Part III

Scheduling

Chapter 7

Protocols with Workloads

The interaction protocol in a concurrent software imposes dependencies amongst the different processes in this software. For example, the rate at which a producer process can fill a buffer of bounded capacity depends on the rate at which a consumer process drains it. In most applications, the protocol is defined implicitly as a combination of locks and semaphores. For software written in an exogenous coordination language (cf., Part I) the protocol is explicit, which reveals the dependencies that are relevant for scheduling.

We can exploit this relevant scheduling information to optimize the execution of concurrent software. For example, smart scheduling of processes can offer protection against concurrent access of shared resources in a concurrent application, without suffering from drawbacks of the standard mutual exclusion protocols (e.g., locks). Imagine we have a crystal ball that accurately reveals when each process accesses its resources and their proper order of execution. We can then use this information to synthesize a scheduler that executes the processes in the correct order and prevents concurrent access to shared resources by speeding up or slowing down the execution of each process. Locks now become redundant, and their overhead can be avoided.

Unfortunately, some relevant scheduling information is lost if we represent the protocol in one of the existing protocol semantics (such as our stream constraints from Part II), which leaves us with a blurred crystal ball. Existing semantics encode precisely the order of all interactions, but they ignore the amount of work that must be performed in between these interactions. Therefore, existing protocol semantics are inadequate and prevent us from formulating the scheduling problem. In the current chapter, we develop *work automata*, which is a semantics for components and protocols that allows us to formalize the scheduling problem[1].

In Section 7.1 we introduce the syntax and semantics of work automata and define a weak simulation relation that allows us to compare the behavior of two work automata. We define two fundamental operations, namely composition and hiding. Composition allows us to construct a work automaton for a large system by composing the work automata of its smaller subsystems. The composition operation synchronizes the behavior on shared (inter)actions. Although these shared

[1]The work in this chapter is based on [DA17]

interactions are relevant for composition, we might want to ignore them in a larger context. The hiding allows us to do this. By defining composition and hiding of work automata, we can extend our Treo language from Chapter 4 with syntax to express primitive work automata. Our generic (semantics-agnostic) Reo then automatically computes the work automaton of the complete application. We potentially benefit from all semantics-independent compiler optimizations, such as the queue-optimization [JHA14] and protocol splitting [JCP16].

Composition of work automata may suffer from a state space explosion. A large number of states in a work automaton complicates its analysis. In Section 7.2, we introduce state-space minimization techniques to counter this state space explosion. We define in Section 7.2 two procedures, called *translation* and *contraction*, that simplify a given work automaton by minimizing its number of states. We provide conditions (Theorems 7.2.3 and 7.2.5) under which translation and contraction preserve weak simulation.

We show by means of an example that some large work automata can be simplified to their respectively "equivalent" single state work automata. The state-invariant of the single state of such a resulting automaton defines a region in a multidimensional real vector space. Geometric features of this region reveal interesting behavioral properties of the corresponding concurrent application. For example, (explicit or implied) mutual exclusion in an application corresponds to a hole in its respective region, and non-blocking executions correspond to straight lines through this region. Since straight lines are easier to detect than non-blocking executions, the geometric perspective provides additional insight into the behavior of an application. We postulate that such information may be used to develop a smart scheduler that avoids the drawbacks of locks.

In Section 7.3, we discuss related work, and in Section 7.4 we conclude and point out future work.

7.1 Work automata

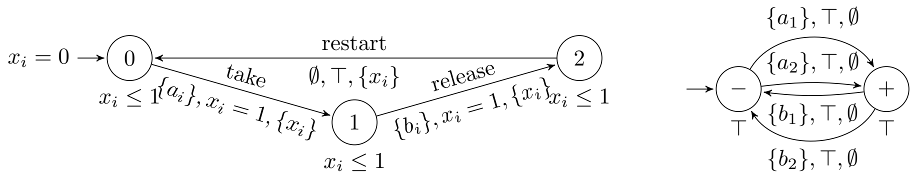
7.1.1 Syntax

Consider an application A that consists of $n \geq 1$ concurrently executing processes $X_1, \ldots, X_n$. We measure the progress of each process X_i in A by a positive real variable $x_i \in \mathbb{R}_+$, called a *job*, and represent the current *progress of application* A by a map $p : J \to \mathbb{R}_+$, where $J = \{x_1, \ldots, x_n\}$ is the set of all jobs in A. We regulate the progress using boolean constraints $\phi \in B(J)$ over jobs:

$$\phi \ ::= \ \top \ \mid \ \bot \ \mid \ x \sim n \ \mid \ \phi_0 \wedge \phi_1 \ \mid \ \phi_0 \vee \phi_1, \tag{7.1}$$

with $\sim \in \{\leq, \geq, =\}$, $x \in J$ a job and $n \in \mathbb{N}_0 \cup \{\infty\}$. We define *satisfaction* $p \models \phi$ of a progress $p : J \to \mathbb{R}_+$ and a constraint $\phi \in B(J)$ by the following rules: $p \models x \sim n$, if $p(x) \sim n$; $p \models \phi_0 \wedge \phi_1$, if $p \models \phi_0$ and $p \models \phi_1$; $p \models \phi_0 \vee \phi_1$, if $p \models \phi_0$ or $p \models \phi_1$. The *interface* of application A consists of a set of ports through which A interacts with its environment via synchronous operations, each one involving a subset $N \subseteq P$ of its ports.

