Complexity Analysis of Propositional Concurrent Programs Using Domino Tiling

Hsu-Chun Yen

Dept. of Electrical Engineering
National Taiwan University
Taipei, Taiwan 10764, R.O.C.

Namhee Pak

Misong Apt. 11-708
Pulgwang-dong
Seoul 122, Korea

Abstract

The complexities of the possible rendezvous and the lockout problems for propositional concurrent programs will be investigated in detail. We develop a unified strategy, based on domino tiling, to show that the above two problems with respect to a variety of propositional concurrent programs are complete for a broad spectrum of complexity classes, ranging from NLOGSPACE, PTIME, NP, PSPACE to EXPTIME. Our technique is novel in the sense that it demonstrates how two seemingly unrelated models, namely propositional concurrent programs and dominoes, can be linked together in a natural and elegant fashion.

1 Introduction

Imagine that a system consisting of k concurrent processes, each of which is a finite-state process of size at most n, is to be analyzed. Besides the obvious parameters, namely n and k, some implicit factors might also have impact on the complexity of the given system. Such factors include, for example, the process structure (cyclic vs. acyclic), the interprocess communication scheme (message passing vs. shared variable), the scheduling scheme, the topological structure of interprocess communication (linear, tree-like, or ring-like), and the degree of symmetry (which can be described by the number of types of processes). In many applications, it is often desirable to know exactly how the complexity of a problem varies when one or more of the above parameters change. In other words, our domain of interest consists of a family of mutually related problems, rather than a single problem. Such attempts can be found, for example, in [11,25].

In this paper, we will utilize a unified approach, based on domino tiling, to establish a comprehensive analysis of the *possible rendezvous* and the *lockout* problems for a variety of propositional concurrent programs. The establishment of a communication between two programs is referred to as a *rendezvous*. The *possible rendezvous problem* [21,22] (**PR**, for short) is that of determining, given a propositional concurrent program and a rendezvous, whether there exists a computation realizing the rendezvous. The *lockout problem* [13] (**LO**, for short) is to decide, given a propositional concurrent program $\mathcal{P}$, a propositional program T and a subset of states D (of T), whether T is locked out from D in $\mathcal{P}$. (T is said to be *locked out* from D if no matter how T proceeds, the remaining propositional programs in $\mathcal{P}$ can always prevent T from entering D.) What makes these two problems interesting and important is that they capture the essence of two fundamental properties, namely, *cooperation* and *antagonism* [11], concerning concurrent systems. PR is a cooperative property due to the fact that all programs act cooperatively to realize a rendezvous; whereas for LO, the distinguished program and the remaining programs have conflicting goals. Compared to previous results of similar nature in [11,13,21], we base our work on a more general model, study a broader class of problems, utilize a new and novel approach, and obtain several new results. (This will be elaborated later.) Our results are summarized in Table 1.

A *propositional program* is a directed labeled graph whose vertices and edges represent *program states* and *state transitions*, respectively. A *propositional concurrent program* is a collection of propositional programs for which inter-program communication is achieved in a hand-shaking fashion using commands of the forms $C!m$ and $C?m$, where $m \in \{0,1\}$, indicating the transmission and reception of data m over channel C. This model can be thought of as a general

0-8186-1060-3425/93 $03.00 © 1993 IEEE

framework within which programs are constructed using standard kinds of programming constructs (including concurrency and rendezvous, for example) from basic, uninterpreted, propositional program letters. Each such atomic program denotes an arbitrary binary relation on a universe of uninterpreted program states. As we will see later that this model of propositional concurrent programs is closely related to that of the *channel* version of *concurrent propositional dynamic logic* (*channel-CPDL*, for brevity) defined by Peleg in [17]. Aside from the connection to channel-CPDL, our model also provides a general framework for describing real-world concurrent programming languages such as ADA.

Prop. concurrent program				*Complexity*	
class	prog. size	prog. #	structure	**PR**	**LO**
$P_{n,n,G}$	O(n)	O(n)	G	Pspace-c	Exptime-c
$P_{n,n,NL}$	O(n)	O(n)	NL	NP-c	Pspace-c
$P_{n,1,G}$	O(n)	O(1)	G	Nlog-space-c	Ptime-c
$P_{1,n,G}$	O(1)	O(n)	G	Pspace-c	Exptime-c
$P_{1,n,NL}$	O(1)	O(n)	NL	NP-c	Pspace-c

Table 1: The complexity results of **PR** and **LO**. (G and NL represent general programs and programs without loops, respectively. X-c means X-complete.) Notice that all the lower bounds hold even when the structure of inter-program communication is tree-like.

A key contribution of this paper is the use of domino tiling for proving lower bounds for a number of problems concerning propositional concurrent programs. Domino tiling has its origins in a paper by Wang [24] more than two decades ago. Since then, it has been gaining more and more popularity. The merit of domino tilings is that dominoes are structurally simple and conceptually easy to visualize. Furthermore, they represent a unified model for which many important complexity classes can be defined in a natural way. As a result, they provide proofs which are easy to present and understand, compared to those involving brute-force reductions from Turing machines.

