Parameterized System Verification: Does Process's Knowledge Affect Decidability?

Preliminary Technical Draft

December 16, 2000

Abstract

It has been well-known that any non-trivial properties of parameterized finite systems is in general undecidable. However all previous undecidability proofs reply on the fact that processes have access to global configuration knowledge (such as system scale) or there is a distinguished (control) process which can collect such type of information. A question arises when we ask if there exists uniform procedure for determining a process's local properties, given a system in which no process can gain global configuration knowledge in any computations. It is shown here that the verification problem on this more restricted class of parameterized systems is still undecidable. The undecidability is established in a specific class of completely identical parameterized finite systems using reduction to *tag systems*.

1 Introduction

It has been well-known that any non-trivial properties of parameterized finite systems is in general undecidable ([AK86], [Suz88], [GS92], [EN95]). However all previous undecidability proofs reply on the fact that processes are aware of system global configurations (such as system scale) or there is a distinguished control process which can collect such type of information. In particular, it would be not a surprise to have undecidability if processes can find the system scale. A proof sketch looks as follows:

> Let every process be a simulator of some *Turing machine*. In the first phase of any computation, processes find out the system scale n by communication. (Surely, if such information has been given explicitly to each process, this phase is omitted.) In the second phase, process who knows the scale n simulates M in n steps. Only if M halts within n steps does a process p_i gratify a primitive proposition, say $halting_i$. Then it is obviously that M is *non-halting* is equivalent to the truth of $\forall i\ \Box\neg halting_i$. It follows that decision of $\forall i\ \Box\neg halting_i$ is Π^0_1-complete as *non-halting* problem is.

Actually, all undecidability results in [AK86], [Suz88], [GS92] and [EN95] were established with the above reduction technique. In [AK86], system scale n is explicitly given to every process and simulation is carried out by each individual process. Since processes' reachable states grow with n (the number of simulation steps), the system is potentially infinite. Sharper results were presented in [Suz88], [GS92], [EN95] where a distinguished *control process* use other *non-control process* as *"memory cell"* to collect such information and do the simulation. In those reduction every process has fixed finite states, however, it is not surprise that undecidability comes in as such systems (with a *control* process) are *universal*. A quick reduction goes like that: the *control* process represents the finite control part of a *Turing machine* **T** and all *non-control* processes compose of the infinite tape of **T**. What would happen if we restrict the communication ability of parameterized systems? Surely, we will get less useful computation models. As is shown in [GS92], such models do exist by allowing a complete communication network. With finite state constraint, it means that every process is anonymous and communication could happen between any two processes. The verification of process local properties is decidable in polynominal time. It is arguable that not many practical distributed algorithms fall in this category. Also as is pointed out in [GS92], the correctness problems of global properties for systems with control process but no complete communication network are undecidable. The effect of control process and correctness of global properties are all related to the system global knowledge. A question arises when we ask if there exists uniform procedure for determining a process's local properties, given a system in which no process can gain global knowledge in any computations. In other words, if a finite-state process has homogeneous view of it computation environment, would it be possible that the process's behavior in principle can be model checked? It is shown here that the verification problem on this more restricted class of parameterized systems is still undecidable. First we formalize the knowledge concept to precisely express the intuition of "knowing something". Then we identify a class of parameterized systems with completely identical finite state processes in which no process can know global knowledge in any computation. It seems that the computation and communication abilities of such class of systems are very limited. However, we show that undecidability doesn't disappear even for a very simple safety property.

2 Preliminaries

In this section, we beliefly review some concepts and introduce related notations for the future discussion.

2.1 Parameterized Systems

Parameterized System **P** is a tuple $\langle \mathbf{P}, \mathbf{N} \rangle$ where **P** is a process template and **N** is an index set. Assume that process template **P** is a *fair transition system* ([MP95]). A process is a *finite* if the underlying *fair transition system* is finite. A *parameterized finite system* is the one whose component processes are all *finite*. In this paper, we only study *parameterized finite systems*. Sometime, we omit word *finite*, however, the meaning should be clear in the context. Also we take **N** be the set of natural numbers for notation convenience. However we shall not assume any arithmetic properties other than indexing unless explicitly stated. We write p_i for the i-th process generated from the template with the index i. The i-th system instance (denoted by P^i) is a system that consists of processes p_k where $k \leq i$. I.e., $P^i = p_1 * p_2 * \ldots * p_i$. The communication network among processes is induced by the process definition. As a convention, a parameterized system has a *communication network* if the communication network is not a *complete graph*. Otherwise, we say the system has *no communication network*. A process is *anonymous* if the process's ID is not its local variable, but only an external reference. A parameterized finite system is *anonymous* if all its processes are *anonymous*. It is obvious that if a parameterized finite system with *no communication network* must be *anonymous*. A formula ϕ is a *local property* if ϕ is of form $\forall i\ \psi_i$ where ψ_i only involves primitive propositions determined by process p_i's local state. Otherwise, ϕ is a global property. We write $P^m \models \phi$ to say that P^m is a model of ϕ (in the normal sense). Let $\langle \mathbf{P}, \mathbf{N} \rangle \models \phi$ denote $\forall m \in \mathbf{N}\ P^m \models \phi$. $\langle \mathbf{P}, \mathbf{N} \rangle$ is *m-reducible* if $\langle \mathbf{P}, \mathbf{N} \rangle \models \phi$ iff $\exists m > 0\ \forall n \leq m\ P^n \models \phi$. The verification problem is to decide the correctness of $\langle \mathbf{P}, \mathbf{N} \rangle \models \phi$.

2.2 Computations

Concepts of *run, fairness and computation* are defined as in [MP95]. As most of our discussions are invariant to *fairness constraints*, we don't distinguish *run* and *computation* explicitly. As a notation convention, we use superscribes for system instance indexing and subscribes for process indexing. For example, σ^m denotes a computation of the instance P^m and (σ^m, j) denotes the system global state at time j in σ^m. Also σ_i^m denotes the *computation projection* of σ^m on process p_i. Obviously, we have the following equations whose meaning should be clear from the above definition.

$$\begin{aligned}
\sigma^m &= (\sigma^m, 0), (\sigma^m, 1), \ldots \\
\sigma^m &= \langle \sigma_0^m, \sigma_1^m, \ldots, \sigma_m^m, \rangle \\
(\sigma^m, i) &= \langle (\sigma_0^m, i), (\sigma_1^m, i), \ldots, (\sigma_m^m, i) \rangle, \ i \in \mathbf{N} \\
\sigma_i^m &= (\sigma_i^m, 0), (\sigma_i^m, 1), \ldots, \ i \in \{1, \ldots, m\}
\end{aligned}$$

As a graph representation,

$$\begin{array}{lccccc}
\sigma^m = \langle & \sigma_0^m & \sigma_1^m & \cdots & \sigma_m^m & \rangle \\
(\sigma^m, 0) = \langle & (\sigma_0^m, 0) & (\sigma_1^m, 0) & \cdots & (\sigma_m^m, 0) & \rangle \\
(\sigma^m, 1) = \langle & (\sigma_0^m, 1) & (\sigma_1^m, 1) & \cdots & (\sigma_m^m, 1) & \rangle \\
\multicolumn{6}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\
\multicolumn{6}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots}
\end{array}$$

Note that the normal process index begins with 1. However, we view environment as a special process representing all non-local properties in a system. We reserve subscript 0 for the environment process as in (σ_0^m, i) and $(\sigma^m, 0)$. To save notations, we may omit superscripts or subscripts whenever the meaning is clear in the context.