We define the exact behavior of a set of processes as a labeled transition system called a *work automaton*. The progress value $p(x)$ of job x may increase in a state



$x_i = 0$
0
1
2
restart
$\emptyset, \top, \{x_i\}$
take
$\{a_i\}, x_i = 1, \{x_i\}$
release
$\{b_i\}, x_i = 1, \{x_i\}$
$x_i \leq 1$
$x_i \leq 1$
$x_i \leq 1$
$\{a_1\}, \top, \emptyset$
$\{a_2\}, \top, \emptyset$
$\{b_1\}, \top, \emptyset$
$\{b_2\}, \top, \emptyset$
$-$
$+$
$\top$
$\top$

(a) Process A_i (b) Lock L

Figure 7.1: Mutual exclusion of processes A_1 and A_2 by means of a lock L.

q of a work automaton, as long as the *state-invariant* $I(q) \in B(J)$ is satisfied. A state-invariant $I(q)$ defines the amount of work that each process can do in state q before it blocks. A transition $\tau = (q, N, w, R, q')$ allows the work automaton to reset the progress of each job $x \in R \subseteq J$ to zero and change to state q', provided that the *guard*, defined as *synchronization constraint* $N \subseteq P$ together with the *job constraint* $w \in B(J)$, is satisfied. That is, the transition can be fired, if the environment is able to synchronize on the ports N and the current progress $p : J \to \mathbb{R}_+$ of A satisfies job constraint w.

Definition 7.1.1 (Work automata)**.** A work automaton is a tuple $(Q, P, J, I, \to, \phi_0, q_0)$ that consists of a set of states Q, a set of ports P, a set of jobs J, a state invariant $I : Q \to B(J)$, a transition relation $\to \subseteq Q \times 2^P \times B(J) \times 2^J \times Q$, an initial progress $\phi_0 \in B(J)$, and an initial state $q_0 \in Q$.

Example 7.1.1 (Mutual exclusion)**.** Figure 7.1 shows the work automata of two identical processes A_1 and A_2 that achieve mutual exclusion by means of a global lock L. The progress of process A_i is recorded by its associated job x_i, and the interface of each process A_i consists of two ports a_i and b_i. Suppose we ignore the overhead of the mutual exclusion protocol. Then, lock L does not need a job and its interface consists of ports a_1, a_2, b_1, and b_2. Each process A_i starts in state 0 with $\phi_0 := x_i = 0$ and is allowed to execute at most one unit of work, as witnessed by the state-invariant $x_i \leq 1$. After finishing one unit of work, A_i starts to compete for the global lock L by synchronizing on port a_i of lock L. When A_i succeeds in taking the lock, then lock L changes its state from $-$ to $+$ and process A_i moves to state 1, its critical section, and resets the progress value of job x_i to zero. Next, process A_i executes one unit of work in its critical section. Finally, A_i releases lock L by synchronizing on port b_i, executes asynchronously its last unit of work in state 2, and resets to state 0. ♢

7.1.2 Semantics

We define the semantics of a work automaton $A = (Q, P, J, I, \to, \phi_0, q_0)$ by means of a finer grained labeled transition system $[\![A]\!]$ whose states are configurations:

Definition 7.1.2 (Configurations)**.** A configuration of a work automaton A is a pair $(p, q) \in \mathbb{R}_+^J \times Q$, where $p : J \to \mathbb{R}_+$ is a state of progress, and $q \in Q$ a state.

The transitions of $[\![A]\!]$ are labeled by two kinds of labels: one for advancing progress of A and one for changing the current state of A. To model advance of

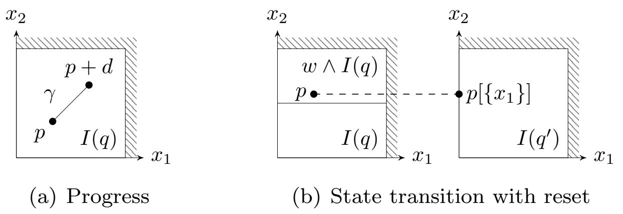


Figure 7.2: Progress (a) of the application along the path γ in $I(q)$ from p to $p+d$, and (b) transition from state q to q' with reset of job x_1.

progress of A, we use a map $d : J \to \mathbb{R}_+$ representing that $d(x)$ units of work has been done on job x. Such a map induces a transition

$$(p, q) \xrightarrow{d} (p + d, q), \tag{7.2}$$

where $+$ is component-wise addition of maps (i.e., $(p + d)(x) = p(x) + d(x)$, for all $x \in J$). Figure 7.2(a) shows a graphical representation of transition Equation (7.2). A state of progress p of A corresponds to a point in the plane.

In practice, the value of each job $x \in J$ continuously evolves from $p(x)$ to $p(x) + d(x)$. We assume that, during transition Equation (7.2), each job makes progress at a constant speed. This allows us to view the actual execution as a path $\gamma : [0, 1] \to \mathbb{R}_+^J$ defined by $\gamma(c) = p + c \cdot d$, where $\mathbb{R}_+^J$ is the set of maps from J to $\mathbb{R}_+$ and $\cdot$ is component-wise scalar multiplication (i.e., $(p+c\cdot d)(x) = p(x)+c\cdot d(x)$, for all $x \in J$). At any instant $c \in [0, 1]$, the state of progress $p + c \cdot d$ must satisfy the current state-invariant $I(q)$. Figure 7.2(a) shows execution γ as the straight line connecting p and $p + d$. For every $c \in [0, 1]$, state of progress $\gamma(c) = p + c \cdot d$ corresponds to a point on the line from p to $p + d$. Note that, since we have a transition from p to $p+c\cdot d$ in $[\![A]\!]$ for all $c \in [0, 1]$, Figure 7.2(a) provides essentially a finite representation of an infinite semantics, i.e., one with an infinite number of transitions through intermediate configurations between (p, q) and $(p + d, q)$. In Section 7.2.1, we use this perspective to motivate our gluing procedure.

The transition in Equation (7.2) is possible only if the execution does not block between p and $p + d$, i.e., state of progress $p + c \cdot d$ satisfies the state-invariant $I(q)$ of q, for all $c \in [0, 1]$. Since $I(q)$ defines a region $\{p \in \mathbb{R}_+^J \mid p \models I(q)\}$ of a $|J|$-dimensional real vector space, the non-blocking condition just states that the straight line γ between p and $p + d$ is contained in the region defined by $I(q)$ (see Figure 7.2(a)).