In summary, the contributions of this paper include the following:

- To some extent, our results complement that of [17] in a nice fashion. For channel-CPDL, our results suggest that even though the validity and satisfiability problems, in general, are undecidable, there are certain interesting properties which are decidable.
- Our work augments that of [21] (by Taylor), [13] (by Ladner), and [11] (by Kanellakis and Smolka). (More will be said about this in Section 3.) Our work can be thought of as a generalization of that of [11,13,21] in that we give a comprehensive analysis of PR and LO for a more general model, namely, propositional concurrent programs, with respect to the size, the number of programs and the structural constraints.
- Our proof technique is novel. We employ a unified strategy by reducing various domino tiling problems to the **PR** and **LO** for propositional concurrent programs in $P_{n,n,G}$, $P_{n,n,NL}$ and $P_{n,1,G}$. To our knowledge, with the exception of [26], domino tilings have never been used to prove results for problems concerning concurrent systems, although they have been used successfully for proving intractability and undecidability for logic and graph problems. Because of the regularity existing in domino tiling, our approach (of reducing from domino tiling problems) makes the proofs very easy to understand.
- Our results also reveal the impact caused by the program size and the number of programs in the overall complexity (which was not addressed in [13,21]). First, the complexities of **PR** and **LO** remain the same for propositional concurrent programs with a fixed program size as long as the number of propositional programs is a variable. However, if a propositional concurrent program has a fixed number of propositional programs, **PR** and **LO** can be determined in polynomial time.

2 Model of propositional concurrent programs

A *propositional program* (*PP*, for short) P is a directed labeled graph (S,T), where S (the set of vertices) and T (the set of edges) represent the sets of *program states* (or *states*, for short) and *state transitions*, respectively. For each propositional program, one of its states is designated as the *initial state*. Each transition is attached with a label, which represents the associated *atomic program*. For example, transition $p \xrightarrow{a} q$ means that atomic program a can be

executed from state p to reach state q. A *propositional concurrent program* (*PCP*, for short) of *concurrency* k (k-PCP) is a k-tuple $(P_1, P_2, ..., P_k)$, where each $P_i, 1 \leq i \leq k$, is a propositional program. In this model, there are two types of atomic programs, namely, *atomic internal programs* and *atomic communication programs*. Atomic internal programs represent those whose executions are independent of the rest of the concurrent program. Atomic communication programs, on the other hand, allow one PP to communicate with another through the utilization of *channels*. Atomic communication programs are of the form $C!0$ (resp. $C!1$) and $C?0$ (resp. $C?1$), indicating the transmission of data 0 (resp. 1) over channel C and the reception of data 0 (resp. 1) over channel C, respectively. In a PCP, the communication between PPs is synchronized based on the notion of 'hand-shaking.' If at any one time, two PPs A and B are in states p and q, respectively, and $p \stackrel{C!m}{\rightarrow} p'$ and $q \stackrel{C?m}{\rightarrow} q'$, $m \in \{0,1\}$, are two transitions from states p and q, respectively, then PPs A and B can communicate with each other by exchanging message m over channel C and simultaneously enter states p' and q', respectively. Then each PP continues its operations separately. The establishment of a such communication between PPs will be referred to as a *rendezvous* throughout the rest of this paper. (The word rendezvous is borrowed from the ADA language, which uses a similar hand-shaking mechanism for inter-task communication.)

Given a k-PCP $\mathcal{P}=(P_1, P_2, ..., P_k)$, a *global state* C is a k-tuple $(c_1, c_2, \cdots, c_k)$, where c_i is a state of P_i. Given two global states $C = (c_1, c_2, \cdots, c_n)$ and $D = (d_1, d_2, \cdots, d_n)$, D is said to *follow* C iff one of the following two conditions holds:

1. $\exists i,\ 1 \leq i \leq n$, such that $c_i \stackrel{a_i}{\rightarrow} d_i$ is a transition of P_i, for some atomic internal program a_i, and $\forall j,\ j \neq i,\ c_j = d_j$.

2. $\exists i, j,\ i \neq j,\ 1 \leq i, j \leq n$ such that $(c_i \stackrel{C!m}{\rightarrow} d_i) \in P_i$, $(c_j \stackrel{C?m}{\rightarrow} d_j) \in P_j$, for $m \in \{0,1\}$, and $\forall k,\ k \neq i$ and $k \neq j$, $c_k = d_k$.

An *execution sequence* in $\mathcal{P}$ is a sequence of global states $C_1, C_2, \cdots, C_k$ such that C_1 is the first global state and C_{i+1} follows C_i, $1 \leq i \leq k-1$. Note that the first global state of $\mathcal{P}$ is $(r_1, r_2, \cdots, r_n)$, where r_i is the initial state of P_i.

A global state C is *possible* iff there exists an execution sequence reaching C. A rendezvous (C!m, C?m), $m \in \{0,1\}$ between PPs P_i and P_j is *possible* iff there exists a possible global state $C = (c_1, c_2, \cdots, c_k)$ such that $c_i \stackrel{C!m}{\rightarrow} d_i \in P_i$ and $c_j \stackrel{C?m}{\rightarrow} d_j \in P_j$.

In this paper, we will focus on the following two problems:

1. The *possible rendezvous problem* (**PR**, for short): Given a PCP P and a rendezvous (e,a), is (e,a) possible ?

2. The *lockout problem* (**LO**, for short): Given a PCP P, a PP T_i and a subset of states D (of T_i), is T_i locked out from D in P ? (A PP T_i is said to be *locked out* from a set of states D if no matter how T_i proceeds, the rest of the system can always prevent T_i from entering D.)

3 Related work

Before presenting our complexity results, we first put our work into perspective by comparing our work with that of similar nature appearing in the literature.

