The Temporal Logic of Distributed Actions

Wolfgang Reisig and Adrianna Foremniak

Humboldt-Universität zu Berlin
August 25, 2000

Abstract. We reformulate Lamport's TLA to cope with Distributed Actions.
Temporal Logic of Distributed Actions (TLDA) retains all valuable properties of TLA (such as "implementation is implication" and "composition is conjunction"). TLDA can express important properties of Distributed Systems that can not be expressed in TLA. In particular, the notion of *progress* replaces the conventional, but less adequate notion of weak fairness. Finally, we challenge Lamport's arguments about concurrent runs.

1 State Sequences

A classical way to model a discrete dynamic system A is by describing its potential *states* as mappings

$$S : var \to val \tag{1.1}$$

with *var* and *val* sets of *variables* and *values*, respectively. The *behaviors* of A then are finite or infinite sequences

$$\sigma : \quad S_0 \quad S_1 \quad S_2 \ldots \tag{1.2}$$

of states S_i.

Operational semantics of programming languages (e.g. in the SOS - style) as well as numerous specification techniques (ASMs, Statecharts, CCS, CSP, among others) adopt this notion of behaviors.

As remarked by [Lam96], this notion of behavior exhibits severe shortcomings when it comes to composition and refinement. We exemplify this by models of clocks, borrowed from [Lam96] and [Lam94]. For simplicity, we ignore the fact that a clock is supposed to tell the real time, and we instead just specify the sequence of times that it displays.

An *hour clock* displays the hours; for this purpose we assume the variable hr. A typical behavior then may be represented as $\sigma = S_0\, S_1 \ldots$ with

$$S_0(hr) = 22, \quad S_1(hr) = 23, \quad S_2(hr) = 0, \quad S_3(hr) = 1, \ldots \quad . \tag{1.3}$$

More generally, every behavior σ of an hour clock starts out with

$$S_0\,(hr) \in \{0, \ldots, 23\} \tag{1.4}$$

and iterates such that

$$S_{i+1}(hr) = suc_1\,(S_i(hr))\,, \tag{1.5}$$

where $suc_1\,(23) = 0$ and $suc_1(n) = n+1$ for $n \neq 23$. The states S_i may assign any value $S_i\,(y)$ to variables $y \neq hr$.

An *hour-minute clock* displays both the hours and the minutes; for this purpose we assume the variables hr and min, respectively. A typical behavior then was $\sigma = S_0\ S_1 \ldots$ with

$$\begin{array}{llll} S_0\,(hr) = 22, & S_1(hr) = 22, & S_2(hr) = 23, & S_3(hr) = 23, \\ S_0\,(min) = 58, & S_1(min) = 59, & S_2(min) = 0, & S_3(min) = 1. \end{array} \quad \ldots \tag{1.6}$$

More generally, every behavior of an hour-minute clock starts out with

$$\begin{array}{l} S_0\,(hr) \in \{0, \ldots, 23\} \\ S_0\,(min) \in \{0, \ldots, 59\} \end{array} \tag{1.7}$$

and iterates by

$$\begin{array}{l} S_{i+1}(hr) = S_i\,(hr) \\ S_{i+1}(min) = min + 1 \end{array} \qquad \text{if min} \neq 59\,,$$

and

$$\begin{array}{l} S_{i+1}(hr) = suc_1\,(hr) \\ S_{i+1}(min) = 0 \end{array} \qquad \text{if min} = 59\,. \tag{1.8}$$

The states S_i may assign any value $S_i\,(y)$ to variables $y \notin \{hr, min\}$.

One usually would expect an hour-minute clock to coincidently be an hour clock as well (just ignore the variable min). But the above behavior (1.8) of our hour-minute clock "stutters" the hours (i.e. $S_0(hr) = S_1(hr)$, $S_2(hr) = S_3(hr) = \cdots = S_{61}(hr)$), whereas hours steadily increase in the hour clock (cf. (1.5)). So we adopted a modelling technique that can not cope with refinement, nor with composition.

There are two proposals to overcome this problem: *stuttering sequences* and *concurrent runs.*

The central idea of *stuttering sequences* is to make each state sequence "s-tutter" : In a state sequence, every state S may equivalently be replaced by any finite sequence $SS \ldots S$. Lamport's logic TLA [Lam94] is based on this idea.