A transition $\tau = (q, N, w, R, q')$ changes the state of the current configuration from q to q', if the environment allows interaction via N and the current state of progress p satisfies job constraint w. As a side effect, the progress of each job $x \in R$ resets to zero. Such state changes occur on transitions of the form

$$(p, q) \xrightarrow{N} (p[R], q'), \tag{7.3}$$

where $p[R](x) = 0$, if $x \in R$, and $p[R](x) = p(x)$ otherwise. Figure 7.2(b) shows a graphical representation of transition Equation (7.3). The current state of progress satisfies both the current state-invariant and the guard of the transition, which allows to change to state q' and reset the value of x_1 to zero. For convenience, we allow at every configuration (p, q) an $\emptyset$-labeled self loop which models idling.

Definition 7.1.3 (Operational semantics)**.** The semantics of a given work automaton $A = (Q, P, J, I, \to, \phi_0, q_0)$ is the labeled transition system $[\![A]\!]$ with states $(p, q) \in \mathbb{R}_+^J \times Q$, labels $\mathbb{R}_+^J \cup 2^P$, and transitions defined by the rules:

$$\frac{d : J \to \mathbb{R}_+, \quad \forall c \in [0,1] : p + c \cdot d \models I(q)}{(p, q) \xrightarrow{d} (p + d, q)} \tag{S1}$$

$$\frac{\tau = (q, N, w, R, q') \in \to, \quad p \models w \wedge I(q), \quad p[R] \models I(q')}{(p, q) \xrightarrow{N} (p[R], q')} \tag{S2}$$

$$\frac{}{(p, q) \xrightarrow{\emptyset} (p, q)} \tag{S3}$$

where $p[R](x) = 0$, if $x \in R$, and $p[R](x) = p(x)$ otherwise.

Based on the operational semantics $[\![A]\!]$ of a work automaton A, we define the *trace semantics* of a work automaton. The trace semantics defines all finite sequences of observable behavior that are *accepted* by the work automaton.

Definition 7.1.4 (Actions, words)**.** Let P be a set of ports and J a set of jobs. An action is a pair $[N, d]$ that consist of a set of ports $N \subseteq P$ and a progress $d : J \to \mathbb{R}_+$. We write $\Sigma_{P,J}$ for the set of all actions over ports P and jobs J. We call the action $[\emptyset, \mathbf{0}]$, with $\mathbf{0}(x) = 0$ for all $x \in J$, the silent action. A word over P and J is a finite sequence $u \in \Sigma_{P,J}^*$ of actions over P and J.

Definition 7.1.5 (Trace semantics)**.** Let $A = (Q, P, J, I, \to, \phi_0, q_0)$ be a work automaton. A run r of A over a word $([N_i, d_i])_{i=1}^n \in \Sigma_{P,J}^*$ is a path

$$r \ : \ (p_0, q_0) \xrightarrow{N_1} \xrightarrow{d_1} s_1 \quad \cdots \quad s_{n-1} \xrightarrow{N_n} \xrightarrow{d_n} s_n$$

in $[\![A]\!]$, with $p_0 \models \phi_0 \wedge I(q_0)$. The language $L(A) \subseteq \Sigma_{P,J}^*$ of A is the set of all words u for which there exists a run of A over u.

Example 7.1.2. The language of the process A_i in Figure 7.1(a) trivially contains the empty word, and the word $u = [\emptyset, \mathbf{1}][\{a\}, \mathbf{1}][\{b\}, \mathbf{1}]$, where $\mathbf{1}(x_i) = 1$. Using Definitions 7.1.3 and 7.1.5, we conclude that $v = [\emptyset, \mathbf{1}][\{a\}, \mathbf{1}][\{b\}, \mathbf{0.5}][\emptyset, \mathbf{0.5}]$, with $\mathbf{0.5}(x_i) = 0.5$, is also accepted by A_i. Note that we can obtain v from u by splitting $[\{b\}, \mathbf{1}]$ into $[\{b\}, \mathbf{0.5}][\emptyset, \mathbf{0.5}]$. $\Diamond$

7.1.3 Weak simulation

Different work automata may have similar observable behavior. In this section, we define *weak simulation* as a formal tool to show their similarity. Intuitively, a weak simulation between two work automata A and B can be seen as a map that transforms any run of A into a run of B with identical observable behavior.

Following Milner [Mil89], we define a new transition relation, $\Rightarrow$, on the operational semantics $[\![A]\!]$ of a work automaton A that 'skips' silent steps.

Definition 7.1.6 (Weak transition relation)**.** For any two configurations s and t in $[\![A]\!]$, and any $a \in \mathbb{R}_+^J \cup 2^P$ we define $s \overset{a}{\Rightarrow} t$ if and only if either

1. $a = \emptyset$ and $s\ (\xrightarrow{\emptyset})^*\ t$; or
2. $a \in 2^P \setminus \{\emptyset\}$ and $s \overset{\emptyset}{\Rightarrow} s' \xrightarrow{a} s'' \overset{\emptyset}{\Rightarrow} t$; or
3. $a \in \mathbb{R}_+^J$, $s \overset{\emptyset}{\Rightarrow} s_1 \xrightarrow{c_1 \cdot a} t_1 \overset{\emptyset}{\Rightarrow} s_2\ \cdots\ t_{n-1} \overset{\emptyset}{\Rightarrow} s_n \xrightarrow{c_n \cdot a} t_n \overset{\emptyset}{\Rightarrow} t$, and $\sum_{i=1}^n c_i = 1$,

with $n \geq 1$, s_i, t_i configurations in $[\![A]\!]$, $c_i \in [0,1]$, $(c_i \cdot a)(x) = c_i \cdot a(x)$, for all $x \in J$ and all $1 \leq i \leq n$.

Definition 7.1.7 (Weak simulation)**.** Let $A_i = (Q_i, P, J, I_i, \rightarrow_i, \phi_{0i}, q_{0i})$, for $i \in \{0,1\}$ be two work automata, and let $\preceq\ \subseteq (\mathbb{R}_+^J \times Q_0) \times (\mathbb{R}_+^J \times Q_1)$ be a binary relation over configurations of A_0 and A_1. Then, $\preceq$ is a weak simulation of A_0 in A_1 (denoted as $A_0 \preceq A_1$) if and only if

1. $p_{00} \models \phi_{00} \wedge I_0(q_{00})$ implies $(p_{00}, q_{00}) \preceq (p_{01}, q_{01})$, with $p_{01} \models \phi_{01} \wedge I_1(q_{01})$;
2. $s \preceq t$ and $s \xrightarrow{a} s'$, with $a \in \mathbb{R}_+^J \cup 2^P$, implies $t \overset{a}{\Rightarrow} t'$ and $s' \preceq t'$, for some t'.