Definition 2.1. *Let σ be a computation of P. A finite prefix of σ is a computation history, denoted by $h(\sigma, n)$ where n is the length of the prefix. Similarly, we use $h_i(\sigma, n)$ to denote a n-length computation history of σ_i.*

By definition,

$$h(\sigma, n) = ((\sigma, 0), \ldots, (\sigma, n))$$
$$h_i(\sigma, n) = ((\sigma_i, 0), \ldots, (\sigma_i, n))$$

We say $h_i(\sigma_i, n)$ is a *computation history projection* of $h(\sigma, n)$ on process p_i, or simply a *computation history* of process p_i. Let $h_i(\sigma, n)$ and $h_i(\sigma', n')$ be two computation histories. As a convention, we write $h_i(\sigma, n) =_s h_i(\sigma', n')$ if state sequences of $h_i(\sigma, n)$ and $h_i(\sigma', n')$ are *stuttering equivalent.* Two computation points (σ, n) and (σ', n') are *equivalent* to process p_i (denoted by $(\sigma, n) \sim_i (\sigma', n')$), if $h_i(\sigma, n) =_s h_i(\sigma', n')$. Obviously, $\sim_i$ gives a equivalence relation on all computation histories in P. We use $H_i(\sigma, n)$ to denote the equivalence class. I.e.,

$$H_i(\sigma, n) = \{(\sigma', n') \mid (\sigma, n) \sim_i (\sigma', n')\}$$

A *computation history* $h_i(\sigma', n')$ is an *extension* of *computation history* $h_i(\sigma, n)$ if $h_i(\sigma, n)$ is a prefix of $h_i(\sigma', n')$. We write $h_i(\sigma, n) < h_i(\sigma', n')$. $h_i(\sigma', n')$ is a *proper extension* of $h_i(\sigma, n)$ if $h_i(\sigma, n) < h_i(\sigma', n')$ and $h_i(\sigma, n) \neq_s h_i(\sigma', n')$. $h_i(\sigma', n')$ is *successor* of $h_i(\sigma, n)$ (vice versa *predecessor*) if $h_i(\sigma', n')$ is the *maximal proper extension* of $h_i(\sigma, n)$. Similar definitions carry out on a global computation history (σ, n).

2.3 Graph Embedding

We use $V(G)$ (respectively $E(G)$) to denote node set (respectively edge set) of G.

Definition 2.2. *Given a graph $G = \langle V, E\rangle$ and a node $v \in V$, the v-neighborhood $N_g(v)$ is a subgraph of G which contains node v and all its adjacent nodes.*

Definition 2.3. *Given two graphs F and G, F is embedded in G (respectively, G is embedding F) if the following two condition hold:*

- *There is a homomorphism f from G to F ($G \xrightarrow{f} F$) such that $(u, v) \in E(G)$ implies $(f(u), f(v)) \in E(F)$ for $u, v \in V(G)$.*
- *For any node v in G, f is an isomorphism between $N_g(v)$ and $N_F(f(v))$ ($\forall\, v \in V_g, N_g(v) \xleftrightarrow{f} N_F(f(v))$*

The *graph embedding* was also called *graph covering* in [Ang80]. It is not hard to see that if graph F is *embedded* in graph G, $|F|$ (the size of F) must divide $|G|$ (the size of G). We define $|G|/|F|$ to be the *embedding degree*, denoted by $d(f)$. In Figure 1, graph F is embedded in graph G. The mapping f is shown by each pair of node labeling in G. Namely, $\{\langle 1, A\rangle, \langle 2, B\rangle, \langle 3, C\rangle, \langle 4, A\rangle, \langle 5, B\rangle, \langle 6, C\rangle\}$. The *embedding degree* $d(f)$ is 2.

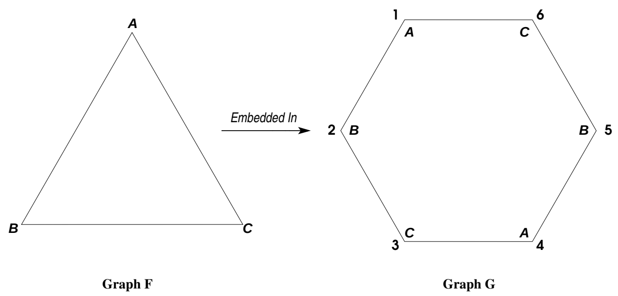


Figure 1: A Graph Embedding Example

3 Knowledge Framework

Formal logical analysis of knowledge has been found useful in reasoning correctness of distributed systems [FHV92]. A semantics framework related to concurrency of distributed systems was given by Halpern and Fagin, et al. in [HM90], [FHV92] and [RFV95a]. We first introduce general syntactic and semantic framework of knowledge and later show how to interpret computation models in such a framework. [1] We shall first point out that the *knowledge* we study here is *external*. We ascribe the *external knowledge* to processes (or generally agents) as a tool in reasoning their behaviors in distributed environments. It has nothing to do with cognitive abilities of processes (if they ever exists). Our formalization tool is *propositional modal* logic. Let $\mathcal{L}_n$ be the formal language augmented with n additional *knowledge operators* K_i for $i \in \{1, \ldots, n\}$. Let $\mathcal{M}_n = \langle W, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n\rangle$ be the semantics structure of $\mathcal{L}_n$. $\mathcal{M}_n$ is a standard Kripke structures with *possible world* relations $\mathcal{K}_i$ associated with each *knowledge operator* K_i for $i \in \{1, \ldots, n\}$. ϕ is a

[1] Most works in this section were done in order to prove positive results. Some are not necessary for the negative results represented in Section 5

mapping function which associates each $w \in W$ a set of primitive propositions. The truth of a formula ϕ defined recursively in standard way.

- $\langle M, w\rangle \models \phi$ iff ϕ is a primitive proposition and $\phi \in \pi(w)$.
- $\langle M, w\rangle \models \neg\phi$ iff $\langle M, w\rangle \not\models \phi$.
- $\langle M, w\rangle \models \phi \wedge \psi$ iff $\langle M, w\rangle \models \phi$ and $\langle M, w\rangle \models \psi$.
- $\langle M, w\rangle \models K_i\phi$ iff $\langle M, w'\rangle \models \phi$ for all w' where $(w, w') \in \mathcal{K}_i$.

The last definition captures the intuition that a process "knows" a formula ϕ in a world (state) w iff ϕ holds in all other worlds (states) w' it can not distinguish. All other truth semantics are defined as usual. Here we require $\mathcal{K}_i$ to be equivalent relations and the $\mathcal{M}_n$ is a standard **S5** structure. With the following axioms (marked in **S5** notation) $\mathcal{L}_n$ is sound and complete axiomization with respect to $\mathcal{M}_n$ [RFV95a].

- (A1) All tautologies substitution of propositional calculus (Tautology)
- (A2) [**K**] $(K_i\phi \wedge K_i(\phi \Rightarrow \psi)) \Rightarrow K_i\psi$ (Distribution Axiom)
- (A3) [**T**] $K_i\phi \Rightarrow \phi$ (Knowledge Axiom)
- (A4) [**4**] $K_i\phi \Rightarrow K_iK_i\phi$ (Positive Introspection Axiom)
- (A5) [**5**] $\neg K_i\phi \Rightarrow K_i\neg K_i\phi$ (Negative Introspection Axiom)
- (R1) From ϕ and $\phi \Rightarrow \psi$ infer ψ (Modus Ponen)
- (R2) From ϕ infer $K_i\phi$ (Knowledge Generalization)

The inference rule (R2) is called *knowledge generalization*. It says that if a formula ϕ is valid in the structure, then every process "knows" it in the structure. It results in the so-called *logic omniscience* problem, for it allows a process to have knowledge of all logical truth and its consequences. Our purpose is to formalize the intuition that a process *knows* some *global* configuration properties through its observation of its own computations. However, as long as we can express such properties in a formal system with rule R2, every process *implicitly knows* it "immediately". To overcome this difficulty, we extends the system to distinguish textit"observationally knows" and textit"logically knows". The new formal language $\mathcal{L}_n^o$ has additional *observational knowledge operator* K_i^o for $i \in \{1, \ldots, n\}$. K_i^o stands for *"observationally know"*. Intuitively, a process *"observationally know"* a formula ϕ in a state if ϕ is **true**and *observable* in the state. The semantics structure (*observational knowledge structure*) $\mathcal{M}_n^o$ is defined as $\langle W, \pi_1, \ldots, \pi_n, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n, \rangle$. Here we only introduce a new mapping function π_i which associate each world (state) with a set of primitive propositions which are *"observable"* in the state. The π_i is independent from π in the sense that some primitive propositions may be **true**by π but not "observable" by π_i in a state, and vice versa. The purpose of introduction of π_i is to define process p_i's *observable* facts in a state. Formally, we define an auxiliary semantics $\models_i$ for each $i \in \{1, \ldots, n\}$.