The central idea of *concurrent runs* is to respect, represent and exploit the *scope of transitions.* For example, the hour-minute clock's transition from $min = 7$ to $min = 8$ must not involve the variable hr at all: hr is not in this transition's scope. The forthcoming logic tackles this idea.

2 Steps and Concurrent Runs

2.1 Steps

A *step* U is usually constructed as a pair (S, S') of two states, the *initial state* S and the *resulting state* S'. For each variable x, the step transforms its initial value $S(x)$ into its resulting value $S'(x)$.

Here we take a closer look into the nature of this transformation; more precisely: in the way it affects variables. A variable x may not be engaged at all into the step U (in which case $S(x) = S'(x)$). If engaged, x is updated coincidently with zero or more other variables. Each set of coincidently updated variables forms a *transition* of U. Each variable is engaged in at most one transition of a step.

For example, at a reasonable clock, a step from $min = 7$ to $min = 8$ is represented by a transition t that engages the variable *min*, but not the variable *hr*. If the clock has a variable *sec* for seconds, *sec* also engages into t, with t transforming $sec = 59$ into $sec = 0$. As a second example, the step from $hr = 10$ to $hr = 11$ would be represented by a transition u that coincidently updates *min* from $min = 59$ to $min = 0$ (and likewise $sec = 59$ to $sec = 0$). The clock may be part of a larger system with more variables around, e.g. variables to switch a chime on and off , to measure temperatures and pressure, to indicate presence of messages, etc. Those variables may or may not engage into the above described transitions t or u, depending on the intended use of the clock.

To understand a distributed system, one must be aware of the involved transitions. Hence, a step U may be represented as a triple

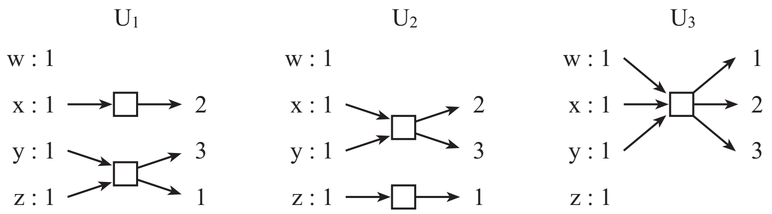
$$U = (S, S', \mathcal{T}), \tag{2.1}$$

consisting of two states $S, S' : var \to val$ (as introduced in Chapter 1), and a set $\mathcal{T}$ of pairwise disjunct sets $t_1, \ldots, t_k$, with $t_i \subseteq var$. Each $t \in \{t_1, \ldots, t_k\}$ is the *scope* of a *transition* $(s_{/t}, s'_{/t})$ (where $s_{/t}$ denotes the restriction of s to t).

States and steps will be outlined graphically in the sequel, based on the observation that usually only a few variables $x_1, \ldots, x_n$ really matter. A state S is then represented as

$$\begin{aligned} &x_1 : S(x_1) \\ &\vdots \\ &x_n : S(x_n) \end{aligned} \tag{2.2}$$

The outline of a step $U = (S, S', \mathcal{T})$ thus includes two such columns, one for S and one for S' . A transition t is represented as a box, placed in between both columns. Arrows link the box with the variables in its scope. Here are three steps, U_1 , U_2 and U_3 :



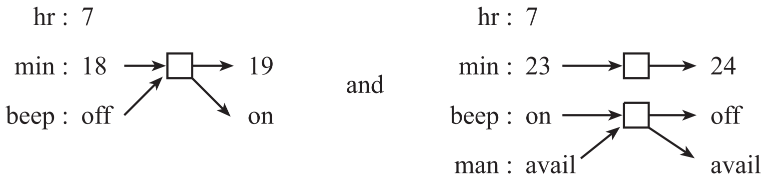
(2.3)

All three steps do with the same initial state S' (with $S\,(w) = S(x) = S(y) = S(z) = 1$) and the same resulting state S' (with $S'\,(w) = S'(z) = 1,\ S'(x) = 2,\ S'(y) = 3$) . The step U_1 has two transitions with scope $\{x\}$ and $\{y,z\}$. The step U_2 has two transitions, too, with scope $\{x,y\}$ and $\{z\}$, and U_3 has only one transition, with scope $\{w,x,y\}$.