We call $\preceq$ a weak bisimulation if and only if $\preceq$ and its inverse $\preceq^{-1} = \{(t,s) \mid s \preceq t\}$ are weak simulations. We call A_0 and A_1 weakly bisimilar (denoted as $A_0 \approx A_1$) if and only if there exists a weak bisimulation between them.

7.1.4 Composition

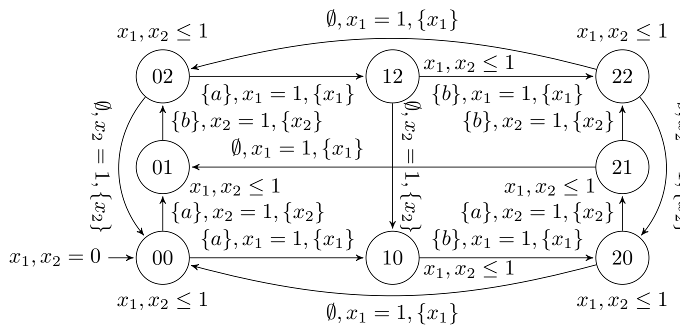
Thus far, our examples used work automata to define the exact behavior of a single job (or just a protocol L in Figure 7.1(b)). We now show that work automata are expressive enough to define the behavior of multiple jobs simultaneously. To this end, we define a product operator $\times$ on the class of all work automata. Before we turn to the definition, we first introduce some notation. For $i \in \{0,1\}$, let $A_i = (Q_i, P_i, J_i, I_i, \rightarrow_i, \phi_{0i}, q_{0i})$ be a work automaton and let $\tau_i = (q_i, N_i, w_i, R_i, q_i') \in \rightarrow_i$ be a transition in A_i. We say that τ_0 and τ_1 are *composable* (denoted as $\tau_0 \frown \tau_1$) if and only if $N_0 \cap P_1 = N_1 \cap P_0$. If $\tau_0 \frown \tau_1$, then we write $\tau_0 \mid \tau_1 = ((q_0, q_1), N_0 \cup N_1, w_0 \wedge w_1, R_0 \cup R_1, (q_0', q_1'))$ for the *composition* of τ_0 and τ_1.

Definition 7.1.8 (Composition)**.** Let $A_i = (Q_i, P_i, J_i, I_i, \rightarrow_i, \phi_{0i}, q_{0i})$, $i \in \{0,1\}$, be two work automata. We define the composition $A_0 \times A_1$ of A_0 and A_1 as the work automaton $(Q_0 \times Q_1, P_0 \cup P_1, J_0 \cup J_1, I_0 \wedge I_1, \rightarrow, \phi_{00} \wedge \phi_{01}, (q_{00}, q_{01}))$, where $\rightarrow$ is the smallest relation that satisfies:

$$\frac{i \in \{0,1\},\ \tau_i \in \rightarrow_i,\ \tau_{1-i} \in \rightarrow_{1-i} \cup \{(q, \emptyset, \top, \emptyset, q) \mid q \in Q_{1-i}\},\ \tau_0 \frown \tau_1}{\tau_0 \mid \tau_1 \in \rightarrow}$$

By means of the composition operator in Definition 7.1.8, we can construct large work automata by composing smaller ones. The following lemma shows that the composite work automaton does not depend on the order of construction.

Lemma 7.1.1. *$(A_0 \times A_1) \times A_2 \approx A_0 \times (A_1 \times A_2)$, $A_0 \times A_1 \approx A_1 \times A_0$, and $A_0 \times A_0 \approx A_0$, for any three work automata A_0, A_1, and A_2.*



$x_1, x_2 \leq 1$
$\emptyset, x_1 = 1, \{x_1\}$
$x_1, x_2 \leq 1$
02
12
22
$x_1, x_2 \leq 1$
$\{a\}, x_1 = 1, \{x_1\}$
$\{b\}, x_1 = 1, \{x_1\}$
$\{b\}, x_2 = 1, \{x_2\}$
$\{b\}, x_2 = 1, \{x_2\}$
$\emptyset, x_2 = 1, \{x_2\}$
$\emptyset, x_1 = 1, \{x_1\}$
$\emptyset, x_2 = 1, \{x_2\}$
$\emptyset, x_2 = 1, \{x_2\}$
01
21
$x_1, x_2 \leq 1$
$x_1, x_2 \leq 1$
$\{a\}, x_2 = 1, \{x_2\}$
$\{a\}, x_2 = 1, \{x_2\}$
$\{a\}, x_1 = 1, \{x_1\}$
$\{b\}, x_1 = 1, \{x_1\}$
$x_1, x_2 = 0$
00
10
20
$x_1, x_2 \leq 1$
$x_1, x_2 \leq 1$
$x_1, x_2 \leq 1$
$\emptyset, x_1 = 1, \{x_1\}$

Figure 7.3: The complete application $M = L \times A_1 \times A_2$. In state $q_1 q_2$, lock L is in state $(-1)^{q_1+q_2+1}$ and process A_i is in state q_i.

Example 7.1.3. Consider the work automata from Example 7.1.1. The behavior of the application is the composition M of the two processes A_1 and A_2 and the lock L. Figure 7.3 shows the work automaton $M = L \times A_1 \times A_1$. Each state-invariant equals $\top \wedge x_1 \leq 1 \wedge x_2 \leq 1$. The competition for the lock is visualized by the branching at the initial state 00. $\Diamond$

7.1.5 Hiding

Given a work automaton A and a port a in the interface of A, the *hiding* operator $A \setminus \{a\}$ removes port a from the interface of A. As a consequence, the hiding operator removes every occurrence of a from the synchronization constraint N of every transition $(q, N, w, R, q') \in \to$ by transforming N to $N \setminus \{a\}$. In case N becomes empty, the resulting transition becomes *silent*. If, moreover, the source and the target states of a transition are identical, we call the transition *idling*.