- (S0) $\langle M, w\rangle \models_i p$ iff $p \in \pi_i(w)$ where p is a primitive proposition.
- (S1) $\langle M, w\rangle \models_i \neg\phi$ implies $\langle M, w\rangle \not\models_i \phi$.
- (S2) $\langle M, w\rangle \models_i \phi \wedge \psi$ iff $\langle M, w\rangle \models_i \phi$ and $\langle M, w\rangle \models_i \psi$.

- (S3) $\langle M, w\rangle \models_i \phi \vee \psi$ iff $\langle M, w\rangle \models_i \phi$ or $\langle M, w\rangle \models_i \psi$.
- (S4) $\langle M, w\rangle \models_i \neg K_i\phi$ iff $\langle M, w\rangle \not\models_i K_i\phi$.
- (S5) $\langle M, w\rangle \models_i \neg K_i^o\phi$ iff $\langle M, w\rangle \not\models_i K_i^o\phi$.
- (S6) $\langle M, w\rangle \models_i K_i\phi$ iff $\langle M, w'\rangle \models_i \phi$ for all w' where $(w, w') \in \mathcal{K}_i$.
- (S7) $\langle M, w\rangle \models_i K_i^o\phi$ iff $\langle M, w'\rangle \models_i \phi$ for all w' where $(w, w') \in \mathcal{K}_i$.
- (S8) $\langle M, w\rangle \models_i \neg\phi$ iff $\langle M, w\rangle \models_i \psi$ (where ϕ is not of any form listed above, and ψ is the syntactically transformed from ϕ by pushing in negation).

The last definition is still sound even if temporal operators are introduced. The semantics for formula $K_i^o\phi$ is:

$$\langle M, w\rangle \models K_i^o\phi \text{ iff } \langle M, w'\rangle \models_i \phi \text{ and } \langle M, w'\rangle \models \phi \textit{ for all } (w, w') \in \mathcal{K}_i$$

For all other type of formulas, the definition is the same as in the original knowledge structure. If we introduce additional *observation operator* O_i for $i \in \{1, \ldots, n\}$, we can unify the semantics definition as follows.

$$\langle M, w\rangle \models O_i\phi \text{ iff } \langle M, w'\rangle \models_i \phi \text{ for all } w' \text{ where } (w, w') \in \mathcal{K}_i$$

$$\langle M, w\rangle \models K_i^o\phi \text{ iff } \langle M, w\rangle \models O_i\phi \text{ and } \langle M, w\rangle \models K_i\phi$$

In the appendix, we show the equivalence between the *observational knowledge structure* and *awareness structure*, a complete syntactic approach to attack *logical omniscience* problem introduced in [RFV95b]. The soundness and completeness follows from soundness and completeness of *awareness structure* by introducing new axioms for *observation operator* O_i. Since the result are mostly unrelated to the following discussion, we leave it to the appendix.

Now we can introduce temporal operators and interpret computation models in the *observational knowledge structure*. Let $(s_1, \ldots, s_n, s_e)$ denote a global state where s_i is the local state of process p_i and s_e is the environment state. Let (σ, n) be the global state at computation time n. I.e., $(\sigma, n) = (s_1, \ldots, s_n, s_e)$. The state(world) set in the knowledge structure consists of all computation histories $h(\sigma, n)$. Let π be a function on $h(\sigma, n)$ which is completely determined by the last state (σ, n). Here we keep π as the standard truth assignment of primitive propositions in a state. Let π_i be a function on $h(\sigma, n)$ which is determined by the i-projection $h_i(\sigma, n)$. Here assignment π_i stands for *observational facts* of process p_i at state (σ_i, n). Note that the difference between mapping function π_i and π is the definition domain. However, they all have the same range, namely the set of primitive propositions under consideration. Recall that two computation points (σ, n) and (σ', n') are i-equivalent ($(\sigma, n) \sim_i (\sigma', n')$) if $h_i(\sigma, n)$ is *stuttering equivalent* to $h_i(\sigma, n')$ ($h_i(\sigma, n) =_s h_i(\sigma', n')$). Obviously, π_i has the same value on two i-equivalent computation points. Note that by definition of π_i, a process has *perfect recall* ability. A process's knowledge is *cumulative* if the process has *perfect recall* ability. Intuitively, it increases the *"knowledge level"* of a process, for a process can maximally distinguish computation sequences. It is not hard to see that *perfect recall* requires *unbounded* memory as the previously computation sequence needs to be coded into the process's current state. However, *perfect recall* ability will give us sharper negative result. Let's imagine that each process is associated with an *external agent* which is observing the computations of a process. Each *external agent* process the *perfect recall* ability and unlimited computation power. If such an *external observant* can not find a global configuration property only through its observation of a process's computation, it is reasonable to believe that a finite state process can not find it either. It turns out crucial in reasoning the limit of a process's knowledge. To define semantics of formulas with *temporal operators*, a new *successor world* relation $\mathcal{S}$ is introduced. $\mathcal{S}$ is a *serial* relation on computation points. $\langle(\sigma', n'), (\sigma, n)\rangle \in \mathcal{S}$ iff $h(\sigma', n')$ is a *successor* of $h(\sigma, n)$. With $\mathcal{S}$, the semantics of *temporal operators* are defined as usual in [MP95]. An exception is that *next* operator X is not allowed as it collides with *stuttering equivalence*. From the semantics definition, each assignment π_i induces a boolean *observation function* δ_i which characterizes all *observable* formulas for process p_i. δ_i satisfies the following soundness constraints. (δ_i is what we really need in reason processes' *observational* abilities.)

- (R0) if $\delta_i(h_i(\sigma, n), \phi)$ iff $h_i(\sigma, n) \models O_i\phi$
- (R1) if $\delta_i(h_i(\sigma, n), \neg\phi)$ implies $\neg h_i(\sigma, n) \models O_i\phi$
- (R2) if $\delta_i(h_i(\sigma, n), \phi \wedge \psi)$ iff $\delta_i(h_i(\sigma, n), \phi)$ and $\delta_i(h_i(\sigma, n), \psi)$
- (R3) if $\delta_i(h_i(\sigma, n), \phi \vee \psi)$ iff $\delta_i(h_i(\sigma, n), \phi)$ or $\delta_i(h_i(\sigma, n), \psi)$
- (R4) $\neg\delta_i(h_i(\sigma, n), \phi) \Longleftrightarrow \delta_i(h_i(\sigma, n), \neg K_i^o\phi) \Longleftrightarrow \delta_i(h_i(\sigma, n), \neg K_i\phi)$
- (R5) $\delta_i(h_i(\sigma, n), \phi) \Longleftrightarrow \delta_i(h_i(\sigma, n), K_i^o\phi) \Longleftrightarrow \delta_i(h_i(\sigma, n), K_i\phi)$

(R0)-(R3) corresponds to semantics rules (S0)-(S3) respectively. (R0) and (R1) state the soundness that a formula and its negation can not be both *observable* by any process. (R4) and (R5) come from semantics rules (S4)-(S7). They guarantee that K_i^o and K_i satisfy axiom (A4) and (A5) which are called *negative introspection* and *positive introspeciton* respectively. We don't require the function δ_i to be complete in the sense that it is possible that both ϕ and $\neg\phi$ are not *observable* by a process. If δ_i is complete, then K_i^o is equivalent to K_i. One question arises that how to fine an assignment function π_i (or alternatively δ_i). Again, to make negative results solid, we like processes to have such magic ability to find whatever function as long as such a function obeys all semantics requirements listed above. Since the validity problem in language with *temporal* and *knowledge* operators is decidable (though nonelementary [HV89]), processes (or actually we) can always have δ_i be a decision procedure. In the extreme, *false* will be a trivial sound candidate.