As presented in the framework of propositional concurrent programs, our model is closely related to that of *propositional dynamic logic* (*PDL*, for short), which has its origins in [5] (cf. [7]). (See [5,7,12,18,19] for related results.) From the standpoint of computational complexity, the *validity* and *satisfiability* problems for PDL are complete for EXPTIME ([18,19]).

As an attempt to provide a framework within which concurrency can be dealt with, the model of PDL has been extended in [16] to encapsulate a new type of program construct '$\cap$.' Such an extended PDL is called *concurrent* PDL (CPDL, for short). Peleg [17] subsequently augmented CPDL to allow *communication* among concurrent programs to be described explicitly. According to the underlying notion of communication, such an extended CPDL is divided into two categories, namely *channel* and *shared variable* versions of CPDL, denoted as *channel-CPDL* and *shared-CPDL*, respectively. In channel-CPDL, communication between parallel processes is by means of channels. Two processes can communicate with each other by transmitting (receiving) 0/1 along channel C using commands $C!0/C!1$ ($C?0/C?1$). As to the complexity issues of CPDL with communication, the validity problem for channel-CPDL is undecidable, so is that for shared-CPDL.

In light of the above, our model is closely related to that of the channel-CPDL. There are, however, several key differences between our work and that of [17]. Most notably is that from the modeling standpoint,

concurrent programs in our model are described individually, as opposed to using a single integrated global state graph in channel-CPDL. In our setting, we are able to treat the number of concurrent programs, say k, as a natural parameter, and subsequently, investigate how the degree of concurrency (i.e., k) will affect the complexity of a problem. In channel-CPDL, the degree of concurrency is implicitly embedded in the number of branches emanating from the root of the global state graph, as the result of invoking '$\cap$' constructs. Viewing this, we feel that making the parameter explicit (as in the case of our model) might offer a better insight in understanding the role it plays in the overall complexity of a problem. As far as complexity analysis is concerned, our work complements the work of [17] by demonstrating that certain interesting properties of channel-CPDL are decidable, and, in fact, are complete for a wide spectrum of complexity classes ranging from NLOGSPACE to EXPTIME.

Since our model of propositional concurrent programs is a very general framework, our results presented in this paper can be applied to appropriate versions of concurrent models such as CCS, CSP, communication finite-state machines, Statecharts, Petri nets, and realistic concurrent languages (including ADA). With respect to the ADA language, related results can be found in [21]. In [21], Taylor analyzed the complexity of the possible rendezvous problem for a number of subclasses of ADA programs on which combinations of the following rules are imposed:

Rule 1. No *branches* and *loops* within a task's control flow are allowed.

Rule 2. No *select* statements are allowed.

Rule 3. Only one task may contain entry call statements for a given entry.

Rule 4. Synchronization statements occurring on the branches of an *if* statement may differ only in order of occurrence.

It turns out that PR is NP-complete in most cases, except for programs on which Rules 1, 2, and 3 are imposed (in this case, PR becomes solvable in PTIME). In particular, Taylor's NP-completeness result for Ada programs without *branches* and *loops* coincides with our NP-completeness of PR for $P_{n,n,NL}$. In a closely related work [11], Kanellakis and Smolka investigated the complexity of cooperative and antagonistic properties for networks of communicating processes. They showed that the problem of deciding cooperative properties such as termination and potential blocking is NP-complete. (The lower bound holds even for networks of acyclic processes in which processes are interconnected in a tree fashion or are of constant size.) On the other hand, it becomes PSPACE-complete for deciding antagonistic properties such as LO, even for networks of acyclic processes interconnected in a tree fashion. The above results parallel that of ours for $P_{n,n,NL}$ and $P_{1,n,NL}$ in Table 1. Finally, our EXPTIME-completeness result of LO for $P_{n,n,G}$ coincides with that in [13], in which LO for *systems of communicating sequential processes* was shown to be EXPTIME-complete.

4 Domino Tilings

A *domino* is a 1 × 1 unit square tile whose edges are colored and whose orientation is fixed. We denote by (left-color, right-color, bottom-color, top-color) a *domino type*, representing dominoes whose left, right, bottom, and top edges are colored left-color, right-color, bottom-color and top-color, respectively. Pictorially, Figure 1 shows a domino of type (a,b,c,d).

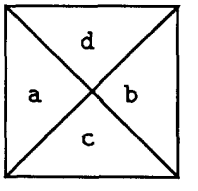

Figure 1: Domino type (a,b,c,d)

Given a domino type d, we use $left(d)$, $right(d)$, $top(d)$ and $bottom(d)$ to denote the left, right, top and bottom colors of d, respectively. For a set of domino types T, we define $Color(T)$ to be $\{left(d), right(d), top(d), bottom(d) \mid d \in T\}$, i.e., the set of colors appearing in T. Assuming that we have an infinite supply of copies of each domino type, the *domino problem*, in the generic sense, is that of determining whether a (finite or infinite) grid region in the Cartesian plane can be tiled using dominoes from T such that some specific constraints are met. Let $G(width, height)$ represent the region $\{(x, y) \mid 0 \leq x \leq width, 0 \leq y \leq height\}$, where $width$ and $height$ are in N $\cup\{\infty\}$. (For example, $G(\infty, \infty)$ represents the first quadrant.) A *domino system* is a tuple $(T, G(width, height), f)$, where T is a set of domino types, $G(width, height)$ is a region, and f: $\{1, 2, \cdots, width\} \times \{1, 2, \cdots, height\} \rightarrow$