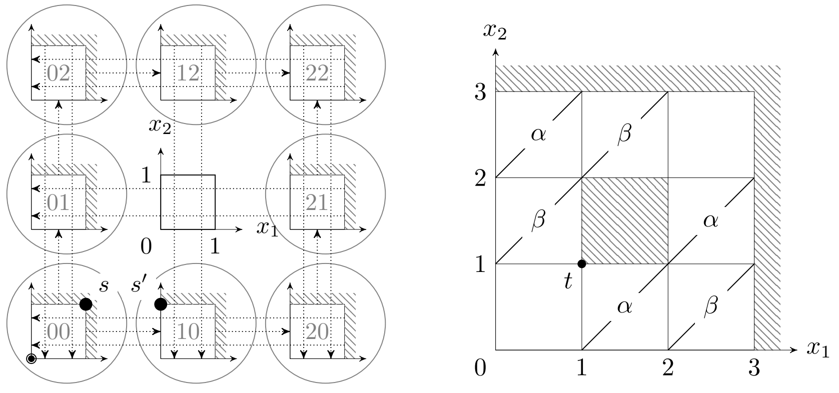
Definition 7.1.9 (Hiding)**.** Let $A = (Q, P, J, I, \to, \phi_0, q_0)$ be a work automaton, and $M \subseteq P$ a set of ports. We define $A \backslash M$ as the work automaton $(Q, P \backslash M, J, \to_M, \phi_0, q_0)$, with $\to_M = \{(q, N \setminus M, w, R, q') \mid (q, N, w, R, q') \in \to\}$.

Lemma 7.1.2. *Hiding partially distributes over composition:* $M \cap P_0 \cap P_1 = \emptyset$ *implies* $(A_0 \times A_1) \setminus M \approx (A_0 \setminus M) \times (A_1 \setminus M)$*, for any two work automata* A_0 *and* A_1 *with interfaces* P_0 *and* P_1*, respectively.*

Example 7.1.4. Consider the work automaton M in Figure 7.3. Work automaton $M' = M \setminus \{a, b\}$ is M where every occurrence of $\{a\}$ or $\{b\}$ is substituted by $\emptyset$. $\Diamond$

7.2 State Space Minimization

The composition operator from Definition 7.1.8 may produce a large complex work automaton with many different states. In this section, we investigate if, and how, a set of states in a work automaton can be merged into a single state, without breaking its semantics. In Section 7.2.1, we present by means of an example the basic idea for our simplification procedures. We define in Section 7.2.2 a *translation* operator that removes unnecessary resets from transitions. We define in Section 7.2.3



(a) $[\![M']\!]$ (b) $[\![K]\!]$

Figure 7.4: Graphical representation (a) of semantics $[\![M']\!]$ of the work automaton M' in Example 7.1.4, where white regions represent state-invariants, and (b) result after gluing the regions in(a). Starting in a configuration below line α and above line β, parallel execution of x_1 and x_2 never blocks on lock L.

a *contraction* operator that identifies different states in a work automaton. We show that translation and contraction are correct by providing weak simulations between their pre- and post-operation automata.

7.2.1 Gluing

The following example illustrates an intuitive *gluing procedure* that relates the product work automaton M in Figure 7.3 to the punctured square in Figure 7.4(b). Formally, we define the gluing procedure as the composition of translation (Section 7.2.2) and contraction (Section 7.2.3).

Example 7.2.1 (Gluing)**.** Consider the work automaton M' in Example 7.1.4 that describes the mutual exclusion protocol for two processes. Our goal is to simplify M' to a work automaton K that simulates M'. To this end, we introduce in Figure 7.4(a) a finite representation of the infinite semantics $[\![M']\!]$ of M', based on the geometric interpretation of progress discussed in Section 7.1.2. For any given state q of M', the state-invariant $I(q) = x_1 \leq 1 \wedge x_2 \leq 1$ is depicted in Figure 7.4(a) as a region in the first quadrant of the plane. Each configuration (p, q) of M' corresponds to a point in one of these regions: q determines its corresponding region wherein point p resides. Each transition of M' is shown in Figure 7.4(a) as a dotted arrow from the border of one region to that of another region. We refer to these dotted arrows as *jumps*. A jump λ from a region R of state q to another region R' of state q' represents infinitely many transitions from configurations (p, q) to configurations (p', q'), for all p and p', as permitted by the semantics $[\![M']\!]$. By the job constraint of the transition corresponding to λ, p and p' must lie on the borders of R and R', respectively, that are connected by λ.

From a topological perspective, a jump from one region to another can be viewed as ‘gluing’ the source and target configuration of that jump. We can glue any two

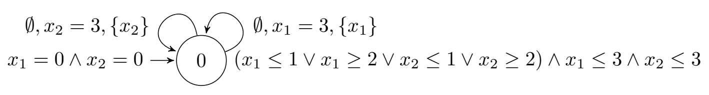

$\emptyset, x_2 = 3, \{x_2\}$
$\emptyset, x_1 = 3, \{x_1\}$
$x_1 = 0 \wedge x_2 = 0$
0
$(x_1 \leq 1 \vee x_1 \geq 2 \vee x_2 \leq 1 \vee x_2 \geq 2) \wedge x_1 \leq 3 \wedge x_2 \leq 3$

Figure 7.5: Work automaton K that corresponds to Figure 7.4(b).

regions in Figure 7.4(a) together by putting regions (i.e., state-invariants) of the source and the target states side by side to form a single state with a larger region. Each jump in Figure 7.4(a) from a source to a target state corresponds to an idling transition (c.f., rule Equation (S3) in Definition 7.1.3) within a single state. When we apply this gluing procedure in a consistent way to every jump in Figure 7.4(a), we obtain a single state work automaton K that is defined by a single large region, as shown in Figure 7.4(b). Figure 7.5 shows the actual work automaton that corresponds to this region. Note that the restart transition allows the state of progress to jump in Figure 7.4(a) from configuration $((x, 1), i2)$ to $((x, 0), i0)$ and from configuration $((1, y), 2j)$ to $((0, y), 0j)$, for all $x, y \in [0, 1]$ and $i, j \in \{0, 1, 2\}$. Thus, the restart transition identifies opposite boundaries in Figure 7.4(b), turning the punctured square into a torus. ♢

The next example shows that the geometric view of the semantics of the work automaton in Example 7.2.1 reveals some interesting behavioral properties of M'.

