_k$. Note that every monomial in $q_{z_{i,j}}^*$ is of the form $x_u y_k$, where x_u is a monomial in q_{x_i} . Then we can substitute $z_{u,k}$ for each $x_u y_k$ in $q_{z_{i,j}}^*$. Thus, we can rewrite $z'_{i,j}$ as

$$z'_{i,j} = p_{z_{i,j}} - z_{i,j} q_{z_{i,j}},$$

where $z'_{i,j}$ is a quadratic form and meets all conditions of Corollary 3.

Hence, we have defined the state set, initial configuration, and differential system of a PP-implementable protocol that computes $\alpha\beta$. ◀

3 Translating Bounded GPACs into PPs

3.1 Construction Overview

We will prove our main theorem in this section. It suffices to show that LPPs can simulate GPACs. Since one can transform a bounded GPAC into a bounded CRN with zeroed initial values [23], it is sufficient to consider only this type of CRN. We develop a four-stage algorithm that translates CRNs into LPPs.

- Input: A CRN that computes a target number α . We now enter a cascade of transformations:
- Stage 1: into a CRN-implementable quadratic system,
- Stage 2: into a TPP-implementable cubic form system,
- Stage 3: into a PP-implementable quadratic form system, and
- Stage 4: into a PLPP.

3.2 Basic Operations

First, we discuss some basic operations as building blocks for our constructive proof. The notations used are as described in the preliminary section. The naming of operations “ ε -trick” and “ λ -trick” (the use of the Greek letters ε and λ) originates in the proof of Lemma 5 in [6].

► **Operation 1** (Constant Dilation, “ ε -trick”). *Let $\mathbf{x}(t)$ be the solution to the ODE $\mathbf{x}'(t) = p(\mathbf{x}(t))$ and $\varepsilon \in \mathbb{Q}^+$ a positive constant, then $\mathbf{x}(\varepsilon t)$ is the solution to the ODE*

$$\mathbf{x}'(t) = \varepsilon p(\mathbf{x}(t)).$$

The operation is essentially a change of variable $t \rightarrow \varepsilon t$ and a direct result of the chain rule. The behavior of the ODE system will not change; it only alters the flow of time. Indeed, t in the new system corresponds to εt of the original system. In our construction, we will use this operation to shrink the coefficient of an ODE system. The operation can be seen in [11, 6].

We can do a more general change of variable (composition.)

► **Operation 2** (Time Dilation ([28], Theorem 3)). *Let $\mathbf{x}(t)$ be the solution to the GPAC $\mathbf{x}(t)' = p(\mathbf{x}(t))$, $g(t)$ be a GPAC-implementable function, and $G(t) = \int_0^t g(s)ds$, then $\mathbf{x}(G(t))$ is the solution to the GPAC*

$$\mathbf{x}'(t) = p(\mathbf{x}(t)) \cdot g.$$

In our construction, we must choose $g(t)$ such that its integral satisfies

$$G(t)|_{t=\infty} = \int_0^\infty g(s)ds = \infty.$$

and we call such a $G(t)$ as “nonterminating time”.

A time dilating operation, as above, controls the flow of time with the function $g(t)$. More accurately, t in the new system corresponds to $G(t) = \int_0^t g(s)ds$ of the original system. If we want the new system to mimic the old system’s limiting behavior (i.e., to achieve the value $\lim_{t \rightarrow \infty} x_1(t)$), we must guarantee $G(\infty) = \infty$. Thus, time in the new system is nonterminating and can go to infinity in the eyes of the old system. Otherwise, the new system would terminate somewhere in the middle, relative to the old system. To

obtain nonterminating time, it is enough to consider functions that converge slowly, e.g., $g(t) = \frac{1}{1+t}$. Alternatively, one could ensure g is bounded from below, i.e., $g(t) > c > 0$, for some constant c .

Time dilation for CRNs can be found in [25] (page 15, Theorem 3.7). \lrcorner

Next, we introduce an operation that shrinks variables to $[0,1]$.

► **Operation 3** (Variable Shrinking, or “ λ -trick”). *Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be a bounded CRN that computes a number α in $[0, 1]$. That is, $\lim_{t \rightarrow \infty} x_1(t) = \alpha$. To transform the CRN into a LPP, we must ensure all other components, some of which may not converge, fall in $[0,1]$ for all t . We can pick a $0 < \lambda < 1$ sufficiently small and a constant $c > 0$ such that*

$$x_1 + \lambda(x_2 + \dots + x_n) < 1 - c.$$

This is possible since all variables are bounded. Then we

- *make a change of variables: $x_2 \leftarrow \lambda x_2, \dots, x_n \leftarrow \lambda x_n$, and*
- *introduce a new variable $x_0(t)$ such that $x_0(t) + x_1(t) + \dots + x_n(t) = 1$ by*
 1. *initializing $x_0(0) = 1$ (recall $x_i(0) = 0$ for $i = 1, \dots, n$), and*
 2. *adding the ODE for x_0 : $x'_0 = -\sum_{i=1}^n x'_i$.*

Note that x_1 remains unchanged in the operation, so it still computes α . \lrcorner

The biggest challenge in translating a CRN into an LPP is: the CRN does not necessarily preserve population, while the LPP requires it. In fact, our input CRN initializes at zero. However, x_1 needs to compute some number α as all other components remain positive throughout time; evidently, not preserving the population. A straightforward idea to “balance” the system, is to introduce a variable x_0 , such that its derivative cancels all changes to the population size by other variables. That is, $x'_0 = -\sum_{i=1}^n x'_i$.