Now with all machinery of knowledge system, we introduce a bunch of concepts for the following sections. Let ϕ_g be a global configuration property. Let P be a system instance with an arbitrary size. A process p_i *observes* ϕ_g in a computation σ if $(P, \sigma, n) \models O_i\phi_g$ for some $n \geq 0$. I.e., $(P, \sigma) \models \Diamond O_i\phi_g$. A computation point (P, σ, n) is an *observation point* of ϕ_g if $(P, \sigma, n) \models O_i\phi_g$. p_i *knows* ϕ_g if $(P, \sigma) \models \Diamond K_i\phi_g$. p_i *observationally knows* ϕ_g if $(P, \sigma) \models \Diamond K_i^o\phi_g$. We also say ϕ_g is *observable* (respectively *knowable*, *observationally knowable*) by p_i. An instance P *observes* ϕ_g if there is a computation σ in P and a process p_i such that p_i observes ϕ_g in σ. Let $P \models \Diamond O\ \phi_g$ denote P *observes* ϕ_g. We have $P \models \Diamond O\ \phi_g$ iff $\exists\sigma\exists i\ (P, \sigma) \models \Diamond O_i\phi_g$. An instance P *consistently observes* a property ϕ_g if P *observes* ϕ and doesn't *observes* $\neg\phi_g$. Let $P \models^c O\phi$ to denote P *consistently observes* ϕ_g. We have $P \models^c O\phi$ iff $P \models \Diamond O\ \phi_g$ and $P \not\models \Diamond O\ \neg\phi_g$. An instance P *observationally knows* ϕ_g if P *consistently observes* ϕ_g and P *knows* ϕ_g at some *observation point* (P, σ, n). I.e., except that P *consistently observes* ϕ_g, there is a computation σ in P and a process p_i such that p_i *observationally knows* ϕ_g in σ. Let $P \models \Diamond K^o\ \phi_g$ denote P *observationally knows* ϕ_g. We have $P \models \Diamond K^o\ \phi_g$ iff $\exists\sigma\exists i\ (P, \sigma) \models \Diamond K_i^o\phi_g$ and $P \models^c O\phi$. A parameterized system $\langle\mathbf{P}, \mathbf{N}\rangle$ is *self-aware* of ϕ_g if every instance *observationally knows* ϕ. Let $\langle\mathbf{P}, \mathbf{N}\rangle \models A\ \phi_g$ denote $\langle\mathbf{P}, \mathbf{N}\rangle$ is *self-aware*. We have $\langle\mathbf{P}, \mathbf{N}\rangle \models A\ \phi_g$ iff $\forall i \in \mathbf{N},\ P \models K^o\phi_g$.

4 Unawareness

In this section, we give a class of *parameterized finite systems* $\mathcal{P}$ that are not *self-aware* of *configurational* global properties, or in short *configurations.* A parameterized property ϕ_g^m is a property indexed by system scale m. A parameterized property ϕ_g^m is *configurational* if ϕ_g^m doesn't hold in system of scale m' where $m' \neq m$. Let $\mathcal{C}_k$ the class of all sets of parameterized systems which are *self-aware* a *configurational* global properties. Let $\bar{\mathcal{C}_k}$ be the complement class of $\mathcal{C}_k$. Obviously, there are sets in $\mathcal{C}_k$ which are undecidable. Let ϕ_g^m be "there are m processes in the computation." ϕ_g^m is *configuration* as it only holds in system instance P^m. Actually, all four undecidability proofs ([AK86], [Suz88], [GS92] and [EN95]) rely on such a type of parameterized systems in $\mathcal{C}_k$, for either system scale is given to every process or can be computed by the control process. A question arises that if sets in class $\bar{\mathcal{C}_k}$ are all decidable. Unfortunately, the answer is still negative. It means there are undecidable parameterized systems even if no process can gain any *configurational* global knowledge.

Now we identify a set of parameterized systems $\mathcal{P}$ and show $\mathcal{P}$ in $\bar{\mathcal{C}_k}$. $\mathcal{P}$ consists of all parameterized systems with the following properties:

(1) All processes are anonymous. I.e., process ID is just an external reference for our discussion. It isn't stored in the local state of a process.

(2) There is no system-wide non-constant variable.

(3) Each process is a completely identical finite state transition system.

(4) Each process communicates with a constant number of processes. A process's *neighbors* are all processes it can communicate with. A process's *channels* is *communication links* between the process and its *neighbors.*

(5) Each process may have local variables of finite domains which is only readable by its *neighbors.* This condition is not necessary but sufficient to guarantee the possibility of the communication model.

In a system instance P^m, we can think all processes are organized on a ring of size m. For a process p_i, the *right* or *clockwise* direction (respectively *left* or *counter-clockwise* direction) consists of all process p_{i+j} (respectively p_{i-j}) where $j \in \{1, \ldots, \lfloor m/2 \rfloor\}$. Similarly, a channel of process has *right* or *clockwise* (respectively *left* or *counter-clockwise*) orientation if the channel direction is from p_i to some process in p_i's *right* or *clockwise* (respectively *left* or *counter-clockwise*) side. Here all arithmetics are defined in $[m]$. As a consequence of identical definition, a channel identically referenced in all processes form a set of communication rings. The set of rings may have more than one member with the same size smaller than the system scale as each channel links two processes j*-step wise* where j is not relatively prime to m. Thus, the communication network breaks up into a set of rings (possibly with different sizes in case that a process have more than one channels). In the following discussion, we concentrate on a ring with other process omitted which are not related to the generating channel. However, all results can be easily generalized to a set of rings. Finally, note that it is impossible to do leader election in such a system [Ang80].

Proposition 4.1. *Let P^m denote an instance of a system $\langle \mathbf{P}, \mathbf{N} \rangle$ in $\mathcal{P}$. Let G^m denote the communication network graph of P^m. Given two graphs G^{m_1} and G^{m_2} which both embeds G^m, then $G^{m_1+m_2}$ embeds G^m.*

Proof Sketch. Recall the convention that processes are sitting on a ring structure. For simplicity, assume nodes in G^m are labeled with $i \in \{1, \ldots, m\}$. Let f_1 (respectively f_2) be an embedding mapping from G^{m_1} (respectively G^{m_2}) to G^m. Label node v in G^{m_1} (respectively G^{m_2}) with i_1 (respectively i_2) if $f_1(v) = i$ (respectively $f_2(v) = i$). The embedding construction divides into two steps: (1) "horizontally" pasting two identical graphs (G^{m_1} and G^{m_2}) at a corresponding node pair i_1 and i_2 forming a new node i, (2) "vertically" split node i with incident edges rearranged and adjecent nodes relabeled if necessary. More specifically, after splitting, both nodes i_1 and i_2 keep the left-side neighborhood unchanged and "exchange" the right-side neighbors and corresponding edges. An illustration is shown in Figure 2. Obviously, the constructed graph is isomorphic to $G^{m_1+m_2}$ since node i_1 and i_2 maintain their neighborhood view by the "exchange", and any other unadjacent nodes are not affected by the construction. It follows that mapping $\{\langle i_1, i \rangle, \langle i_2, i \rangle \mid i \in \{1, \ldots, m\}\}$ will be an embedding homomorphism from $G^{m_1+m_2}$ to G^m. Notice that the construction is also applicable if some of communication connections are unidirectional. □