Example 7.2.2. Consider the mutual exclusion protocol in Example 7.1.1. Is it possible to find a configuration such that parallel execution of jobs x_1 and x_2 (at identical speeds) never blocks, even temporarily, on lock L? It is not clear from the work automata in Figure 7.1 (or in their product automaton as, e.g., in Figure 7.3) whether such a non-blocking execution exists. Since only one process can acquire lock L, the execution that starts from the initial configuration blocks after one unit of work. However, using the geometric perspective offered by Figure 7.4(b) and the fact that a parallel execution of jobs x_1 and x_2 at identical speeds correspond to a diagonal line in this representation, it is not hard to see that any execution path below line α and above line β is non-blocking. ♢

Regions of lock-free execution paths as revealed in Example 7.2.2 are interesting: if some mechanism (e.g., higher-level semantics of the application or tailor-made scheduling) can guarantee that execution paths of an application remains contained within such lock-free regions, then their respective locks can be safely removed from the application code. With or without such locks in an application code, a scheduler cognizant of such lock-free regions can improve resource utilization and performance by regulating the execution of the application such that its execution path remains in a lock-free region.

Example 7.2.3 (Correctness)**.** Let M' be the work automaton in Example 7.1.4, and K the work automaton in Figure 7.5. We denote a configuration of M' as a tuple $(p_1, p_2, q_0, q_1, q_2)$, where $p_i \in \mathbb{R}_+$ is the state of progress of job x_i, for $i \in \{0, 1\}$, and $(q_0, q_1, q_2) \in \{-, +\} \times \{0, 1, 2\}^2$ is the state of M'. We denote a configuration of K as a tuple $(p_1, p_2, 0)$, where $p_i \in \mathbb{R}_+$ is the state of progress of job x_i, for $i \in \{0, 1\}$. The binary relation $\preceq$ over configurations of M' and

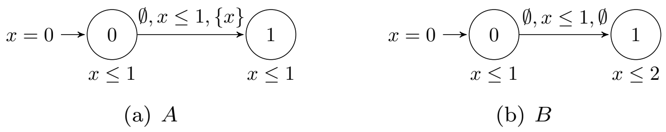


Figure 7.6: Shifting state-invariant $x \leq 1$ of state 1 in A by one unit.

K defined by $(p_1, p_2, q_0, q_1, q_2) \preceq (q_1 + p_1, q_2 + p_2, 0)$, for all $0 \leq p_i \leq 1$ and $(q_0, q_1, q_2) \in \{-, +\} \times \{0, 1, 2\}^2$, is a weak simulation of M' in K.

Note that $\preceq^{-1}$ is not a weak simulation of K in M' due to branching. Consider the configurations $s = (1, 1, -, 0, 0)$ and $s' = (0, 1, +, 1, 0)$ of M', and $t = (1, 1, 0)$ of K (cf., Figures 7.4(a) and 7.4(b)). While in configuration t job x_2 can make progress, execution of x_2 is blocked at s' because process A_1 has obtained the lock. Since $s' \preceq t$, we conclude that $\preceq^{-1}$ is not a weak simulation of K in M'.

Fortunately, we can still prove that K is a correct simplification of M by transforming $\preceq^{-1}$ into a weak simulation. Intuitively, such transformation remove pairs like $(t, s') \in \preceq^{-1}$. We make this argument formal in Section 7.2.3. ♢

As illustrated in Example 7.2.2, gluing can reveal interesting and useful properties of an application. To formalize the gluing procedure, we define two operators on work automata. The main idea is to transform a given work automaton A_1 into an equivalent automaton A_2, such that (almost) any step $(p_1, q_1) \xrightarrow{\emptyset} (p_1', q_1')$ in $[\![A_1]\!]$ corresponds with an *idling* step $(p_2, q_2) \xrightarrow{\emptyset} (p_2', q_2')$ in $[\![A_2]\!]$, i.e., a step with $p_2' = p_2$ and $q_2' = q_2$. To achieve this correspondence, we define a translation operator that ensures $p_2' = p_2$, and a contraction operator that ensures $q_2' = q_2$.

7.2.2 Translation

In this section, we define the translation operator that allows us to remove resets of jobs from transitions. The following example shows that removal of job resets can be compensated by shifting the state-invariant of the target state.

Example 7.2.4 (Shifting)**.** Suppose we remove the reset of job x on the transition of work automaton A in Figure 7.6(a). If we fire the transition at $x = a \leq 1$, then the state of progress of x in state 1 equals a instead of 0. We can correct this error by *shifting* the state-invariant of 1 by a, for every $a \leq 1$. We, therefore, transform the state-invariant of 1 into $x \leq 2$ (see Figure 7.6(b)). ♢

The transformation of work automata in Example 7.2.4 suggests a general translation procedure that, intuitively,(1) shifts each state-invariant $I(q)$, $q \in Q$, along the solutions of some job constraint $\theta(q) \in B(J)$, and(2) removes for every transition $\tau = (q, N, w, R, q')$ some resets $\rho(\tau) \subseteq J$ from R.

Definition 7.2.1 (Shifts)**.** A shift on a work automaton $(Q, P, J, I, \longrightarrow, \phi_0, q_0)$ is a tuple (θ, ρ) consisting of a map $\theta : Q \to B(J)$ and a map $\rho : \longrightarrow \to 2^J$.