Typical steps of a clock then were

(2.4)

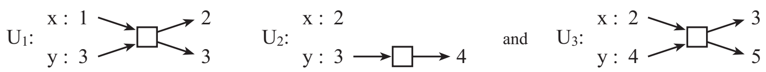
The clock may be extended with a beeper, set to start beeping at 7 : 19 and to be stopped manually. It may evolve steps such as



(2.5)

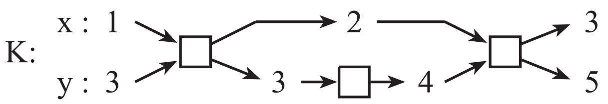
2.2 Concurrent Runs

A run of a system may be conceived as composed of steps. For example the following three steps U_1 , U_2 and U_3



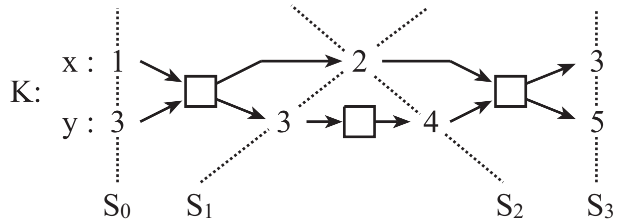
(2.6)

may be put together, yielding the concurrent run K :



(2.7)

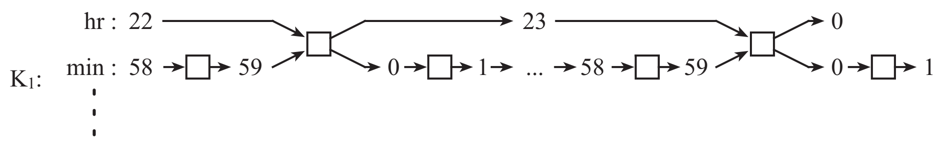
Dotted lines may help to identify global states S_i in K :



(2.8)

U_1 then leads from S_0 to S_1 , U_2 leads from S_1 to S_2 and U_3 leads from S_2 to S_3 . It will frequently turn out useful to consider the involved transitions only, and to forget about steps.

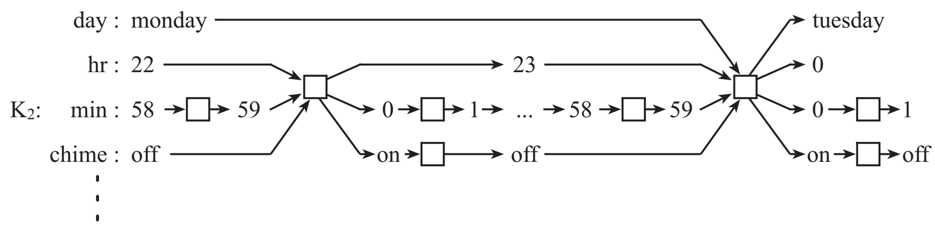
A typical example of a concurrent run is the following behavior of a hour-minute clock:



(2.9)

It starts at 22 : 58 and stucks at 0 : 01 hours.

If extended with a display for days as well as a chime to beep every hour, a typical behavior of the clock may be shaped



(2.10)

Notice the transition to switch off the chime: It is not synchronized with the minutes. It occurs sometime before the next full hour.

2.3 Technicalities

Concurrent runs such as (2.9) and (2.10) may be conceived as labelled partially ordered sets. Such a set consists of *local states* (usually represented by their label, e.g. "*min* = 1") and *transitions* (usually unlabelled, and represented as boxes). Arrows indicate direct succession in the partial order.

A transition causes new local states for the variables in its scope. Consequently, local states and transitions alternate in the partial order. Furthermore, each local state s has at most one predecessor transition (which *causes* s) and at most one successor transition (which *ceases* s). The number of predecessor states of a transition is equal to the number of its successors: one for each variable in its scope.

The elements of a concurrent run denote *occurrences* of transitions, hence the underlying set is denoted an *occurrence net.*
We recall some basics on relations, orders and functions. So, let A be a set and let $R \subseteq A \times A$ be a relation.