T is the *tiling function.* (For convenience, we use $f_{i,j}$ to denote $f(i,j)$ throughout the remainder of this section.) $f_{i,j}$ can be thought of as the domino (in the tiling defined by f) whose upper right-hand corner is located at coordinate (i,j). A domino system $(T, G(width, height), f)$ is said to have a *solution* iff

1. $\forall i, 1 \leq i \leq width, bottom(f_{i,1})$ ='White',
2. $\forall i, 1 \leq i \leq width, top(f_{i,height})$ ='White', if $height < \infty$,
3. $\forall j, 1 \leq j \leq height, left(f_{1,j})$ ='White',
4. $\forall j, 1 \leq j \leq height, right(f_{width,j})$ ='White', if $width < \infty$,
5. $\forall i, j, 1 < i \leq width, 1 \leq j \leq height, left(f_{i,j}) = right(f_{i-1,j})$, and $\forall i, j, 1 \leq i \leq width, 1 < j \leq height, bottom(f_{i,j}) = top(f_{i,j-1})$.

In words, (1) - (4) ensure that external edges are colored 'White'. (5) ensures that adjacent edges of dominos have matching colors. The *domino problem* is that of determining, given T and G, whether there exists an f such that the domino system (T,G,f) has a solution.

In the literature, various bounded domino problems have been shown to be complete for a wide spectrum of complexity classes, including NP, PSPACE, EXPTIME, 2EXPTIME and more recently the entire polynomial time hierarchy. As a consequence, domino tiling has been shown to be very useful, as a reduction tool, for proving lower bound results. (See e.g., [1,3,4,6,8,9,14,15,20,23].) Among those domino problems discussed in the literature, the *2-person domino game*, introduced by Chlebus [3], consists of two players, CONSTRUCTOR and SABOTEUR, taking turns to tile a finite region using a given set of domino types. CONSTRUCTOR plays the first move by placing a domino in the lower left-hand corner. SABOTEUR then selects a domino for the second column and first row. Then CONSTRUCTOR selects a domino for the third column and first row, and so on. This sort of alternation continues until either the region is completely tiled or neither player is able to move. CONSTRUCTOR *wins* if the tiling is successful; otherwise, SABOTEUR wins. (At any time, if no legal move is available, then SABOTEUR wins.) A 2-person domino game is said to have a *winning strategy* for CONSTRUCTOR iff CONSTRUCTOR can always manage to win regardless of SABOTEUR's counter-moves. (The goal of CONSTRUCTOR is to finish the tiling, while SABOTEUR will try all he can to prevent this from happening.)

Four domino tiling problems to be used in this paper are described as follows:

- SQUARE TILING: Given a set of domino types T and an N (in unary), determine whether an $N \times N$ region can be tiled by dominoes from T.
- RECTANGLE TILING: Given a set of domino types T and an N (in unary), determine whether there exists a K such that a region of size $K \times N$ can be tiled by dominoes from T.
- SQUARE TILING GAME: Given a set of domino types T and an N (in unary), determine whether there exists a winning strategy for CONSTRUCTOR in tiling an $N \times N$ region using dominoes from T.
- RECTANGLE TILING GAME: Given a set of domino types T and an N (in unary), determine whether there exists a K such that CONSTRUCTOR has a winning strategy in tiling a rectangle of size $K \times N$ using dominoes from T.

Throughout the rest of this paper, we let $NLOGSPACE = NSPACE(\log n)$, $P = \bigcup_{i=1}^{\infty} DTIME(n^i)$, $NP = \bigcup_{i=1}^{\infty} NTIME(n^i)$, $PSPACE = \bigcup_{i=1}^{\infty} DSPACE(n^i)$, and $EXPTIME = \bigcup_{i=1}^{\infty} DTIME(2^{n^i})$, where $DTIME(T(n))$ (respectively, $NTIME(T(n))$) and $DSPACE(S(n))$ (respectively, $NSPACE(S(n))$) denote the classes of languages accepted in $T(n)$ time and $S(n)$ space, respectively, by some deterministic (respectively, nondeterministic) TMs. The reader is referred to [10] for more details.

The complexities of SQUARE TILING, RECTANGLE TILING, SQUARE TILING GAME and RECTANGLE TILING GAME are summarized in Table 2. See [3] and [26] for their proofs.

5 Complexity results

Recall that $P_{x,y,z}$, $x, y \in \{1, n\}$ and $z \in \{G, NL\}$, denotes the class of $O(y)$ PCPs, in which each PP is of size at most $O(x)$, and the structure of each PP is z (where G and NL stand for 'general' and 'no loop,' respectively).

Domino problem	*Complexity*
SQUARE TILING	NP-c
RECTANGLE TILING	PSPACE-c
SQUARE TILING GAME	PSPACE-c
RECTANGLE TILING GAME	EXPTIME-c
RECTANGLE TILING with Fixed Width	NLOGSPACE-c
RECTANGLE TILING GAME with Fixed Width	PTIME-c

Table 2: Complexities of various domino tiling problems.