We define how to shift state-invariants along the solutions of a job constraint.

Definition 7.2.2. Let $\phi, \theta \in B(J)$ be two job constraints with free variables among $\mathbf{x} = (x_1, \ldots, x_n)$, $n \geq 0$. We define the shift $\phi \uparrow \theta$ of ϕ along (the solutions of) θ as any job constraint equivalent to $\exists \mathbf{t}(\phi(\mathbf{x} - \mathbf{t}) \wedge \theta(\mathbf{t}))$.

Lemma 7.2.1. *$\uparrow$ is well-defined: for all $\phi, \theta \in B(J)$ there exists $\psi \in B(J)$ such that $\exists \mathbf{t}(\phi(\mathbf{x} - \mathbf{t}) \wedge \theta(\mathbf{t})) \equiv \psi$.*

We use a shift (θ, ρ) to translate guards and invariants along the solutions of job constraint θ and to remove resets occurring in ρ:

Definition 7.2.3 (Translation)**.** Let $\sigma = (\theta, \rho)$ be a shift on a work automaton $A = (Q, P, J, I, \rightarrow, \phi_0, q_0)$. We define the translation $A \uparrow \sigma$ of A along the shift σ as the work automaton $(Q, P, J, I_\sigma, \rightarrow_\sigma, \phi_0 \uparrow \theta(q_0), q_0)$, with $I_\sigma(q) = I(q) \uparrow \theta(q)$ and $\rightarrow_\sigma = \{(q, N, w \uparrow \theta(q), R \setminus \rho(\tau), q') \mid \tau = (q, N, w, R, q') \in \rightarrow\}$.

Lemma 7.2.2. *If $\theta \in B(J)$ has a unique solution $\delta \models \theta$, then $p + \delta \models \phi \uparrow \theta$ implies $p \models \phi$, for all $p \in \mathbb{R}_+^J$ and $\phi \in B(J)$.*

Theorem 7.2.3. *If $p \models w \wedge I(q)$ and $\delta \models \theta(q)$ implies $(p + \delta)[R \setminus \rho(\tau)] - p[R] \models \theta(q')$, for every transition $\tau = (q, N, w, R, q')$ and every $p, d \in \mathbb{R}_+^J$, then $A \preceq A \uparrow \sigma$. If, moreover, $\theta(q)$ has for every $q \in Q$ a unique solution, then $A \approx A \uparrow \sigma$.*

For at transition $\tau = (q, N, w, R, q')$, suppose $\theta(q)$ and $\theta(q')$ define unique solutions δ and δ', respectively. If σ eliminates job $x \in R$ (i.e., $x \in \rho(\tau)$), then $p(x) + \delta(x) = \delta'(x)$, for all $p \models w \wedge I(q)$. Thus, $w \wedge I(q)$ must imply $x = \delta'(x) - \delta(x)$, which seems a strong assumption. For a deterministic application, however, it makes sense to have only equalities in transition guards. In this case, a transition is enabled only when a job finishes some fixed amount of work, which corresponds to having only equalities in transition guards.

Example 7.2.5. Let M' be the work automata in Example 7.1.4, $\sigma = (\delta, \rho)$ the shift defined by $\theta(q) := x_1 = q_1 \wedge x_2 = q_2$, and $\rho(\tau) = R_\tau$. Theorem 7.2.3 shows that $M' \uparrow \sigma$ and M' are weakly bisimilar. $\Diamond$

7.2.3 Contraction

In this section, we define a contraction operator that merges different states into a single state. To determine which states merge and which stay separate, we use an equivalence relation $\sim$ on the set of states Q.

Definition 7.2.4 (Kernel)**.** A kernel of a work automaton A is an equivalence relation $\sim \subseteq Q \times Q$ on the state space Q of A.

Recall that an *equivalence class* of a state $q \in Q$ is defined as the set $[q] = \{q' \in Q \mid q \sim q'\}$ of all $q' \in Q$ related to q. The *quotient set* of Q by $\sim$ is defined as the set $Q/{\sim} = \{[q] \mid q \in Q\}$ of all equivalence classes of Q by $\sim$. By transitivity, distinct equivalence classes are disjoint and $Q/{\sim}$ partitions Q.

Definition 7.2.5 (Contraction)**.** The contraction $A/{\sim}$ of a work automaton $A = (Q, P, J, I, \rightarrow, \phi_0, q_0)$ by a kernel $\sim$ is defined as $(Q/{\sim}, P, J, I', \rightarrow', \phi_0, [q_0])$, where $\rightarrow' = \{([q], N, w, R, [q']) \mid (q, N, w, R, q') \in \rightarrow\}$ and $I'([q]) = \bigvee_{\tilde{q} \in [q]} I(\tilde{q})$.

The following results provides sufficient conditions for preservation of weak simulation by contraction. The relation $\preceq$ defined by $(p, [q]) \preceq (p, q)$, for all $(p, q) \in \mathbb{R}_+^J \times Q$, is not a weak simulation of $A/{\sim}$ in A. As indicated in Example 7.2.3, we can restrict $\preceq$ and require only $(p, [q]) \preceq (p, \alpha(p, [q]))$, for some *section* α.

Definition 7.2.6 (Section)**.** A section is a map $\alpha : \mathbb{R}_+^J \times Q/{\sim} \to Q$ such that for all $q, q' \in Q$ and $p, d \in \mathbb{R}_+^J$

1. $p \models I'([q])$ implies $p \models I(\alpha(p, [q]))$;

2. $q \sim \alpha(p, [q])$;

3. $p \models \phi_0 \wedge I(q_0)$ implies $\alpha(p, [q_0]) = q_0$;

4. $(p, [q]) \xrightarrow{N} (p', [q'])$ implies $(p, \alpha(p, [q])) \overset{N}{\Rightarrow} (p', \alpha(p', [q']))$;

5. $(p, q) \xrightarrow{d} (p + d, q)$ implies $(p, \alpha(p, [q])) \overset{d}{\Rightarrow} (p + d, \alpha(p + d, [q]))$.