We usually write $a\,R\,b$ for $(a, b) \in R$. (2.11)

For $a, b \in R$ let $b\,R^{-1}\,a$ iff $a\,R\,b$. (2.12)

For $a \in A$ and $B \subseteq A$, let $R\,(a) = \{b \mid a\,R\,b\}$ and $R\,(B) = \cup\{R\,(b) \mid b \in B\}$. (2.13)

The *transitive closure* R^+ *of* R is the smallest relation such that (1) $R \subseteq R^+$ and (2) if aR^+b and $bR\,c$ then aR^+c. (2.14)

R is a *strict partial order on* A iff for all $a, b, c \in A$: (1) if $a\,R\,b$ then not $b\,R\,a$ and (2) if $a\,R\,b$ and $b\,R\,c$ then $a\,R\,c$. Strict partial orders will be called *orders* in the sequel. (2.15)

An order R on A is *Noetherean* iff, for each $a \in A$, $R^{-1}(a)$ is finite. (2.16)

$B \subseteq A$ is an *antichain of an order* R *on* A iff for all $b, c \in B : b \neq c \Rightarrow \neg\,(b\,R\,c)$. (2.17)

$B \subseteq A$ is a *suffix of* A *wrt an order* R iff there exists a maximal antichain C of A such that $B = C \cup R\,(C)$. (2.18)

For a function $f : A \to B$, the relation $\{(a, f(a) \mid a \in A\}$ is the *graph of* f. (2.19)

We are now prepared to define concurrent runs for given sets *var* and *val* of variables and values:

$K = (P, T, F, l)$ is a *concurrent run over var and val* if P and T are disjoint sets and $F \subseteq (P \times T) \cup (T \times P)$ is a relation (called *local states, transitions*, and *direct successor*, respectively), such that

$$F^{+} \text{ is a Noetherean order on } P; \tag{2.20}$$

$$\text{for each } p \in P,\ |\,F(p)\,| \leq 1 \text{ and } |\,F^{-1}(p)\,| \leq 1; \tag{2.21}$$

$$\text{for each } t \in T, |\,F(t)\,| \geq 1 \text{ and } |\,F^{-1}(t)\,| \geq 1; \tag{2.22}$$

$$\begin{array}{l} l : P \to \mathit{var} \times \mathit{val} \text{ is a function such that for each maximal} \\ \text{antichain } S \subseteq P \text{ of } F^{+},\ \text{the set } \{l(p)\,|\,p \in S\} \text{ is} \\ \text{the graph of a state } \hat{S} : \mathit{var} \to \mathit{val}. \end{array} \tag{2.23}$$

The usual graphical representation of a concurrent run $K = (P, T, F, l)$ depicts local states p, transitions t and elements (x, y) of the direct successor by their labels $l(p)$, boxes $\square$ and arrows $x \to y$, respectively. A label $l(p) = (x, n)$ is usually represented by "n" only, with x indicated at the left margin of the corresponding line. (2.9) and (2.10) give examples for this representation of concurrent runs. Transitions as defined above in fact denote transitions as introduced in 2.1.

The above definition of concurrent runs implies some important properties that will be exploited in the sequel: The minimal elements of K are local states that together form a maximal antichain; each transition can be assigned its *scope*, i.e. the variables it updates; and each maximal antichain of local states is associated its unique step:

Let $K = (P, T, F, l)$ be a concurrent run, as defined in (2.20)-(2.23). Let

$$\mathit{min}(K) := \{x \in P \cup T\,|\,F^{-1}(x) = \emptyset\}. \tag{2.24}$$

min(K) is a maximal antichain, due to (2.17).
Furthermore, $\mathit{min}(K) \subseteq P$ due to (2.21) and (2.22).

The *scope* of a transition $t \in T$ is the set of all variables that are affected by t. Formally,

$$\mathit{scope}(t) := \mathit{pr}_1(l\,(F(t))). \tag{2.25}$$

The set $\mathit{scope}(t)$ is in fact a transition as defined in Chapter 2.1. We deliberately confuse both notions, as they are very closely linked.

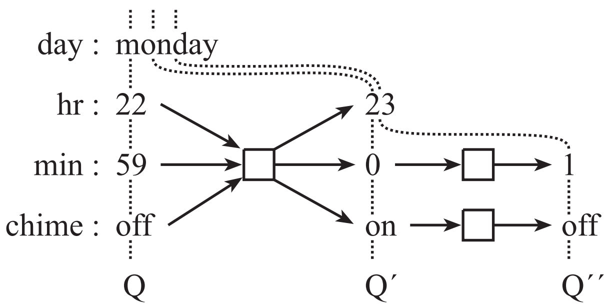
For each maximal antichain Q with $F^{-1}(t) \subseteq Q$, the set $S' := S \setminus F^{-1}(t) \cup F(t)$ is a maximal antichain, too. So, with (2.23), scope(t) could likewise have been defined by $\mathit{pr}_1\,(l\,(F^{-1}(t)))$. Furthermore, $|\,F^{-1}(t)\,| = |\,F(t)\,|$.