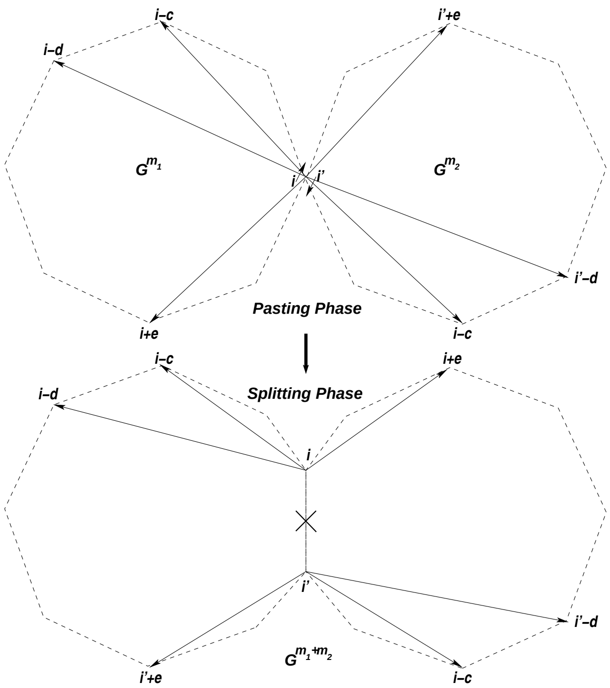


Figure 2: A Network Embedding Construction

Proposition 4.2. *Let G^m defined as before. G^{km} embeds G^m for all $k \in \mathbf{N}$.*

Proof Sketch. Immediate from proposition 4.1 by applying the paste-split construction $k-1$ times with G^m (for G^m embeds itself). □

Now we show $\mathcal{P}$ is in $\bar{\mathcal{C}}_k$. Let P^{m_1} and P^{m_2} be two system instances. Let σ^{m_1} and σ^{m_2} be computations in P^{m_1} and P^{m_2} respectively. We say σ^{m_1} is *embedded* in σ^{m_2} (or σ^{m_2} is *embedding* σ^{m_1}) if there exists a onto index mapping f from $\{1, \ldots, m_2\}$ to $\{1, \ldots, m_1\}$ such that the *computation projection* $\sigma^{m_1}_{f(i)}$ is a *stuttering equivalent* to *computation projection* $\sigma^{m_2}_i$. P^{m_1} is *computationally embedded* in P^{m_2} if any computation σ^{m_1} in P^{m_1} is *embedded* in some computation σ^{m_2} in P^{m_2}.

Proposition 4.3. *Given two system P^{m_1} and P^{m_2} with communication graph G^{m_1} and G^{m_2} respectively. If G^{m_1} is embedded in G^{m_2}, P^{m_1} is computationally embedded in P^{m_2}.*

Proof Sketch. Let function f be the embedding mapping from G^{m_2} to G^{m_1}. By graph embedding definition, f is onto. For any computation σ^{m_1} in P^{m_1}, we "clone" a computation σ^{m_2} in P^{m_2} by letting any process $p_i^{m_2}$ in P^{m_2} behave as the process $p_{f(i)}^{m_1}$ in P^{m_1}. If τ is a tranisiton executed in computation σ^{m_1} by process $p_i^{m_1}$, then we have all processes $p_j^{m_2}$s (where $j \in f^{-1}(i)$) to execute τ in a consecutive sequence in σ^{m_2}. More precisely, for any $n \geq 0$, we have $(\sigma^{m_1}, n) \xrightarrow{\tau} (\sigma^{m_1}, n+1)$ iff $(\sigma^{m_2}, n) \xrightarrow{\tau} (\sigma^{m_2}, n+1) \cdots \xrightarrow{\tau} (\sigma^{m_2}, n+|d(f)|)$ where $d(f)$ is the mapping degree of f. Obviously, σ^{m_2} is a computation in P^{m_2}. And by the construction, σ^{m_1} is *computationally embedded* in σ^{m_2}. □

Proposition 4.4. *Let σ^{m_1} and σ^{m_2} defined as before. Let ϕ_g a property. If σ^{m_1} is embedded in σ^{m_2} by function f,*

$$(P^{m_1}, \sigma^{m_1}) \models \Diamond O_{f(i)}\phi_g \textit{ implies } (P^{m_2}, \sigma^{m_2}) \models \Diamond O_i \phi_g$$

.

Proof Sketch. By the assumption that σ^{m_1} is *embedded* in σ^{m_2}, $\sigma_{f(i)}^{m1}$ is *stuttering equivalent* to σ_i^{m2}. By induction, it is easy to show $(\sigma^{m_1}, n) \models O_{f(i)}\phi_g$ implies $(\sigma^{m_2}, d(f) * n) \models O_i\phi_g$. That's

$$(P^{m_1}, \sigma^{m_1}) \models \Diamond O_{f(i)}\phi_g \text{ implies } (P^{m_2}, \sigma^{m_2}) \models \Diamond O_i \phi_g$$

. □

Proposition 4.5. *Let P^{m_1}, P^{m_2} and ϕ_g defined as before. If P^{m_1} is computationally embedded in P^{m_2}, then P^{m_1} observes ϕ_g implies P^{m_2} observes of ϕ_g.*

Proof Sketch. By Proposition 4.3, any computation σ^{m_1} in P^{m_1} is embedded in a computation σ^{m_2} in P^{m_2}. Let σ^{m_1} be a computation which *observes* ϕ_g. By Proposition 4.4, σ^{m_2} *observes* ϕ_g too. □

Proposition 4.6. *Let P^{m_1}, P^{m_2} defined as before. Let ϕ_g and ψ_g denote two observation inconsistent properties. If P^{m_1} is computationally embedded in P^{m_2}, then P^{m_1} consistently observes ϕ_g implies P^{m_2} doesn't consistently observe ψ_g.*

Proof Sketch. Consider the contrary that P^{m_1} and P^{m_2} both *consistently observe* ϕ_g and ψ_g respectively. By Proposition 4.5, P^{m_2} *observes* ϕ_g. Since ϕ_g and ψ_g are *observation inconsistent*, it follows that P^{m_2} *observes* $\neg\psi_g$. It contradicts to the fact that P^{m_2} *consistently observes* ψ_g. □

Proposition 4.7. *Let $\langle \mathbf{P}, \mathbf{N} \rangle$ be a parameterized system in $\mathcal{P}$. $\langle \mathbf{P}, \mathbf{N} \rangle$ is unware of any configurational global property ϕ_g.*

Proof Sketch. Assume $\langle \mathbf{P}, \mathbf{N} \rangle$ is *aware* of some *configurational* global property ϕ_g^m. It must be case that $\langle \mathbf{P}, \mathbf{N} \rangle$ has two instances P^{m_1} and P^{m_2} which both *observationally know* $\phi_g^{m_1}$ and $\phi_g^{m_2}$ repspetively. It follows that P^{m_1} (respectively P^{m_2}) *consistently observes* $\phi_g^{m_1}$ (respectively $\phi_g^{m_2}$). It is not difficult to also have P^{m_1} *computationally embedded* in P^{m_2}. By definition of *configurational* global property, $\phi_g^{m_1}$ is *observation inconsistent* to $\phi_g^{m_2}$. A contradiction immediately follows from Proposition 4.5. □

Theorem 4.1. *$\mathcal{P}$ is in $\bar{\mathcal{C}}_k$.*

Proof Sketch. Immediate from Proposition 4.7. □

5 Undecidabilities

Let $\mathcal{P}$ be the class of parameterized systems defined before. In this section, we present several undecidability proofs on correctness problems of $\mathcal{P}$ and its sublclasses. The first proof is based on reduction from *non-halting* problem of *two register machines* with empty input tape ([Min67a]) to the correctness of a safety property of $\mathcal{P}$. A two register machine is a finite control machine with only two types of instructions. Namely, the *increment instruction* $\mathbf{a}'$ and the *conditional decrement instruction* $\mathbf{a}^-(n)$ where a is a register's name. $\mathbf{a}'$ increases a's value by one unity. $\mathbf{a}^-(n)$ decreases a's value by one unity if the value is not zero. Otherwise $\mathbf{a}^-(n)$ makes control jump to the n-th instruction.