However, the resulting x_0 is not CRN-implementable in general. As a result, it is not transformable to an LPP. The rationale is clear in the derivative of x_i . In general, x'_i has the form

$$x'_i = p_i - q_i \cdot x_i, \quad \text{where } p, q \in \mathbb{P}^+.$$

A typical p_i would contain at least one monomial m . Evidently, $x_0 \notin m$ since the variable x_0 is newly introduced. Then m will contribute to a negative term in x'_0 and $x_0 \notin m$, making x_0 not CRN-implementable and, in turn, not PP-implementable. That is why Bournez et al.’s construction in [6] (Lemma 2) fails to produce a population protocol in general. To address this issue, we purposefully *multiply* x_0 with every x'_i and introduce the following operation.

► **Operation 4** (Balancing Dilation, or “Garbage Collection” (“g-trick”). *Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be a CRN-computable system, (i.e., every component x_i is CRN-computable). Then the following operation (multiplying an unknown variable x_0 determined by an ODE) produces a conservative CRN-implementable system.*

$$\begin{cases} x'_i = (p_i - q_i \cdot x_i) \cdot x_0, & \text{for } i \in \{1, \dots, n\} \\ x'_0 = -\sum_{i=1}^n x'_i. \end{cases}$$

What makes the operation different from a normal time dilation operation is, the function x_0 is not known in advance: It needs to adjust/customize according to the system it is dilating. In our construction, we also need to combine the operation with the λ -trick to achieve nonterminating time. This operation also turns a non-conservative system into a conservative CRN-implementable one.

We will present our construction in the following subsections.

3.3 Stage 1: Conversion to CRN-implementable Quadratic Systems

The following is a CRN version of [9] (Theorem 1).

► **Theorem 12.** *Any solution of a CRN is a solution of a CRN-implementable system of degree at most two.*

With this result, we can assume a CRN's associated polynomial system is quadratic with out loss of generality.

3.4 Stage 2: Conversion to TPP-implementable Cubic Form Systems

► **Theorem 13.** *Any solution of a quadratic CRN is a solution of a TPP-implementable cubic form system.*

We sketch the proof here. Please see the appendix for more detail.

Proof. Given a quadratic CRN $\mathbf{x} = (x_1, x_2, \dots, x_n)$, we first apply the λ -trick to “make room” for a new variable x_0 . That is, we choose a $0 < \lambda < 1$ and a constant $0 < c < 1$, such that $x_1 + \lambda(x_2 + \dots + x_n) < 1 - c$. Then apply the one-trick and balancing dilation to get a cubic form:

1. Introduce a new variable x_0
2. Rewrite each p_i (quadratic), q_i (linear) in $x'_i = p_i - q_i \cdot x_i$ to forms:
 - for a linear monomial in p_i of the form $a \cdot x_j$, apply the “one”-trick and rewrite it as $ax_j \sum_{k=0}^n x_k$.
 - for a constant term in p_i of the form a , rewrite it as $a \cdot (\sum_{k=0}^n x_k) \cdot (\sum_{k=0}^n x_k)$.
 - for a constant term a in q_i , rewrite it as $a \cdot \sum_{k=0}^n x_k$.
 We call the resulting forms P_i and Q_i respectively.
3. Apply balancing dilation with x_0 . The new system looks like

$$\begin{cases} x'_i = (P_i - Q_i x_i) \cdot x_0, & \text{for } i = 1, \dots, n \\ x'_0 = -\sum_{i=1}^n x'_i. \end{cases} \quad \text{and} \quad \begin{cases} x_i(0) = 0, & \text{for } i = 1, \dots, n \\ x_0(0) = 1. \end{cases}$$

It is a TPP-implementable cubic form system. See the appendix for a full proof. ◀

► **Remark 14.** Note that in the above proof x_0 is bounded from below, away from zero. We can observe that:

1. The variable x_0 starts from 1 and remains greater than $c > 0$ for all t . It will not introduce new equilibria.
2. The new system has at most a linear slowdown compared to the old system, since $\int_0^t x(0) dt > \int_0^t c dt = ct$.

The variable x_0 acts like a source, distributing the total mass (sum = 1) to all other variables (initialized at 0).