For ease of expression, throughout the rest of this paper we will generalize the forms of the atomic communication programs, namely $C!a$ and $C?a$ (where $a \in \{0,1\}$), to allow message a to be a string of 0s and 1s. More precisely, $C!a$ and $C?a$ (where $a = a_1a_2...a_n \in \{0,1\}^*$) denote $C!a_1, C!a_2, ..., C!a_n$ and $C?a_1, C?a_2, ..., C?a_n$, respectively. As we will see later that in most cases, the intermediate states during the course of the execution of $C!a$ and $C?a$ are irrelevant; hence, they will be omitted.

Theorem 1 : **PR** *is NP-complete for* $P_{n,n,NL}$, *even when the structure of inter-program communication is linear.*

Proof.
1) (Upper bound)
Without loss of generality, consider a possible rendezvous $(C!a, C?a)$, $a \in \{0,1\}$, between PPs T_i, and T_j. Since loops are not allowed, any execution sequence realizing rendezvous $(C!a, C?a)$ is of length at most $O(n^2)$. Thus, **PR** for programs in $P_{n,n,NL}$ can be solved in NP.
2) (Lower bound)
This is shown by reducing an NP-complete problem, namely the SQUARE TILING problem, to the **PR**. We start with a simple example to demonstrate the first few steps of the simulation so as to allow the reader to have a better understanding of the proof. Consider a simple example of tiling a 3×3 square as shown in Figure 2(a). The corresponding simulation (by a PCP) of the first row is depicted in Figure 2(b). The action of placing the first domino is simulated by PP T_1. Upon reaching atomic program $C_{1,2}!r$, T_1 will become idle until T_2 reaches $C_{1,2}?r$; a rendezvous is then established. The synchronization between T_2 and T_3 is carried out in a similar fashion. One can easily see that the mechanism used for enforcing a matching color between two adjacent dominoes is by means of rendezvous. Nodes are used for keeping track of the color changes along the vertical direction during the course of the simulation.

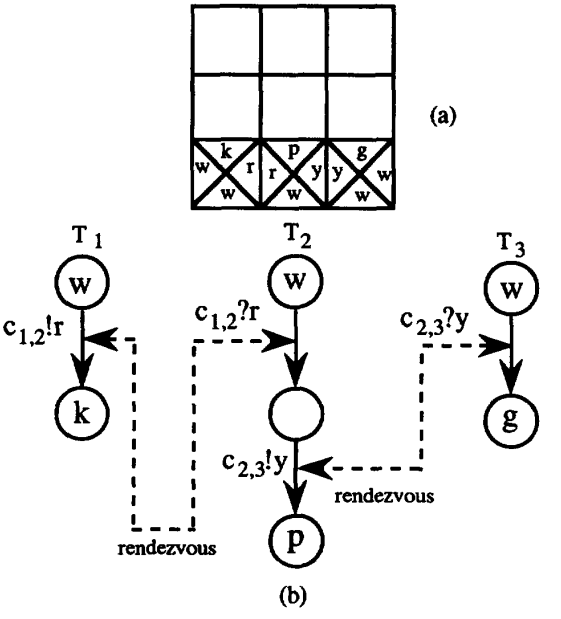

Figure 2: An example.

In the actual simulation, three kinds of PPs are involved, namely a PP simulating the first column of G, PPs simulating the intermediate columns of G, and a PP simulating the last column of G. Each PP is designed to simulate a particular column of G. In tiling G, PPs work together in a cooperative manner using dominoes from T. (In our previous example, PP T_i $(1 \leq i \leq 3)$ simulates the i-th column of G.) During the course of the simulation, the following two properties hold:

- In each column, the top color of the up-most domino will be kept in the current state of the simulating PP.
- The rule of which adjacent edges of dominoes have matching colors is simulated by rendezvous between PPs simulating adjacent columns of G.

For a PP simulating an intermediate column, if we ignore the intermediate states of the generalized atomic communication programs, three 'states' appearing in the order given below are used to mimic the action of placing a domino of type (a, b, c, d).

- $c \overset{C?a}{\rightarrow} x \overset{C'!b}{\rightarrow} d$, where C and C' are the respective channels and x is some intermediate state.

In the above sequence, states c and d record the top colors of the current column before and after a domino of type (a, b, c, d) is placed. $C?a$ is to synchronize with the preceding PP to ensure a matching color along the common boundary of the two dominoes (simulated by the PP and the preceding one). Finally, $C'!b$ is used so that the right color of the added domino, i.e., b, can be passed on to its right neighbor.

In the case of the first and the last PPs, only two 'states' are required to simulate the action of placing a domino of type (w, b, c, d) and (a, w, c, d), respectively. (This is because the left and right boundaries of G must always be 'white.') They are:

- First PP: $c \xrightarrow{C!b} d$.
- Last PP: $c \xrightarrow{C'?a} d$.

Since we are dealing with $P_{n,n,NL}$, each of the constructed PPs must be acyclic. This constraint can be enforced by making a slight modification to the above construction. Before doing this, we first explain why the above construction is not quite satisfactory. Consider the following scenario. Suppose in a column, two dominoes belonging to two different rows have the same top color, say p. If we do not distinguish these two p's from each other, then in the actual construction only one state will be used to represent p; this, in turn, will result in a loop, which is not allowed in $P_{n,n,NL}$. An easy way to circumvent this difficulty is to attach a number to each color, representing the row number in which the color appears. In this way, a unique state can then be used for each occurrence of a color. In Figure 2, such row numbers are omitted for the sake of simplicity.