Each maximal antichain Q is canonically associated a step, denoted $\mathit{step}(Q)$. To define this step, for a set $Q \subseteq P$ let $T_Q := \{t \in T\,|\,F^{-1}(t) \subseteq Q\}$ and let $Q' := Q \setminus F^{-1}(T_Q) \cup F(T_Q)$. Then, for a maximal antichain Q of K,

$$step(Q) := (l\,(Q), l\,(Q'), \{scope(t) \mid t \in T_Q\}) \tag{2.26}$$

is in fact a step of K.

As an example, in the concurrent run K:



(2.27)

the dotted lines indicate some maximal antichains of local states. Q is equal to the set $min(K)$ of minimal elements of K. Let S, S' and S'' be the states $l(Q)$, $l(Q')$, $l(Q'')$, respectively. They are apparently given by

$S(day) = S'(day) = S''(day) = monday$;
$S(hr) = 22,\ S'(hr) = S''(hr) = 23$;
$S(min) = 59,\ S'(min) = 0,\ S''(min) = 1$;
$S(chime) =$ *off* , $S'(chime) =$ *on* and $S''(chime) =$ *off* .

Then the above definition (2.26) yields

step $(Q) = (\,S, S', \{\{hr, min, chime\}\})$ and
step $(Q'\,) = (\,S', S'', \{\{min\}, \{chime\}\})$

3 The Logic TLDA

We stick to the fundamental constructs of TLDA here, in analogy to "Simple TLA".

3.1 State Functions

We closely follow the line of TLA, starting out with *state functions*: A state function f is an expression built from variables, constant symbols and operation symbols, for example, $x^2 + y - 3$. We assume a corresponding algebra such that for a given valuation S of the variables, i.e. for a state S, a state function f can be evaluated, returning a value $S[\![f\,]\!]$. Intuitively, this is achieved when each occurrence in f of a variable v is replaced by $S(v)$, and all constants and operation symbols are replaced by the respective values and operations. In accordance with [Lam94] we denote this by

$$S\,[\![f]\!] \triangleq f(\forall\,'v' : s\,[\![v]\!] \,/\, v). \tag{3.1}$$

A *predicate* is a boolean-valued state function, for example, $f \triangleq x^2 = y - 3$. For a predicate f we write

$$S \models f \tag{3.2}$$

in case $S[\![f]\!] =$ true.

3.2 Actions

Actions are expressions, intended to represent steps. We follow the line of TLA, using special kinds of variables that are interpreted in a special way.

First we extend the assumed set *var* of variables: To each $x \in var$ we assume a further, "fresh" variable, x'. Furthermore, to each subset $Y \subseteq var$ we assume a "fresh" variable $\widetilde{Y}$. Variables $\widetilde{Y}$ will always take boolean values.

An *action* is any boolean valued expression with conventional variables (from *var*), primed variables and $\sim$-ed variables, as introduced above - for example, $\widetilde{\{x, y\}} \to x' + 1 = z$. Actions are to be interpreted over steps $U = (S, S', \mathcal{T})$, with variables, primed variables and $\sim$-ed variables referring to S, S' and $\mathcal{T}$, respectively: To compute the value of an action $\mathcal{A}$ at a step U, replace each occurrence of a variable v by $S(v)$ and each occurrence of v' by $S'(v)$. $\widetilde{Y}$ holds at U iff all variables are involved in one transition of U; technically: $Y \subseteq t$ for some $t \in \mathcal{T}$.

In a more formal presentation, for a set $\mathcal{T} \subseteq \wp(var)$ of sets of variables, let $\mathcal{T}^*$ be the *subset closure* of $\mathcal{T}$, i.e.

$$\mathcal{T}^* \subseteq \wp(var) \text{ such that } X \in \mathcal{T}^* \text{ iff } X \subseteq t \text{ for some } t \in \mathcal{T}. \tag{3.3}$$

Now, for an action $\mathcal{A}$ and a step $U = (S, S', \mathcal{T})$ we define

$$U \models \mathcal{A} \text{ iff} \tag{3.4}$$

$$\mathcal{A}\,(\forall\,`v`\,\forall\,`Y` : S(v) \,/\, v,\ S'(v) \,/\, v',\ Y \in \mathcal{T}^* / \widetilde{Y}\,).$$