3.5 Stage 3: Conversion to PP-implementable Quadratic Form Systems

► **Theorem 15.** *A number computable by a TPP-implementable function given by Stage 2 can be computed by the sum of a finite set of PP-implementable functions.*

Proof. Sketch of the proof. We utilize the self-product of the TPP-implementable cubic form system. Let $\mathbf{x} = (x_0, x_1, \dots, x_n)$ be the system resulting from the previous stage that computes a real number α with initial values $x_0 = 1$ and 0 otherwise. We consider the system $(z_{i,j})_{(i,j) \in [n+1]^2}$ where $z_{i,j} = x_i \cdot x_j$ and $[n+1] = \{0, 1, \dots, n\}$. Let $z_{0,0}(0) = 1$ and

0 otherwise. The marked set $M_z = \{z_{1,j} | 0 \leq j \leq n\}$. Then we can verify the z -system is PP-implementable and the sum of the marked variables traces α . Check the appendix for a full proof. ◀

3.6 Stage 4: Converting PP-implementable Quadratic Form Systems to PLPPs

In this subsection, we turn the PP-implementable quadratic form given as a result of the previous stage into a PLPP. The following construction is from the proof of Lemma 5 in [6].

► **Construction 1.** Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be a PP-implementable quadratic form. We denote the derivative x_i as

$$x'_i = p_i - q_i x_i, \quad i = 1, 2, \dots, n, \quad (8)$$

where p_i is a quadratic form and q_i is a linear form.

We will apply the ε -trick (Operation 1) to shrink the coefficient of the above system, where $\varepsilon \in \mathbb{Q}$ is an undetermined parameter. In order to align with the form in Equation (7), we add and subtract a term $2x_i$. Now the system has the form

$$\begin{aligned} x'_i &= \varepsilon(p_i - q_i x_i) + 2x_i - 2x_i, \quad i = 1, 2, \dots, n. \\ &= f_i(\mathbf{x}), \quad \text{denote } f_i(\mathbf{x}) = \varepsilon(p_i - q_i x_i) + 2x_i. \end{aligned} \quad (9)$$

We then use the one-trick to rewrite $f_i(\mathbf{x}) = \varepsilon(p_i - q_i x_i) + 2x_i \sum_{j=1}^n x_j$. Note that the term brings in $2x_i x_j$ for $j = 1, \dots, n$. We will then pick ε sufficiently small

1. to ensure the $2x_i x_j$ terms dominate the monomials in $-q_i x_i$ such that $f_i(\mathbf{x}) \in \mathbb{P}^+$ for all i .
2. and to ensure

$$\frac{\sum_{i,j \neq k} C(x_j x_k, f_i)}{4} \leq 1 \quad \text{and} \quad \frac{\sum_{i,j} C(x_j^2, f_i)}{2} \leq 1 \quad (11)$$

where $C(x_j x_k, f_i)$ denotes the coefficient of $x_j x_k$ in f_i , and $i, j, k \in \{1, \dots, n\}$.

Next, we construct a rule set for a PLPP. For an ordered pair (x_j, x_k) and a term $\alpha_{i,j,k} x_j x_k$ in $f_i(\mathbf{x})$ we do the following:

- If $j = k$, add a rule $(x_j, x_j) \rightarrow \frac{\alpha_{i,j,k}}{2}(x_i, x_i)$;
- otherwise, add two rules $(x_j, x_k) \rightarrow \frac{\alpha_{i,j,k}}{4}(x_i, x_i)$ and $(x_k, x_j) \rightarrow \frac{\alpha_{i,j,k}}{4}(x_i, x_i)$;
- In the above rules, if $s = \sum_{i=1}^n \alpha_{i,j,k} < 1$ for an order pair (k, l) , we add an idle reaction

$$(x_j, x_k) \rightarrow (1 - s)(x_j, x_k);$$

If a term $x_j x_k$ appears in $f_i(\mathbf{x})$, it implies that the pair (x_j, x_k) will produce x_i . We use the all-in greedy strategy, that is, we produce two x_i 's and let the pair (x_k, x_j) do the same. So we need to divide the coefficient $\alpha_{i,j,k}$ by 4 for $j \neq k$; yet, when $k = j$ we don't have a flipped order pair, so we only divide the coefficient by 2.

There need to make sure that the rule set generated above forms a PLPP.

► **Theorem 16.** Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be a quadratic system satisfying $\sum_{i=1}^n x_i = 1$. Then it can be turn into a PLPP by Construction 1 if and only if it is PP-implementable.

Proof. See appendix. ◀

The following theorem can help in transforming a PLPP to an LPP.

► **Theorem 17** ([6], Lemma 4). *Let $\nu \in [0, 1]$, and assume that there exists a PLPP computing ν , with rational probabilities. Then there exists a (deterministic) LPP computing ν .*

Note that although the above theorem is based on the original notion of LPP-computable number with the finite-equilibria constraint, the construction in the proof of the theorem still correctly encodes the balance equation. So the theorem also holds under the new definition.

We therefore finish the proof of our main theorem.

► **Main Theorem.** *LPPs compute the same set of numbers in $[0, 1]$ as GPACs and CRNs.*

► **Corollary 18.** *Algebraic numbers are LPP computable.*

Proof. Algebraic numbers are computable by a GPAC [6] (Lemma 5) so they are also LPP-computable by the main theorem. ◀

The above corollary fix the gap in Bournez et al.'s construction.