Now we describe the detailed construction. Let $T = \{d_1, d_2, \cdots, d_m\}$ be the set of domino types in an instance of the SQUARE TILING and let N (in unary notation) be the width and height of the grid region to be tiled.

PP T_1: $G_1 = (S_1, F_1)$ (with initial state r_1), where $r_1 = 1w$,

- $S_1 = \{lc \mid 1 \leq l \leq N+1,\ c \in Color(T)\} \cup \{Done_1\}$,
- F_1 contains
  $lc \xrightarrow{C_2!lb} (l+1)d$, $\forall 1 \leq l \leq N$, if $(w, b, c, d) \in T$,
  and
  $(N+1)w \xrightarrow{C_2!*} Done_1$.

PP T_N: $G_N = (S_N, F_N)$ (with initial state r_N), where $r_N = 1w$,

- $S_N = \{lc \mid 1 \leq l \leq N+1,\ c \in Color(T)\} \cup \{Done_N\}$,
- F_N contains
  $lc \xrightarrow{C_N?la} (l+1)d$, $\forall 1 \leq l \leq N$, if $(a, w, c, d) \in T$,
  and
  $(N+1)w \xrightarrow{C_N?*} Done_N$.

PP T_i: $G_i = (S_i, F_i)$ (with initial state r_i), $1 < i < N$, where $r_i = 1w$,

- $S_i = \{lc \mid 1 \leq l \leq N+1,\ c \in Color(T)\} \cup \{D_i, Done_i\}$,
- F_i contains
  $lc \xrightarrow{C_i?la} x_l,\ x_l \xrightarrow{C_{i+1}!lb} (l+1)d$, $\forall 1 \leq l \leq N$, if $(a, b, c, d) \in T$,
  $(N+1)w \xrightarrow{C_i?*} D_i$, and $D_i \xrightarrow{C_{i+1}!*} Done_i$.

Note that in the above construction, the intermediate states needed to describe generalized atomic communication programs are omitted simply for the sake of simplicity.

Suppose the domino system has a solution. Then adjacent edges of columns must have matching colors, meaning that there exists an execution sequence finishing each PP T_i $(1 \leq i \leq N)$ successfully. Eventually, rendezvous $(C_N!*, C_N?*)$ is possible. Suppose rendezvous $(C_N!*, C_N?*)$ is possible. Then there exists an execution sequence finishing each PP successfully. By our construction, this execution sequence constitutes a solution of the domino system.

In general, a domino system consisting of domino types $T = \{d_1, d_2, \cdots, d_m\}$ with respect to $G(N, N)$ can be simulated by a PCP with N PPs. Each PP consists of $O(m*N*log(mN))$ states. (The $log(mN)$ factor comes from the number of the intermediate states needed to encode a channel message.) So the construction can be done in polynomial time. Thus, **PR** for $P_{n,n,NL}$ is NP-complete. It is worth pointing out that in the above construction, PP $T_i, 1 < i < N$, interacts only with T_{i-1} and T_{i+1}; hence, the structure of inter-program communication is linear. □

The upper bound results of Theorems 2-8 are easy to obtain. In what follows, we concentrate on the lower bound proofs.

Theorem 2 : **PR** *is PSPACE-complete for* $P_{n,n,G}$, *even when the structure of inter-program communication is linear.*

Proof. The lower bound can be shown by reducing the RECTANGLE TILING to the **PR** for $P_{n,n,G}$. Since the proof is very similar in style to that of Theorem 1, in what follows we only point out the differences and leave the details to the reader.

First notice that in the RECTANGLE TILING, the height of the rectangle to be tiled is not known in advance. In fact, this quantity can very well be exponential in the size of the input in a successful tiling ; hence, the proof technique used in Theorem 1 does not quite work. (Recall that in the previous construction, the number of nodes in a PP is proportional to the height of the region to be tiled.) Since PPs in $P_{n,n,G}$ are allowed to be cyclic, we can slightly alter the previous proof by removing the row number preceded a color in the representation of a node. That is, each color will be simulated by a single node, even when the color appears in two different rows. (Recall that in Theorem 1, different nodes are used if the color appears in two different rows.) In this way, the size of each PP can be limited to a polynomial (in the size of the input). □

Theorem 3 : **LO** *is PSPACE-complete for* $P_{n,n,NL}$, *even when the structure of inter-program communication is tree-like.*

Proof sketch.
1) (Upper bound)
To determine whether a PP T_i of a PCP P in $P_{n,n,NL}$ is locked out from a set of states D (of T_i) or not, it suffices to check whether P can always reach a state $C = (c_1, c_2, \cdots, c_n)$, for some $c_i \in D$, even when the remaining PPs are *malicious* against T_i. In a sense, the execution of a such program resembles the behavior of an alternating Turing machine. Consider the graph representing all possible computations of PCP P. Then all the states corresponding to PP T_i's moves can be thought of as *existential* states; the remaining states can be thought of as *universal states*. Accepting states are those whose i-th components (i.e., the states of T_i) are in D. As a result, the computation of P can be simulated by an alternating Turing machine. Furthermore, such an alternating Turing machine terminates in polynomial time due to the fact that PP P is acyclic. Thus, the PSPACE upper bound follows immediately from a well-known fact that PSPACE= APTIME. The reader is referred to [2] for more about alternating Turing machines and their related complexity classes.
2) (Lower bound)
This is shown by reducing the 2-person SQUARE TILING GAME (known to be PSPACE-complete) to the **LO**. To simulate the tiling of an $N \times N$ square area (where N is in unary notation), we use $N+2$ PPs E, A, and T_i, $1 \leq i \leq N$, where E (respectively, A) is used to simulate CONSTRUCTOR's (respectively, SABOTEUR's) moves and T_i is to keep track of the current top edge color in column i. Without loss of generality, assume that N is even. Compared to the proof technique used in Theorem 1, the key difference here is that each T_i does not select the next domino directly; instead, the selection is performed by PPs E and A in an alternate fashion. The key ideas are as follows:

1. The action of putting a domino in column i, $1 \leq i \leq N$, is simulated by a rendezvous between PPs T_i and E (respectively, A), if i is an odd (respectively, even) number.
2. As was the case in the proof of Theorem 1, PP T_i is used for keeping track of the color changes along the vertical direction in the i-th column.
3. PPs E and A simulate the moves made by CONSTRUCTOR and SABOTEUR, respectively.

For example, to simulate the first two moves made by CONSTRUCTOR and SABOTEUR, respectively, the following steps are involved:

1. PPs E and T_1 establish a rendezvous. (The selection is made by PP E, who is simulating CONSTRUCTOR.)
2. The control, together with the right color of the domino selected in the previous step, is then passed on from E to PP A by means of a rendezvous.
3. PP A then takes over. A will simulate the action of placing the second domino by first 'waking up' PP T_2; then executing an $C_{2A}?w$, for some color w, to retrieve (from T_2) the bottom color w of the domino to be tiled next; finally, establishing a rendezvous with T_2. (The last step corresponds to the selection of a domino by SABOTEUR.)
4. PP A then passes the control, together with the right color of the selected domino, back to E. (It is worth pointing out that in the previous step, if A cannot find a legal domino to tile, then A (i.e., SABOTEUR) will surrender by sending a 'win' message to PP E.)

Since the construction closely parallels that of Theorem 1, we therefore leave the remaining details to the reader. Notice that in the above construction, the structure of inter-program communication is tree-like. □

Theorem 4 : **LO** *is EXPTIME-complete for* $P_{n,n,G}$, *even when the structure of inter-program communication is tree-like.*

Proof. The lower bound can be shown by reducing from the 2-person RECTANGLE TILING GAME in a way similar to that in Theorem 3. The details are left to the reader. □

We now turn our attention to PCPs with a fixed number of PPs.

Theorem 5 : **PR** *is NLOGSPACE-complete for* $P_{n,1,G}$, *even when the structure of inter-program communication is linear.*

Proof. We first show the problem to be in NLOGSPACE. Given a PCP P in $P_{n,1,G}$, let k, a fixed constant, be the number of programs in P. The computation of P can be described by means of a 'global state graph,' in which each node represents a global state of P. In this way, the problem of determining whether a rendezvous is possible can be equated with that of deciding whether a global state enabling the rendezvous is reachable in the global state graph. However, the above global state graph should only be treated conceptually, for it takes $O(n^k)$ space simply to record a such graph. A straightforward nondeterministic search procedure (using $k * logn$ space to keep track of the contents of the current global state during the course of the search) is sufficient to find a global state enabling the given rendezvous, if a such state exists. Hence, the NLOGSPACE upper bound for the PR follows.

Using the same reduction technique as described in Theorems 1 and 3, we can reduce the RECTANGLE TILING with fixed width to the PR for $P_{n,1,G}$. The NLOGSPACE-hardness result follows. □

Now consider the lockout problem.

Theorem 6 : **LO** *is PTIME-complete for* $P_{n,1,G}$, *even when the structure of inter-program communication is tree-like.*

Proof. The upper bound can be proved along the same line as that of Theorems 3 and 4. That is, we can design an alternating Turing machine operating in $k * logn$ space to determine whether a given PP in a PCP in $P_{n,1,G}$ can be locked out of a designated set of states by the rest of the program. (Here k denotes the number of PPs.) It is well-known that ALOGSPACE=PTIME; the upper bound follows immediately.

The lower bound can be proved by reducing from the 2-person RECTANGLE TILING GAME with fixed width, which is known to be PTIME-complete (see [26]). □

In the rest of this section, we will show that the program size (constant vs. variable) plays a negligible role as far as the complexity of the PR is concerned. In other words, even if we restrict ourselves to PCPs with fixed size PPs only, the PR has the same upper and lower bounds as that for programs whose program size is a problem parameter. The result is somewhat surprising. This, in turn, reflects an important fact that the complexity of the PR is dominated by the number of PPs, rather than by the size of each PP. Before proving the above result, we require the following lemma:

Lemma 1 : *With respect to the PR and LO, PCPs in* $P_{n,n,G}$ *and* $P_{n,n,NL}$ *can be simulated by PCPs in* $P_{1,n,G}$ *and* $P_{1,n,NL}$, *respectively.*

Proof. First consider the PR. Essentially, we want to carry out a node-by-node transformation on the state graph to replace each node (i.e., state) together with its outgoing transitions (i.e., atomic programs) by a PP of fixed size. The first step is to, given a PP of a PCP in $P_{n,n,G}$, construct an equivalent PP in which each node has at most two incoming and two outgoing transitions. (This can easily be accomplished by introducing null statements at the juncture of each branch. Note that in the worst case, the number of nodes in the new PP is at most doubled.) Then the transformation is quite straightforward.