(notice that "$Y \in \mathcal{T}^*$" denotes a boolean value). To discuss examples, we refer to the steps U_1, U_2, U_3 of (2.3). Let

$$\mathcal{A} \triangleq \quad w = 1 \ \wedge\ w' = 1 \ \wedge\ x = 1 \ \wedge\ x' = 2 \ \wedge\ y = 1 \ \wedge\ y' = 3 \ \wedge\ z = 1 \ \wedge\ z' = 1. \tag{3.5}$$

Then

$$U_1 \models \mathcal{A} \wedge \neg\, \widetilde{w} \wedge \widetilde{x} \wedge \widetilde{yz}; \quad U_2 \models \mathcal{A} \wedge \neg\, \widetilde{w} \wedge \widetilde{xy} \wedge \widetilde{z}; \quad U_3 \models \mathcal{A} \wedge \widetilde{wxy} \wedge \neg\, \widetilde{z} \tag{3.6}$$

Even more, $U_1 \models \neg \widetilde{xy} \wedge \widetilde{y} \wedge \widetilde{z}$ etc.

$$\text{A step } U \text{ is said to be } \textit{a step of } \mathcal{A} \text{ iff } U \models \mathcal{A}. \tag{3.7}$$

There are actions that do not have any step at all. Examples include e.g. $x = 0 \wedge x = 1$, $x \neq x' \wedge \neg \widetilde{x}$ or $\widetilde{xy} \wedge \neg \widetilde{x}$. There is no need however, to rule out this kind of "impossible" actions. Similarly, there are actions that have more than one step, such as $x' = x + 1$, $x' = 2 \vee x' = 3$, and any action that doesn't address all variables.

To each step U there exists a unique action $\hat{U}$ that describes U uniquely up to isomorphism (provided the assumed set *var* of variables is finite). Refering again to (2.3),

$$\begin{aligned} &\hat{U}_1 = \mathcal{A} \wedge \widetilde{x} \wedge \widetilde{y} \wedge \widetilde{z} \wedge \widetilde{yz} \wedge \neg \widetilde{Y_1} \wedge \ldots \wedge \neg \widetilde{Y_k} \\ &\text{with } \{Y_1, \ldots, Y_k\} = \wp(\mathit{var}) \setminus \{\{x\}, \{y\}, \{z\}, \{y, z\}\}. \end{aligned} \tag{3.8}$$

3.3 Temporal Formulas

Temporal formulas are built from actions, using the unary temporal *always* operator, $\Box$, and the usual propositional operators. For example, if $\mathcal{A}$ and $\mathcal{B}$ are actions, then $\neg \mathcal{A} \wedge \Box(\neg \mathcal{B})$ and $\Box(\mathcal{A} \rightarrow \Box(\mathcal{A} \vee \mathcal{B}))$ are temporal formulas.

Temporal formulas are interpreted over concurrent runs K, as introduced in 2.3. Actions $\mathcal{A}$ are interpreted in the initial step:

$$K \models \mathcal{A} \text{ iff } \mathit{step}(\mathit{min}(K)) \models \mathcal{A}. \tag{3.9}$$

The right hand side, step$(\mathit{min}(K)) \models \mathcal{A}$ is well defined by (2.26) and (3.7).

The temporal operator, $\Box$, considers all suffixes:

$$K \models \Box\mathcal{A} \text{ iff } L \models \mathcal{A} \text{ for all suffixes } L \text{ of } K. \tag{3.10}$$

Propositional operators are as to be expected:

$$K \models \neg \mathcal{A} \text{ iff not } K \models \mathcal{A} \tag{3.11}$$

$$K \models \mathcal{A} \wedge \mathcal{B} \quad \text{iff} \quad K \models \mathcal{A} \text{ and } K \models \mathcal{B} \tag{3.12}$$

As usual, we write $\Diamond p$ for $\neg\Box\neg p$ and $p \mapsto q$ for $\Box(p \rightarrow \Diamond q)$.

3.4 The *Enabled* and *Involved* Predicates

For any action $\mathcal{A}$ we define *Enabled* $\mathcal{A}$ to be the predicate that is true at a state S iff there exists a step $U = (S, S', \mathcal{T})$ such that $U \models \mathcal{A}$.