► **Corollary 19.** *Famous math constants $\frac{\pi}{4}$, e^{-1} , Euler's γ , Dottie number, and Catalan's constant are LPP-computable.*

This result answers Bournez et al.'s question about whether solutions of trigonometric equations (Dottie number) and $\frac{\pi}{4}$ are “computable” asymptotically by population protocols.

4 Conclusion and Discussion

In this paper, we extend the notion of LLP-computable numbers and connect it with GPAC/CRN-computable numbers. Moreover, we show that LLPs and GPACs/CRNs essentially compute the same set of real numbers. The result, to some degree, says LPPs, or more straightforwardly, population protocols with the mass-action semantic, can simulate GPACs in some way. We would ask a natural question: What is the next “weak” or minimal (according to some measure) model that can simulate GPACs? Unimolecular protocols are provably incapable since they only compute rational numbers. However, a model discussed in [16], which can be viewed as a two-state (“black” or “white”) k -PP with restrictions on reactions (the product of a reaction must either be all black or all white), can compute algebraic numbers such as $\frac{3-\sqrt{5}}{2}$ but not rational numbers like $\frac{1}{5}$. Diving into the sub-models of population protocols and characterizing the power spectrum will be an interesting avenue for further exploration.

The ODE systems we used to compute transcendental numbers have a continuum of equilibria. We have not addressed the semistability nor the system's convergence rate in this paper. Regarding the latter, since our construction essentially rewrites the system with a new set of auxiliary variables, the system's solutions in some sense remain the same, except for the *linear* slowdown introduced in Stage 2. It is hoped that an LPP's convergence rate is similar to the input CRN. If an input CRN converges exponentially fast to a number α , we desire the translated LPP also converges (exponentially) fast. We would need a thorough stochastic analysis along this line in future work.

The definition of GPAC/CRN-computable numbers and LPP-computable numbers have a major difference: We can trace a real number by a *set* of variables in an LPP, rather than one, as in a GPAC/CRN. Currently, the proof of our main theorem relies on using a set of marked variables. Our question: Is the difference intrinsic? Or can we compute the same set of numbers using *one* variable?

Another area of exploration is protocols with a large population (tending to infinity) that also have “scarce” variables with a limited population. The behaviors of these systems, due to the existence of the scarce variables, can not be governed by Kurtz’s theorem. Therefore, a new analytic tool or theoretical benchmark (probably not a computable number) awaits development.

Our construction in Section 3 is not optimal. To implement the translation algorithm in a software package, one would want to have a better algorithm for Stage 1. To apply the λ -trick, one needs to know the bounds of the variables, which would require numerical methods. A theory to predict the bounds is desired but difficult to achieve due to the non-linear nature of ODEs.

References

- 1 Dan Alistarh, Bartłomiej Dudek, Adrian Kosowski, David Soloveichik, and Przemysław Uznański. Robust detection in leak-prone population protocols. In *International Conference on DNA-Based Computers*, pages 155–171. Springer, 2017.
- 2 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.
- 3 Guillaume Aupy and Olivier Bournez. On the number of binary-minded individuals required to compute. *Theoretical Computer Science*, 412(22):2262, 2011.
- 4 Sanjay P. Bhat and Dennis S. Bernstein. Nontangency-based lyapunov tests for convergence and stability in systems having a continuum of equilibria. *SIAM Journal on Control and Optimization*, 42(5):1745–1775, 2003.
- 5 Olivier Bournez, Philippe Chassaing, Johanne Cohen, Lucas Gerin, and Xavier Koegler. On the convergence of population protocols when population goes to infinity. *Applied Mathematics and Computation*, 215(4):1340–1350, 2009.
- 6 Olivier Bournez, Pierre Fraigniaud, and Xavier Koegler. Computing with large populations using interactions. In *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science*, pages 234–246. Springer, 2012.
- 7 Olivier Bournez, Daniel S. Graça, and Amaury Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length. *Journal of the ACM*, 64(6):38, 2017.
- 8 Vannevar Bush. The differential analyzer. A new machine for solving differential equations. *Journal of the Franklin Institute*, 212(4):447–488, 1931.
- 9 David C. Carothers, G. Edgar Parker, James S. Sochacki, and Paul G. Warne. Some properties of solutions to polynomial systems of differential equations. *Electronic Journal of Differential Equations (EJDE)[electronic only]*, 2005:Paper–No, 2005.
- 10 David Cox, John Little, and Donal OShea. *Ideals, Varieties, and Algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer Science & Business Media, 2013.
- 11 David Doty and Eric Severson. Ppsim: A software package for efficiently simulating and visualizing population protocols. In *International Conference on Computational Methods in Systems Biology*, pages 245–253. Springer, 2021.
- 12 Bartłomiej Dudek and Adrian Kosowski. Universal protocols for information dissemination using emergent signals. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 87–99, 2018.
- 13 François Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Proceedings of the 15th International Conference on Computational Methods in Systems Biology*, pages 108–127. Springer International Publishing, 2017.
- 14 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