Theorem 5.1. *If communications in $\mathcal{P}$ are bidirectional, there exists a safety property whose correctness problem is Π^0_1-complete (co-RE complete).*

Proof Sketch. The proof follows the techniques used in [EN95] and [GS92]. More precisely, given any two register machine R, we can construct a parameterized system $\langle \mathbf{P}, \mathbf{N} \rangle$ in $\mathcal{P}$ such that a safety property's correctness is equivalent to the *non-halting* problem of R. The only problem is that for systems in $\mathcal{P}$ there is no control process to coordinate the simulation of a *Turing machine.* However, bidirectional communication can overcome this difficulty. Recall the convention that process p_{i+c} (respectively, p_{i-d}) is the *right* or *clockwise* (respectively, *left* or *counter-clockwise*) neighbor of process p_i where c and d are small constants compared to the system instance size. Let R be a two register machine, $\langle \mathbf{P}, \mathbf{N} \rangle$ is constructed as follows. The process template $\mathbf{P}$ is a simulation program of R in the sense that would be clear soon. Let p_i stands for an arbitrary process. p_i has a register a_i of binary type. At beginning, all a_is are initialized to 0. p_i has two modes, namely control mode and slave mode, which are chosen undeterministically at the beginning of a computation.

- If p_i chooses the control mode, it begins the simulation of R by issuing instructions ($\mathbf{a}'$ or $\mathbf{a}^-(n)$) to its *right* neighbor and waits for response from that neighbor. p_i simply bounce back any instructions that it received from left side. It doesn't care whether the instructions are initiated by itself or other processes in control mode. Actually it can not tell the difference and it doesn't need to care at all.
- If p_i chooses the slave mode, it waits instructions sent from its *left* neighbor and processes the instruction. Upon the procession result, it may forwards the processed instruction to its right neighbor or send back a response message to its *left* neighbor. More precisely, if p_i receives $\mathbf{a}'$ instruction and the value of a_i is 0, it changes the value to 1 and responds its *left* neighbor with message *"$\mathbf{a}'$ is done"*. Similarly, if p_i receives $\mathbf{a}^-$ instruction and the value in a_i is 1, it changes the value to 0 and responds its *left* neighbor with the message *"$\mathbf{a}^-$ is done"*. In all other cases, p_i simply forwards $\mathbf{a}'$ or $\mathbf{a}^-$ intact to its *right* neighbor. Also, p_i simply forwards a message which it received from the *right* neighbor to the *left* neighbor.

Figure 3 illustrates such a simulation. From the above construction, it is easy to see that by the message received from its *right* neighbor, a control process can know the instruction result.

Thus, each control processes partition the network graph to intervals and "manipulate" the slave processes on the *right interval* as its *"memory cells"*. In case that the instruction $\mathbf{a}'$ returns intact, the control process knows that an overflow happens and it stop the simulation. Only if a simulation halts, does the control process p_i make primitive proposition $halting_i$ true. Clearly, when a simulation halts, it has gone steps at least as the number of *"interval memory cells"*. It is obvious that there always exists such a simulation with the number of *"memory cell"* nearly as many as the maximum length of the communication ring. It is the case that only one process chooses the control mode and others choose the slave mode. Since the network size can grow unbounded, R is *non-halting* iff for any process p_i in all system instances $\langle \mathbf{P}, \mathbf{N} \rangle$, $\Box\neg halting_i$ holds. Thus, the decision problem of $\forall i\ \Box\neg halting_i$ is co-RE hard. Obviously, it is in co-RE, and then Π^0_1-complete (co-RE complete). $\Box$

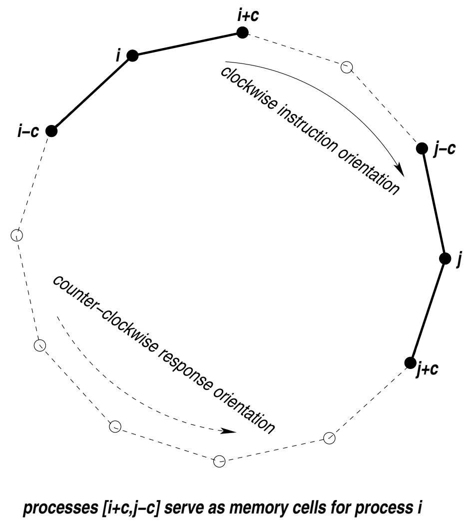
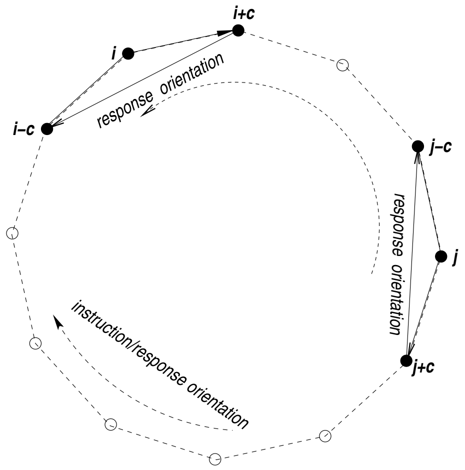


Figure 3: A Two Register Machine Simulation in A Bidirectional Network

Corollary 5.1. *If communications in $\mathcal{P}$ are unidirectional but have different orientations, there exists a safety property whose correctness problem is Π^0_1-complete (co-RE complete).*

Proof Sketch. We only need to show that a communication network with different orientations can simulate bidirectional communications. Figure 4 gives an example of such communication network. $i \rightarrow i+c$ is a *clockwise instruction/response* channel and $i \rightarrow i-2c$ is a *counter-clockwise response* channel. Most of construction is the same as in the proof of proposition 5.1. As a process can distinguish channels, it can duplex the *clockwise* channel for both instruction and 1-step response relay. In Figure 4 $i \rightarrow i-2c \rightarrow i-c$ simulates bidirectional communications. Note that the construction still keep every process totally identical. $\Box$

The above proofs replies on the existence of bidirectional communications or different *communication orientations*. The proofs breaks down if all communication channels are in the same orientation. In this case, any simulator may interfere with others since *"memory cells"* (the slave processes) can not distinguish instructions from any two simulators. However we soon show the situation doesn't improve even if the communication network is unidirectional and of the same orientation. Let $\mathcal{P}_u$ be a subclass of $\mathcal{P}$ with such type of parameterized systems. The reduction is established between *non-halting* problem of *tag systems* to the truth of a safety property in $\mathcal{P}_u$.