For a set X of variables, the predicate $\mathit{Involved}(\mathcal{A}, X)$ is true at a state S iff each occurrence of $\mathcal{A}$ at S requires one of the variables of X.

To define this predicate formally, we follow the line of TLA, slightly extended actions by reached variables and existence quantification.

Let $\mathcal{A}$ be an action and let e be a fresh variable. Then $\mathcal{A}(e/x)$ denotes the term $\mathcal{A}$ where each occurrence of x is replaced by an occurrence of e.
For example, let $\mathcal{A} \triangleq \widetilde{xy} \rightarrow (y = x' + 1)$.
Then

$$\mathcal{A}(c/x') \triangleq \widetilde{xy} \rightarrow (y = c + 1)$$

and

$$\mathcal{A}(C/\widetilde{xy}) \triangleq C \rightarrow (y = x' + 1).$$

We furthermore have to make explicit that tha valuation $\widetilde{Y} = \mathit{true}$ must imply $\widetilde{Z} = \mathit{true}$ for all $Z \subseteq Y$:
for two $\sim$-ed variables $\widetilde{Y}, \widetilde{Z}$ let

$$\mathit{Impl}(\widetilde{Y}, \widetilde{Z}) \triangleq \{\widetilde{Y} \rightarrow \widetilde{Z},\ \text{if } \widetilde{Y} \supseteq \widetilde{Z}$$
$$\widetilde{Z} \rightarrow \widetilde{Y},\ \text{if } \widetilde{Z} \supseteq \widetilde{Y}$$
$$\mathit{true},\ \text{otherwise}\}$$

We generalize this to

$$\mathit{Impl}(\widetilde{Y_1}, \ldots, \widetilde{Y_m}) \triangleq \bigwedge_{1 \leq i < j \leq m} \mathit{Impl}(\widetilde{Y_i}, \widetilde{Y_j})$$

The predicate *Enabled* can be defined syntactically as follows: If $v_1', \ldots, v_n'$ are the primed variables and $\widetilde{Y_1}, \ldots, \widetilde{Y_m}$ are the $\sim$-ed variables that occur in $\mathcal{A}$, then

$$\mathit{Enabled}\ \mathcal{A} \triangleq \exists\, c_1, \ldots, c_n, C_1, \ldots, C_m :$$
$$(\mathcal{A} \wedge \mathit{Impl}(\widetilde{Y_1}, \ldots, \widetilde{Y_m}))(c_1/v_1', \ldots, c_n/v_n', C_1/\widetilde{Y_1}, \ldots, C_m/\widetilde{Y_m}). \tag{3.13}$$

According to the usual meaning of quantifiers, *Enabled* $\mathcal{A}$ is true at a state S iff there exist values $c_1, \ldots, c_n$ and truth values $C_1, \ldots, C_m$ such that

$$S \models \mathcal{A}(c_1/v_1', \ldots, c_n/v_n', C_1/\widetilde{Y_1}, \ldots, C_m/\widetilde{Y_m})$$

and additionally for all $1 \leq i, j \leq m$: if $Y_i \supseteq Y_j$ then $C_i \rightarrow C_j$.

In addition, for a subset X of $\{v_1, \ldots, v_n\}$ let

$$\mathit{Involved}\ (\mathcal{A}, X) \triangleq \neg\ \text{Enabled}\ \neg(\mathcal{A} \rightarrow \bigvee_{x \in X} \widetilde{x}) \tag{3.14}$$

It may be intuitively more convenient to consider the equivalent representation

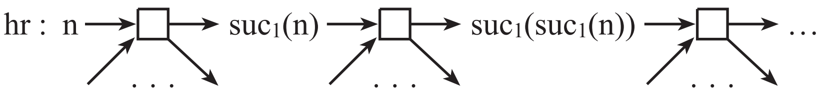
$$\mathit{Involved}\ (\mathcal{A}, X) \triangleq \forall c_1 \ldots c_n C_1 \ldots C_m :$$
$$(\mathcal{A} \rightarrow \bigvee_{x \in X} \widetilde{x} \wedge \mathit{Impl}(\widetilde{Y_1}, \ldots, \widetilde{Y_m}))(c_1/v_1', \ldots, c_n/v_n', C_1/\widetilde{Y_1}, \ldots, C_m/\widetilde{Y_m})$$