- 15 Willem Fletcher, Titus H. Klinge, James I. Lathrop, Dawn A. Nye, and Matthew Rayman. Robust real-time computing with chemical reaction networks. In *International Conference on Unconventional Computation and Natural Computation*, pages 35–50. Springer, 2021.
- 16 Lucas Gerin and Marie Albenque. On the algebraic numbers computable by some generalized ehrenfest urns. *Discrete Mathematics & Theoretical Computer Science*, 14, 2012.
- 17 Daniel S. Graça. Some recent developments on shannon’s general purpose analog computer. *Mathematical Logic Quarterly: Mathematical Logic Quarterly*, 50(4-5):473–485, 2004.
- 18 Daniel Silva Graça and José Félix Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5):644–664, 2003.
- 19 Daniel S. Graça, Manuel L. Campagnolo, and Jorge Buescu. Computability with polynomial differential equations. *Advances in Applied Mathematics*, 40(3):330–349, 2008.
- 20 Indranil Gupta, Mahvesh Nagda, and Christo Frank Devaraj. The design of novel distributed protocols from differential equations. *Distributed Computing*, 20(2):95–114, 2007.
- 21 Vera Hárs and János Tóth. On the inverse problem of reaction kinetics. In *Colloquia Mathematica Societatis János Bolyai, 30: Qualitative Theory of Differential Equations*, pages 363–379, 1981.
- 22 Xiang Huang. *Chemical Reaction Networks: Computability, Complexity, and Randomness*. PhD thesis, Iowa State University, 2020.
- 23 Xiang Huang, Titus H. Klinge, and James I. Lathrop. Real-time equivalence of chemical reaction networks and analog computers. In *International Conference on DNA Computing and Molecular Programming*, pages 37–53. Springer, 2019.
- 24 Xiang Huang, Titus H. Klinge, James I. Lathrop, Xiaoyuan Li, and Jack H. Lutz. Real-time computability of real numbers by chemical reaction networks. *Natural Computing*, 18(1):63–73, 2019.
- 25 Titus H. Klinge. *Modular and Robust Computation with Deterministic Chemical Reaction Networks*. PhD thesis, Iowa State University, 2016.
- 26 Thomas G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics*, 57(7):2976–2978, 1972.
- 27 Leonard Lipshitz and Lee A Rubel. A differentially algebraic replacement theorem, and analog computability. *Proceedings of the American Mathematical Society*, 99(2):367–372, 1987.
- 28 G. Edgar Parker and James S. Sochacki. Implementing the picard iteration. *Neural, Parallel & Scientific Computations*, 4(1):97–112, 1996.
- 29 Marion B Pour-El and Jonathan I Richards. Abstract computability and its relations to the general purpose analog computer. *Transactions of the American Mathematical Society*, 199:1–28, 1974.
- 30 Claude E Shannon. Mathematical theory of the differential analyzer. *Studies in Applied Mathematics*, 20(1-4):337–354, 1941.
- 31 Justin R. Smith. *Introduction to Algebraic Geometry*. Justin Smith, 2014.
- 32 Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936.

A Optional Technical Appendix

Proof of Observation 8. Take the x_r component in Equation (6). We have

$$\begin{aligned}
 \frac{dx_r}{dt} &= \sum_{j \in [n]} \left(-x_r x_j \right) + \sum_{j \in [n]} \left(-x_r x_j \right) \\
 &\quad + Proj_r \left(\sum_{i, j \in [n]} \left(x_i x_j \sum_{k, l \in [n]} \alpha_{i, j, k, l} (e_k + e_l) \right) \right), \\
 &= -2x_r + f(\mathbf{x}), \quad \text{by Observation 7.}
 \end{aligned}$$

where $f(\mathbf{x}) = Proj_r \left(\sum_{i,j \in [n]} \left(x_i x_j \sum_{k,l \in [n]} \alpha_{i,j,k,l} (e_k + e_l) \right) \right)$ and $Proj_r(\cdot)$ is the r -th component a vector. This completes the proof. \blacktriangleleft

Proof of Theorem 12. Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be a CRN, then each x_i has the form $x'_i = p_i - q_i x$, where $p_i, q_i \in \mathbb{P}^+$. We in introduce variables (we call them v -variables) for each monomial as follows:

$$v_{i_1, \dots, i_n} = x_1^{i_1} x_2^{i_2} \cdot x_n^{i_n}.$$

Specially, $v_{1,0,\dots,0} = x_1$, $v_{0,1,0,\dots,0} = x_2$, and so on. Note that we can rewrite p_i and q_i in each x'_i with the v -variables and denote them as P_i and Q_i . We can see that P_i and Q_i are of degree one in terms of the v -variables.