link i+c~i–c~i serves as response channel from i+c to i–c

Figure 4: A Two Register Machine Simulation in A Unidirectional Network

A *tag system* T is a tuple $\langle \Sigma, \Gamma, w_0, P \rangle$ where Σ is a finite alphabet, Γ a finite set of production rules, w_0 is an initial word on Σ, and P is a natural number, called *deletion number.* Let α_i be a letter of Σ for $i \in \{1, \ldots, |\Sigma|\}$. Each production rule τ_i is of form $\alpha_i \longrightarrow \beta_i$ where β_i is a word on Σ. If a word W on Σ beginning with letter α_i, then application of rule τ_i is to delete the beginning P letters and appending word β_i to the end. We say a word W' is the *successor* of W if W' is the result of application of some rule. *tag systems* have monogenic property that no more than one production rule can ever apply to a word. Except for monogenicity, *tag systems* also possess a simplicity feature: the production rule application to a word is totally determined by the first letter of the word. Surprisingly, such simple systems is still *universal.* It turns out that the *simplicity* and *universality* is essential in constructing an undecidability reduction. A run of *tag system* T is a sequence of words beginning with w_0 such that each word is the *successor* of the previous word in the sequence. T halts if its run is finite (*mortal*). I.e., at some point in the sequence, no further rule is applicable. Otherwise, T is *immortal.* For an *immortal* tag system T, there are two possibilities of the sequence depending on whether there is a repetitive word. Obviously, if a word is repeated in the sequence, by the *monogenic* property, the sequence will enter a loop forever after. In this case, T is *periodic.* If no word appears more than once, T is *infinite.* A related problem is that given a *tag system* without the initial word to decide if there exists a word whose production sequence is *immortal* or *periodic.* This is *immortality* or *unconstrained halting* (respectively *periodic* or *unconstrained periodic*) problem. The unsolvability of *halting problem* was established by Minsky by constructing a *universal* tag system with *deletion number* $P = 2$ [Min67b]. In [AB71], the result is strenghened such that halting problem for a tag system can have an arbitrary recursively enumerable degree of undecidability. Also in [AB71], the *immortality problem* is shown to be Σ_2^0-complete.

Our reduction is based on those undecidability results. Without loss of generality, we assume the communication orientation be *right* (*clockwise*). The construction is similar to those in the proof of proposition 5.1. Let T be a *tag system.* Let p_i defined as in Proposition 5.1. p_i encodes production rules Γ of T in its finite control. p_i has a set of instructions each of which corresponds to a production rule $\tau_j : \alpha_j \longrightarrow \beta_j$ in Γ (where $j \in \{1, \ldots, |\Sigma|\}$). Let τ_j also denote such an instruction. Also, p_i has a register $\mathbf{a_i}$ whose value domain is Σ. Let a_i stand for $\mathbf{a_i}$'s value. Assume we have two additional new symbols \$ and # which are not in Σ. \$ represents the end of a word and # represents an empty letter. At the beginning, the value in the register is \$. A special symobl \$ is introduced to indicate the end of a word. As before, p_i can choose its mode at initialization. We call a control process *head* and a slave process with \$ in the register *tail*.

- If p_i chooses the control mode, p_i sends instruction τ_j if $a_i = \alpha_j$. By sending τ_j, p_i is commanding its successors to execute τ_i and p_i itself shifts to the slave mode waiting for messages from its *left* neighbor. If no rule is applicable, it stops. The instruction τ_i encodes two numbers: *deleting count* C_d and *appending count* C_a that are initialized to *deletion number* P and 0 respectively.

- If p_i chooses the slave mode, it receives instructions from the *left* neighbor, process it and forward the processed to the *right* neighbor if necessary. More precisely, at receiving an instruction τ_j, p_i decodes the two counts C_d and C_a from τ_j and then decides in which phase τ_j has been carried out. If $C_d > 1$ (That means *"I'm be to deleted."*), it resets a_i to $\#$, decreases C_d and forwards the processed τ_j to its *right* neighbor. If $C_d = 1$ (That means *"I'm the new head."*), it sets C_d to 0, forwards the processed τ_j to its *right* neighbor, and shifts to the control mode. If $C_d = 0$, $C_a = 0$ and $a_i \neq \$$ (That means *"I'm not the tail."*), it simply forwards τ_j upon. Only if $C_d = 0$, $C_a = 0$, and $a_i = \$$, does an execution enter the appending phase. In this phase, p_i compares C_a with β_j. If $C_a \neq \beta_j$ (That means *"I'm in the appending phase."*), p_i sets a_i to $\beta_j[C_a]$, increase C_a by one and forwards it on. If $C_a = \beta_j$ (That means *"I'm the new tail."*), p_i only sets a_i to \$. At this time, the execution of τ_j is finished.

However, there are several technical details to be fixed.

- The communication is *synchronized.* If p_i is in the slave mode, it is in the *receive-process* loop. p_i will block if the other party is not ready. If p_i is in the control mode, it only receives messages from the *left* if sending is blocked. I.e., only if p_i can not issue instructions.

- If p_i (in the slave mode) with $a_i = \$$ receives τ_j with $C_d > 1$, it knows that a control process acutally is executing an *unapplicable* rule (as the word transformation already halted). In this case, it ignores τ_j and make primitive proposition $halting_i$ true.

- If p_i (in the control mode) receives an instruction from its *left* neighbor, it knows that there is another control process that are doing word processing. It is possible that the "other" process is just itself. But as usual, neither can it distinguish nor does it care. It switches to the slave mode and process the instruction according rules defined above. Note that by the communication rule, two word processing on a ring can not interfere with each other. When a control process receives a message, it hasn't sent out its own instruction. But after it sends out an instruction, the word that it is processing has begun moving 1-step away and has a new head (control process) instead. This is a worm-like moving since the tail hasn't expanded. Surely, it is possible that a *head* (control process) can be disabled by an instruction from its *left*. However, nothing bad happens as a word without *head* (control process) turns to be silent *"memory cells"*! It was *"eaten"* by another word (*worm*) on the ring (or maybe itself!)

- At the beginning of the simulation, p_i (if in the control mode) *"write out"* the initial word w_0 of T. It can be view as an execution of a special rule with $P = 0$ and $\beta_i = w_0$. It is not hard to fill out the details.

Note that all analyses applies even if communication is *asynchronous* but with order preserving and no loss. Figure 5 shows three execution phases of a production.

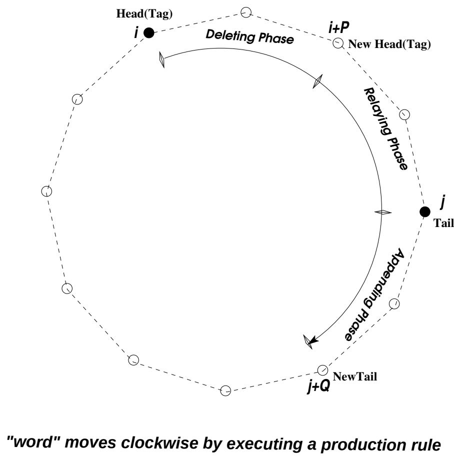


Figure 5: A Tag System Simulation in A Uniformly Oriented Network

Proposition 5.1. *A tag system T is not halting iff no process enter halting state forever.*

Proof Sketch. Immediate from the above construction. □

Proposition 5.1 gives a many-one reduction between *non-halting* problem of tag systems and correctness decision of the safety property $\forall i \ \Box \neg halting_i$.