We now turn our attention to the LO. First notice that the above construction does not quite work for the LO. This is mainly because for the LO, we are dealing with a 'one against the rest of the system' situation. As a result, breaking a PP into several fixed-sized PPs, as was done in the above construction, does not preserve the flavor of 'one against many.' To overcome this difficulty, consider the following fix. Let (P,T,D) be an instance of the LO, where P is a PCP, T is a PP and D is the set of designated states that T is trying to execute. We first utilize the procedure mentioned above to transform P into a group P′ of fixed-sized PPs equivalent to P. Recall that in our transformation, each PP in P′ corresponds to a state for some PP in P. Let D′ be a subset of P′ such that each of D′ corresponds to a state in D. Now we introduce an additional PP T′ with a single transition $C?0$ leading to state x; in addition, a transition $C!0$ is inserted to each PP in D′. Clearly, PP T′ is locked out from $\{x\}$ in P′ iff T is locked out from D in P. This completes the proof. □

The following results follow directly from Lemma 1 and Theorems 1-4.

Theorem 7 : **PR** *is NP-complete for* $P_{1,n,NL}$.

Theorem 8 : **PR** *is PSPACE-complete for* $P_{1,n,G}$.

Theorem 9 : **LO** *is PSPACE-complete for* $P_{1,n,NL}$.

Theorem 10 : **LO** *is EXPTIME-complete for* $P_{1,n,G}$.

6 Conclusion

We have utilized a unified approach, based on domino tiling, to show that the possible rendezvous and the lockout problems for a variety of propositional concurrent programs are complete for a broad spectrum of complexity classes, ranging from NLOGSPACE, PTIME, NP, PSPACE to EXPTIME. (All of our complexity bounds are tight.) Aside from the novel technique itself, our results fully explain the role played by each of the parameters (such as the number of programs, the program size, the structure of interconnection, and the program structure) in the complexities of the above two problems.

Acknowledgements

We would like to thank the referees for suggestions which improved the content as well as the presentation of our results.

References

[1] Berger, R. The undecidability of the domino problem. *Mem. Amer. Math. Soc.*, 66, 1966.

[2] Chandra, A., Kozen, D. and Stockmeyer, L. Alternation. *JACM*, 28: 114-133, 1981.

[3] Chlebus, B. Domino-tiling games. *Journal of Computer and System Sciences*, 32: 374-392, 1986.

[4] Chlebus, B. Proving NP-completeness using bounded tiling. *J. Inf. Process. Cybern. EIK*, 8: 479-484, 1987.

[5] Fischer, M. and Ladner, R. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194-211, 1979.

[6] Gradel, E. Domino games and complexity. *SIAM J. Computing*, 19:787-804, 1990.

[7] Harel, D. Dynamic logic. *Handbook of Philosophical Logic* (D. Gabbay and F. Guenthner, eds.), Reidel Publishing Company, Dordrecht, 497-604, 1984.

[8] Harel, D. Recurring dominoes: making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24: 51-72, 1985.

[9] Harel, D. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *JACM*, 33(1): 224-248, 1986.

[10] Hopcroft, J. and Ullman, J. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Cambridge, MA, 1979.

[11] Kanellakis, P. and Smolka, S. On the analysis of cooperation and antagonism in networks of communicating processes. *Algorithmica*, 3: 421-450, 1988.

[12] Kozen, D. and Tiuryn, J. Logics of programs. *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), Vol. B, Elsevier, Amsterdam, 789-840, 1990.

[13] Ladner, R. The complexity of problems in systems of communicating sequential processes. *Journal of Computer and System Sciences* 21: 179-194, 1980.

[14] Lewis, H. Complexity of solvable cases of the decision problem for the predicate calculus. *IEEE FOCS*, pp. 35-47, 1978.

[15] Lewis, H. and Papadimitriou, C. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[16] Peleg, D. Concurrent dynamic logic. *JACM*, 34: 450-479, 1987.

[17] Peleg, D. Communication in concurrent dynamic logic. *Journal of Computer and System Sciences*, 35: 23-58, 1987.

[18] Pratt, V. Models of program logics. *Proc. 20th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 115-122, 1979.

[19] Pratt, V. A near-optimal method for reasoning about actions. *Journal of Computer and System Sciences*, 20:231-254, 1980.

[20] Savlsberg, M. and van Emde Boas, P. Bounded tiling, an alternative to satisfiability? *Proc. 2nd Frege Conference, Schwerin 1894, Mathematische Forschung*, pp. 354-363, 1984.

[21] Taylor, R. Complexity of analyzing the synchronization structure of concurrent programs. *Acta Informatica*, 19: 57-84, 1983.

[22] Taylor, R. A general-purpose algorithm for analyzing concurrent programs. *Communications of the ACM* 26(5): 362-376, 1983.

[23] van Emde Boas, P. Dominoes are forever. *Proc. of 1st GTI Workshop (Paderborn)*, pp. 76-95, 1983.

[24] Wang, H. Proving theorems by pattern recognition ii. *Bell System Tech. J.*, 40: 1-4, 1961.

[25] Yen, H. Communicating processes, scheduling, and the complexity of nontermination. *Mathematical Systems Theory*, Vol. 23, pp. 33-59, 1990.

[26] Yen, H. A multiparameter analysis of domino tiling with an application to concurrent systems. *Theoretical Computer Science*, Vol. 98, No. 2, pp. 263-287, 1992.