Consider the system that contains the collection of all v_{i_1, \dots, i_n} 's. Notice that

$$\begin{aligned} v'_{i_1, \dots, i_k, \dots, i_n} &= \sum_{k=1}^n x_1^{i_1} \dots x_k^{i_k-1} \dots x_n^{i_n} \cdot x'_k \\ &= \sum_{k=1}^n v_{i_1, \dots, i_k-1, \dots, i_n} (P_k - Q_k x_k) \\ &= \sum_{k=1}^n P_k v_{i_1, \dots, i_k-1, \dots, i_n} - \sum_{k=1}^n Q_k (x_k \cdot v_{i_1, \dots, i_k-1, \dots, i_n}) \\ &= \sum_{k=1}^n P_k v_{i_1, \dots, i_k-1, \dots, i_n} - \left(\sum_{k=1}^n Q_k \right) \cdot v_{i_1, \dots, i_k, \dots, i_n}. \end{aligned}$$

Note that the v -system is a CRN-implementable system with degree less than two. \blacktriangleleft

Proof of Theorem 13. Let $\mathbf{x} = (x_1, \dots, x_n)$ be a quadratic CRN-implementable system. Then we have $x'_i = p_i - x_i q_i$ such that $p_i, q_i \in \mathbb{P}^+$ for $i = 1, \dots, n$. Since the CRN is quadratic, it follows that $\deg(p_i) \leq 2$ and $\deg(q_i) \leq 1$ for $i = 1, \dots, n$.

First, apply the λ -trick to shrink variables x_2, \dots, x_n . Choose $0 \leq \lambda \leq 1$ and a constant $0 < c < 1$, such that $x_1 + \lambda(x_2 + \dots + x_n) < 1 - c$. Then perform the change of variables: $x_2 \leftarrow \lambda x_2, \dots, x_n \leftarrow \lambda x_n$.

Next, we introduce a new variable x_0 such that $\sum_{i=0}^n x_i = 1$. Use this summation and the "one-trick" to ensure that every term in x'_i is quadratic for $i = 1, \dots, n$. Once each x'_1, \dots, x'_n is a quadratic form, we signify these changes by writing $x'_i = P_i - x_i Q_i$ where $P_i, Q_i \in \mathbb{P}^+$. Note that P_i is a quadratic form and Q_i is a linear form.

We can now perform time dilation with x_0 . As a result, we have the cubic form system,

$$\begin{cases} x'_i = (P_i - Q_i x_i) \cdot x_0, & \text{for } i = 1, \dots, n \\ x'_0 = -\sum_{i=1}^n x'_i \end{cases}$$

where we let $x_0(0) = 1$ and $x_i(0) = 0$ for $i = 1, \dots, n$. Note that $P_i x_0, Q_i x_0 \in \mathbb{P}^+$ and x_i is in every negative term of x'_i for $i = 1, \dots, n$. Some algebraic manipulation reveals that

$$x'_0 = -\sum_{i=1}^n x'_i = -\sum_{i=1}^n (P_i - Q_i x_i) \cdot x_0 = \sum_{i=1}^n (Q_i x_i x_0 - P_i x_0).$$

Clearly, $Q_i x_0, P_i x_0 \in \mathbb{P}^+$ and x_0 is in every negative term of x'_0 . Thus, \mathbf{x}' is CRN implementable.

7:20 Computing Real Numbers with LPPs Having a Continuum of Equilibria

Since x_0 is a factor of every term in the cubic form \mathbf{x}' , x_i^3 is not a positive term of x'_i for $i = 1, \dots, n$. For the case of x'_0 , recall that the positive terms of x'_0 come from the negative terms of x'_1, \dots, x'_n . Given that every negative term of x'_i will have x_i as a factor, every positive term of x'_0 will have at least one $x_i \neq x_0$ as a factor. Thus, x_0^3 cannot occur as a positive term in x'_0 .

Hence, we have satisfied Corollary 4 to obtain a TPP-implementable cubic form system. \blacktriangleleft

Proof of Theorem 15. Let $\mathbf{x} = (x_0, x_1, \dots, x_n)$ be the TPP-implementable cubic form system from Stage 2. Then we have

$$\mathbf{x}' = \begin{cases} x'_i = (P_i - Q_i x_i) \cdot x_0, & \text{for } i = 1, \dots, n \\ x'_0 = -\sum_{i=1}^n x'_i \end{cases}$$

where x_1 computes some α and the system is initialized such that $x_0(0) = 1$ and 0 otherwise.

We construct our proof based on the “one trick” given by Observation 7,

$$\sum_{i=0}^n x_i \cdot \sum_{j=0}^n x_j = 1 \cdot 1 = 1.$$

We begin by defining a new variable vector, \mathbf{z} . Introduce new variables $z_{i,j} = x_i x_j$ for all $(x_i, x_j) \in \mathbf{x} \times \mathbf{x}$. To compute α , define the marked states as $M_z = \{z_{1,j} \mid 0 \leq j \leq n\}$. Given $\mathbf{x}(0)$, let $z_{0,0}(0) = 1$ and 0 otherwise.