Theorem 5.2. *[Min61] Any Turning machine can be represented , in a simple sense, as a "Tag" system, obtaining the unsolvability of the "Tag" problems for those systems representing, say, universal Turing machines.*

Theorem 5.3. *In $\mathcal{P}_u$, there exists a safety property whose correctness problem is Π^0_1-complete (co-RE complete).*

Proof Sketch. Immediate from Proposition 5.1 and Theorem 5.2. □

Proposition 5.2. *A tag system T is not periodic iff eventually forever all processes are out of control state.*

Proof Sketch. ($\Rightarrow$) By the above construction, if T is not *periodic*, it must be either *halting* or *infinite*. If T is *halting*, clearly a *control relay* will stop at some point. Given the fact that only finite control processes at any time, eventually all process are out of control state. If T is *infinite*, eventually a word's *tail* will catch up its *head*. Thus, each *head* is at least eventually "eaten" by itself.
($\Leftarrow$) Suppose T is *periodic*, there always exists a possible configuration (by enlarging the network size) that a word's *tail* can not catch up its *head*. Thus, in this configuration, at least one control process always survives. As the word *circulates* along the ring periodically, every process will enter control mode infinitely often. □

Proposition 5.2 also holds if we are considering *unrestricted periodic* problem. We could let a process have no initial word and no write-out at initialization. In stead, each process undeterministically chooses a letter during its initialization. Thus it leads to *"simpler"* finite controls in the sense that except for production rules, each process has no other *internal* state. Obviously, a tag system doesn't have *periodic* word iff $\forall i\ \Diamond\Box\neg control_i$. It is not very surprising that *decidability of periodicity problem* is also recursively unsolvable.

6 Conclusions

My belief was that if process can not gain global knowledge, its behavior should be saturated at some point. The totally symmetric systems are seemingly good candidates although their computation ability is very limited. It was my hope to first establish some positive results for totally symmetric systems. And the next step would be finding out how to decouple asymmetric part with symmetric part or transform almost symmetric systems to completely symmetric one (surely not in naive way if ever possible) and synthesize results to get correctness proofs. I too firmly believed that those total symmetric systems "should" be m-reducible and then can not evade model checkings in principle as each process has *homogeneous view* (local knowledge) beyond some scale. It would have saved a lot of time if I had searched counter-examples more carefully as they are not hard to construct given all conthers in the literature. However, this work at least shows w vigorously the correctness parameterized systems resists model-checking techniques. It seems that we can hardly avoid the case-by-case deductive approach in reasoning parameterized systems.

References

[AB71] S. Aanderaa and D. Belsnes. Decision problem for tag systems. *Journal of Symbolic Logic*, 36(2):229–239, 1971.

[AK86] K.R. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 30 May 1986.

[Ang80] D. Angluin. Local and global properties in networks of processes. In *Symposium on Theory of Computing*, pages 82–93, 1980.

[EN95] E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *Symposium on Principles of Programming Languages*, pages 85–94, 1995.

[FHV92] R. Fagin, J.Y. Halpern, and M.Y. Vardi. What can machines know? on the properties of knowledge in distributed systems. *Journal of the ACM*, 39:328–376, 1992.

[GS92] S.M. German and A.P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39:675–735, 1992.

[HM90] J.Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37:549–587, 1990.

[HV89] J.Y. Halpern and M.Y. Vardi. The complexity of reasoning about knowledge and time, i: lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[Min61] M.L. Minsky. Recursive unsolvability of post's problem of tag and other topics in theory of turing machines. *The Annals of Mathematics*, 74(3):437–455, November 1961.

[Min67a] M.L. Minsky. *Computation: Finite and Infinite Machines*, pages 255–258. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[Min67b] M.L. Minsky. *Computation: Finite and Infinite Machines*, pages 267–273. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[RFV95a] Y.M. Ronald Fagin, Joseph Y. Halpern and M.Y. Vardi. *Reasoning about Knowledge*. The MIT Press, Cambridge, Massachusetts, USA, 1995.

[RFV95b] Y.M. Ronald Fagin, Joseph Y. Halpern and M.Y. Vardi. *Reasoning about Knowledge*, pages 337–342. The MIT Press, Cambridge, Massachusetts, USA, 1995.

[Suz88] I. Suzuki. Proving properties of a ring of finite-state machines. *Information Processing Letters*, 28(4):213–214, 29 July 1988.

Appendix

To attack *logic omniscience* problem, [RFV95a] introduces *awareness structure*. An *awareness structure* is a tuple $M = \langle W, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n, \mathcal{A}_1, \ldots, \mathcal{A}_k \rangle$, where $\mathcal{A}_i$ are *awareness functions* for $i \in \{1, \ldots, n\}$. $\mathcal{A}_i$ associates each world (state) a set of syntactic formulas of which process p_i is *aware* in the world (state). The purpose of $\mathcal{A}_i$ is to disable processes' *logic omniscience* ability by letting them *unware* of something which is **true**. Also introduced are new modal operators A_i ("*aware of*") and X_i ("*explicitly know*"), and corresponding semantics rules. Let $\mathcal{L}_n^{\mathcal{A}}$ and $\mathcal{M}_n^{\mathcal{A}}$ denote the formal language and semantics framework of *awareness structure* respectively.

$$\langle M, w \rangle \models A_i\phi \text{ iff } \phi \in \mathcal{A}_i(w)$$
$$\langle M, w \rangle \models X_i\phi \text{ iff } \langle M, w \rangle \models A_i \text{ and } \langle M, w \rangle \models K_i$$

It is not hard to see that our *observational structure* with additional O_i operators is essentially an *awareness structure*. *Observational functions* $\mathcal{O}_i$ coincides with $\mathcal{A}_i$. In addition to the pure syntactic approach of *awareness structure*, we require *observational structure* to fulfill some semantics obligations ((S1)-(S7)). Let $\mathcal{L}_n^{\mathcal{O}}$ and $\mathcal{M}_n^{\mathcal{O}}$ denote the formal language and semantics framework respectively. As a consequence, $\mathcal{O}_i$ should satisfy the following constraints.

- $\phi \wedge \psi \in \mathcal{O}_i$ iff $\phi \in \mathcal{O}_i$ and $\psi \in \mathcal{O}_i$.
- $\phi \vee \psi \in \mathcal{O}_i$ iff $\phi \in \mathcal{O}_i$ or $\psi \in \mathcal{O}_i$.
- $\phi \in \mathcal{O}_i$ iff $O_i\phi \in \mathcal{O}_i$ iff $K_i\phi \in \mathcal{O}_i$ iff $K_i^o\phi \in \mathcal{O}_i$.
- $\neg O_i\phi \in \mathcal{O}_i$ iff $O_i\phi \notin \mathcal{O}_i$
- $\neg K_i\phi \in \mathcal{O}_i$ iff $K_i\phi \notin \mathcal{O}_i$
- $\neg K_i^o\phi \in \mathcal{O}_i$ iff $K_i^o\phi \notin \mathcal{O}_i$
- $\mathcal{O}_i(w) = \mathcal{O}_i(w')$ if $\langle w, w' \rangle \in \mathcal{K}_i$.

Theorem .1. *[RFV95a] $\mathcal{L}_n^{\mathcal{A}}$ is a sound and complete axiomization with respect to $\mathcal{M}_n^{\mathcal{A}}$ with the following additional axiom schema.*

$$X_i\phi \iff A_i\phi \wedge K_i\phi$$

Similarly, we have

Theorem .2. *$\mathcal{L}_n^{\mathcal{O}}$ is a sound and complete axiomization with respect to $\mathcal{M}_n^{\mathcal{O}}$ with the following additional axiom schemas.*

$$K_i^o\phi \iff O_i\phi \wedge K_i\phi$$
$$O_i(\phi \wedge \psi) \iff O_i\phi \wedge O_i\psi$$
$$O_i(\phi \vee \psi) \iff O_i\phi \vee O_i\psi$$
$$K_i^o \text{ and } O_i \text{ satisfy all axiom schemas of } \mathbf{S5}$$

Proof Sketch. Soundness proof is as usual. *Completeness* proof is essentially the same as that for Theorem .1 (by conjoining semantics truth and syntactical provability with *maximally consistent* sets.) Note that inference rule (R2) is disabled for K_i^o and O_i. Details should be carried out later. □