In contrast with conditions (1), (2), and (3) in Definition 7.2.6, conditions (4) and (5) impose restrictions on the contraction $A/{\sim}$. These restrictions allow us to prove, with the help of the following lemma, weak simulation of $A/{\sim}$ in A.

Lemma 7.2.4. *If* $(p, [q]) \xrightarrow{d} (p+d, [q])$*, then there exist* $k \geq 1$*,* $0 = c_0 < \cdots < c_k = 1$ *and* $q_1, \ldots, q_k \in [q]$ *such that* $p + c \cdot d \models I(q_i)$*, for all* $c \in [c_{i-1}, c_i]$ *and* $1 \leq i \leq k$*.*

Theorem 7.2.5. $A \preceq A/{\sim}$*; and if there exists a section* α*, then* $A/{\sim} \preceq A$*.*

In our concluding example below, we revisit our intuitive gluing procedure motivated in Section 7.2.1 to show how the theory developed in Sections 7.2.2 and 7.2.3 formally supports our derivation of the geometric representation of $[\![K]\!]$ from $[\![M']\!]$ and implies the existence of mutual weak simulations between K and M'.

Example 7.2.6. Consider the work automaton $M' \uparrow \sigma$ from Example 7.2.5, and let $\sim$ be the kernel that relates all states of $M' \uparrow \sigma$. The contraction $(M' \uparrow \sigma)/{\sim}$ results in K, as defined in Example 7.2.1 (modulo some irrelevant idling transitions). Define $\alpha(p, [(q_1, q_2)]) = \min H$, where $H = \{(q_1, q_2) \in \{0, 1, 2\}^2 \mid p \models I_\sigma(q_1, q_2)\}$ is ordered by $(q_1, q_2) \leq (q_1', q_2')$ iff $q_1 \leq q_1'$ and $q_2 \leq q_2'$. By Theorem 7.2.5, we have $M' \preceq K$ and $M \preceq M'$. By Example 7.2.3, M' and K are not weakly bisimilar. $\diamondsuit$

The work automaton in Figure 7.3 and the geometric representation of its infinite semantics in Figure 7.4(a), only indirectly define a mutual exclusion protocol in M'. By Example 7.2.6, we conclude that M' is weakly language equivalent to a much simpler work automaton K that explicitly defines a mutual exclusion protocol by means of its state-invariant. Having such an explicit dependency visible in a state-invariant, reveals interesting behavioral properties of M', such as existence of non-blocking paths. These observations may be used to generate schedulers that force the execution to proceed along these non-blocking paths, which would enable a lock-free implementation and/or execution.

7.3 Related work

Work automata without jobs correspond to *port automata* [KC09], which is a data-agnostic variant of *constraint automata* [BSAR06]. In a constraint automaton, each synchronization constraint $N \subseteq P$ is accompanied with a data constraint that interrelates the observed data d_a, at every port $a \in N$. Although it is straightforward to extend our work automata with data constraints, we refrain from doing so because our work focuses on synchronization rather then data-aware interaction. Hiding on constraint automata defined by Baier et al. in [BSAR06] essentially combines our hiding operator in Definition 7.1.9 with contraction from Theorem 7.2.5.

The syntax of work automata is similar to the syntax of *timed automata* [AD94]. Semantically, however, timed automata are different from work automata because jobs in a work automaton may progress independently (depending on whether or not they are scheduled to run on a processor), while clocks in a timed automaton progress at identical speeds. For the same reason, work automata differ semantically from *timed constraint automata* [ABdBR04], which is introduced by Arbab et al. for the specification of time-dependent connectors.

This semantic difference suggests that we may specify a concurrent application as a *hybrid automaton* [Hen00], which can be seen as a timed automaton wherein the speed of each clock, called a *variable*, is determined by a set of first order differential equations. Instead of fixing the speed of each process beforehand, via differential equations in hybrid automata, our scheduling approach aims to determine the speed of each process only after careful analysis of the application. Therefore, we do not use hybrid automata to specify a concurrent application

Weighted automata [DKV09] constitute another popular quantitative model for concurrent applications. Transitions in a weighted automaton are labeled by a weight from a given semiring. Although weights can define the workload of transitions, weighted automata do not show dependencies among different concurrent transitions, such as mutual exclusion [vGV97]. As a consequence, weighted automata do not reveal dependencies induced by a protocol like work automata do.

A geometric perspective on concurrency has already been studied in the context of *higher dimensional automata*, introduced by Pratt [Pra91] and Van Glabbeek [vG91]. This geometric perspective has been successfully applied in [vGV97] to find and explain an essential counterexample in the study of semantic equivalences [vG06], which shows the importance of their, and indirectly our, geometric perspective. A higher dimensional automaton is a geometrical object that is constructed by gluing hypercubes. Each hypercube represents parallel execution of tasks associated with each dimension. This geometrical view on concurrency allows inheritance of standard mathematical techniques, such as homology and homotopy, which leads to new methods for studying concurrent applications [GJ92, Gun01].

7.4 Discussion

We extended work automata with state-invariants and resets and provided a formal semantics for these work automata. We defined weak simulation of work automata and presented translation and contraction operators that can simplify work automata while preserving their semantics up to weak simulation. Although translation is defined for any shift (θ, ρ), the conditions in Theorem 7.2.3 prove bisimulation only if θ has a unique solution. In the future, we want to investigate if this condition can be relaxed—and if so, at what cost—to enlarge the class of applications whose work automata can be simplified using our transformations.

Our gluing procedure in Example 7.2.1 associates a work automaton with a geometrical object, and Example 7.2.2 shows that this geometric view reveals interesting behavioral properties of the application, such as mutual exclusion and existence of non-blocking execution paths. This observation suggests our results can lead to smart scheduling that yields lock-free implementation and/or executions.

State-invariants and guards in work automata model the exact amount of work that can be performed until a job blocks. In practice, however, these exact amounts of work are usually not known before-hand. This observation suggests that the 'crisp' subset of the multidimensional real vector space defined by the state-invariant may be replaced by a density function. We leave the formalization of such stochastic work automata as future work.