Now, we find \mathbf{z}' . For all $z_{i,j} \in \mathbf{z}$ we have $z'_{i,j} = (x_i x_j)' = x'_i x_j + x_i x'_j$, where $x'_i, x'_j \in \mathbf{x}'$. Recall that \mathbf{x}' consists of cubic forms. Therefore, $x'_i x_j + x_i x'_j$ is a quartic form in variable \mathbf{x} . It follows from a change of variables from \mathbf{x} to \mathbf{z} that $z'_{i,j}$ is a quadratic form. For each $z_{i,j} \in \mathbf{z}$, we check conditions (i) and (ii) of Theorem 3. There are three non-trivial cases.

Case 1: Assume $z_{i,j} = x_i x_j$ such that $i \neq 0$ and $j \neq 0$. Then by the chain rule and substitution of \mathbf{x}' , we have

$$z'_{i,j} = x'_i x_j + x_i x'_j = (x_0 P_i - x_i x_0 Q_i) \cdot x_j + x_i \cdot (x_0 P_j - x_j x_0 Q_j).$$

We then distribute and group terms by their sign to obtain:

$$z'_{i,j} = (x_j x_0 P_i + x_i x_0 P_j) - (x_i x_j x_0 Q_i + x_i x_j x_0 Q_j)$$

Note that we can factor $(x_i x_j)$ from every negative term,

$$z'_{i,j} = (x_j x_0 P_i + x_i x_0 P_j) - (x_i x_j)(x_0 Q_i + x_0 Q_j).$$

$$\begin{aligned} z'_{i,j} &= (x_j x_0 P_i + x_i x_0 P_j) - (x_i x_j)(x_0 Q_i + x_0 Q_j) \\ &= (z_{0,j} P_i + z_{0,i} P_j) - z_{i,j} \cdot (x_0 Q_i + x_0 Q_j) \end{aligned}$$

and the terms P_i , P_j , $x_0 Q_i$, and $x_0 Q_j$ can be further written into linear form of $\{z_{i,j}\}_{i,j}$'s. We will not repeat these argument in the rest of the proof, though.

The above implies $z_{i,j}$ is a factor of every negative term. Also, since x_0 can be factored from every positive term, we cannot have a positive $z_{i,j}^2$ (recall that i, j are nonzero). Thus, conditions (i) and (ii) are satisfied.

Case 2: Assume $z_{0,j}$ such that $j \neq 0$. From the chain rule and substituting equations from \mathbf{x}' , we have

$$z'_{0,j} = x'_0 x_j + x_0 x'_j = \sum_{i=1}^n (x_i x_0 Q_i - x_0 P_i) \cdot x_j + x_0 \cdot (x_0 P_j - x_j x_0 Q_j)$$

Distribute and group terms by their sign to obtain,

$$z'_{0,j} = x_0^2 P_j + \sum_{i=1}^n x_i x_j x_0 Q_i - (x_j x_0^2 Q_j + \sum_{i=1}^n x_0 x_j P_i)$$

Note that we can factor $(x_0 x_j)$ from every negative term,

$$z'_{0,j} = x_0^2 P_j + \sum_{i=1}^n x_i x_j x_0 Q_i - (x_j x_0)(x_0 Q_j + \sum_{i=1}^n P_i)$$

which implies $z_{0,j}$ is a factor of every negative term. Suppose a positive term in $z'_{0,j}$ has the monomial $(x_0 x_j)^2$. By carefully choosing the change of variable (i.e., choose $z_{j,j} z_{0,0}$, not $z_{0,j}^2$), we satisfy condition (ii). If there is no positive term in $z'_{0,j}$ with the monomial $(x_0 x_j)^2$, then condition (ii) is met. Then, both conditions (i) and (ii) are met.

Case 3: Assume $z_{0,0}$. With the chain rule and substitution, we have

$$z'_{0,0} = 2x_0 \cdot x'_0 = 2x_0 \cdot \sum_{i=1}^n (x_0 x_i Q_i - x_0 P_i)$$

Distributing and grouping terms by their sign yields,

$$z'_{0,0} = \sum_{i=1}^n (2x_0^2 x_i Q_i) - \sum_{i=1}^n (2x_0^2 P_i)$$

Since x_0^2 can be factored from every negative term, condition (i) is satisfied. Since $i = 1, \dots, n$, it follows that $x_i \neq x_0$. Thus, condition (ii) is satisfied.

In summary, we have defined the states \mathbf{z} , marked states $M_{\mathbf{z}}$, initial configuration $\mathbf{z}(0)$, and differential system \mathbf{z}' . Furthermore, for each $z'_{i,j} \in \mathbf{z}'$ we have established: each $z'_{i,j}$ is a quadratic form, $z'_{i,j}$ can be written to avoid a positive $z_{i,j}^2$ in $z'_{i,j}$, and the existence of $z_{i,j}$ in every negative term of $z'_{i,j}$. Hence, a TPP-implementable cubic form system can be transformed into a PP-implementable system that computes α . ◀

Proof of Theorem 16. The “only if” part is clear. Therefore, it suffices to show the “if” direction. We adopt the same symbols and notations as in Construction 1. Recall that we denote $f_i(\mathbf{x}) = \varepsilon(p_i - q_i x_i) + 2x_i \sum_{j=1}^n x_j$. Consider an arbitrary ordered pair (x_j, x_k) . We need to show that in the construction:

1. The selection of ε satisfying the two conditions is always possible.
2. The rule set generated by the construction forms an PLPP. Specifically, for the ordered pair (x_j, x_k) , we must show that $\sum_{i=1}^n \alpha_{i,j,k} = 1$ and $\alpha_{i,j,k} > 0$ for all i . That is, the $\alpha_{i,j,k}$'s form a discrete probability measure with j, k fixed.

We break the proof into two cases:

- Case $j \neq k$: The term $x_j x_k$ occurs in $f_i(\mathbf{x})$ from two sources.
 - Case 1: $x_j x_k \in \varepsilon(p_i - q_i x_i)$. The coefficients of such terms for all i must sum to zero, since the system is assumed to be PP-implementable.
 - Case 2: $x_j x_k \in 2x_i(x_1 + \dots + x_n)$. We have two sources: $x_j x_k$ occurs in $f_j(\mathbf{x})$ as a term in $2x_j(x_1 + \dots + x_k + \dots + x_n)$ and similarly in $f_k(\mathbf{x})$ as a term in $2x_k(x_1 + \dots + x_j + \dots + x_n)$. The sum of these coefficients is 4. Note that the coefficients are divided by 4 in the construction, which ensures the resulting sum of $\alpha_{i,j,k}$ is one. One can verify that the selection of ε is always possible under this case.

Case $j = k$: For term x_j^2 , which occurs in $f_i(\mathbf{x})$, there are two subcases:

- $i \neq j$: Then $x_j^2 \in \varepsilon p_i$ and the coefficient must be positive.
- $i = j$: Then $x_j^2 = x_i^2 \in -\varepsilon q + 2x_i \sum_{k=0}^n x_k$ and coefficient is less than 2. ($x_j^2 = x_i^2$ can not occur in εp since the system is PP-implementable.)

Recall that when constructing the rule set, the coefficients are divided by 2. As a result, generated $\alpha_{i,j}$'s are positive and less than 1. Due to the system's conservative nature, the coefficients x_j^2 in $\varepsilon(p_i - q_i x_i)$ for i 's in $\{1, \dots, n\}$ will sum to zero. Since the $2x_j \sum_{k=1}^n x_k$ term introduces a $2x_j^2$ term, the resulting coefficient sum for x_j^2 is 2. In the construction, these coefficients are always divided by 2 to generate $\alpha_{i,j}$ in $f_i(\mathbf{x})$. Thus, the $\sum_{i=1}^n \alpha_{i,j} = 1$. One can then verify that the selection of ε is feasible.

In contrast, assume the system is not PP-implementable. Then there is an $x_i^2 \in \varepsilon p_i$ with a positive coefficient. Once combined with the term $2x_i^2 \in 2x_i \sum_{k=1}^n x_k$, the sum of coefficients will be greater than two. Then dividing by 2 will generate an $\alpha_{i,i} > 1$, which is not permitted by PLPPs. Hence, the selection of ε is not feasible in this case. ◀

Proof of Corollary 19. The first four numbers are GPAC/CRN computable (see [22]). Note that all real-time computable numbers by CRNs in [23] are computable in this paper. One just need to disregard the real-time constraint. Then by our main theorem, they are also LPP computable. It suffices to show that Catalan's constant is GPAC computable. We use the formula $G = \frac{1}{2} \int_0^\infty \frac{t}{\cosh t} dt$. We have

$$\begin{aligned} G &= \int_0^\infty \frac{t}{e^t + e^{-t}} dt, \quad \text{since } \cosh(t) = \frac{e^t + e^{-t}}{2} \\ &= \int_0^\infty (t \cdot e^{-t}) \cdot \frac{1}{1 + e^{-2t}} dt. \end{aligned}$$

We let

$$R = t \cdot e^{-t}, \quad E = e^{-t}, \quad V = \frac{e^{-2t}}{1 + e^{-2t}},$$

and

$$G(t) = \int_0^t (te^{-t}) \cdot \frac{1}{1 + e^{-2t}} dt = \int_0^t R \cdot (1 - V) dt.$$

Taking derivatives against t , we get

$$\begin{cases} G' = R(1 - V) \\ R' = E - R \\ E' = -E \\ V' = (1 - V)^2 \cdot (-2E^2), \end{cases} \quad \begin{cases} G(0) = 0 \\ R(0) = 0 \\ E(0) = 1 \\ V(0) = \frac{1}{2}. \end{cases}$$

Therefore Catalan's constant can be computed by the above PIVP. Note that the system's non-zero initial values can be transformed into a system with zeroed initial values ([23], Theorem 3). Hence by our Main Theorem, Catalan's constant is LPP computable. ◀