3.5 Specifications with TLDA

TLDA formulas can now be used to *specify* systems, i.e. sets of concurrent runs: A concurrent run K belongs to a system specified by a TLDA-formula $\mathcal{A}$ if $K \vDash \mathcal{A}$. As an example, we specify hour clocks by

$$HR_1 \triangleq hr \in \{0, \ldots, 23\} \wedge \Box\, (\widetilde{hr} \rightarrow hr' = suc_1(hr)) \tag{3.15}$$

Hence, a system is an hour clock if HR_1 holds for all its runs. This means that each run of an hour clock includes a substructure shaped



(3.16)

with $n \in \{0, \ldots, 23\}$. For example, with $n = 22$ (3.6) is part of both K_1 and K_2 in (2.9) and (2.10). Hence, $K_1 \vDash HR_1$ as well as $K_2 \vDash HR_1$. Notice that this specification allows clocks to stuck. Specification of perfect, never stucking clocks is postponed to Chapter 4.

As a further example, minute clocks are specified by

$$MIN_1 \triangleq min \in \{0, \ldots, 59\} \wedge \Box\, (\widetilde{min} \rightarrow min' = suc_2(min)). \tag{3.17}$$

Consequently, $K_1 \vDash MIN_1$ and $K_2 \vDash MIN_1$.

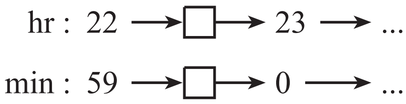
Finally, hour-minute clocks are specified by

$$HR{-}MIN_1 \triangleq HR_1 \;\wedge\; MIN_1 \;\wedge\; SYNC_1 \tag{3.18}$$

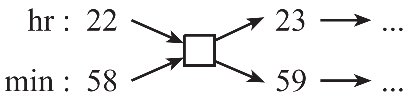
where

$$SYNC_1 \triangleq \Box\, (\widetilde{hr} \rightarrow \widetilde{hr\; min}) \;\wedge\; \Box\, (min = 59 \longleftrightarrow \widetilde{hr\; min}) \tag{3.19}$$

$SYNC_1$ excludes unwanted steps such as

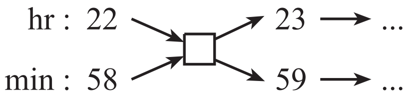


(3.20)



(3.21)

as well as



(3.22)

In fact, $HR{-}MIN_1$ holds for both runs K_1 and K_2 of (2.9) and (2.10), respectively. Hence, they both represent behaviors of hour-minute clocks (as well as hour clocks and minute clocks).

4 Progress and Fairness

Systems are usually assumed to *proceed*: It goes without saying that execution of a PASCAL program doesn't stall at an executable statement. This likewise applies to distributed systems. For example, to describe a perfect, never stuck minute clock, we rule out runs that stall with the enabled transition $min' = min + 1$: We assume *progress* for this transition. Details follow in Chapter 4.1.

Interleaving based formalisms can not adequately cope with progress. Such formalisms apply the stronger notion of *weak fairness* instead.

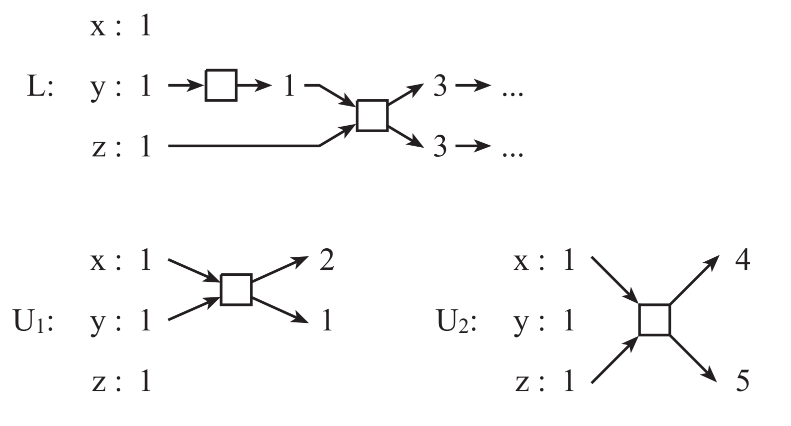
Nondeterministic systems may evolve runs that decide alternatives in an unfair way: A run may infinitely often reach a state with two (or more) *alternatively* enabled transitions, i.e. transitions that compete for some scarce resource. A run that always decides in favor of the same transition (up to finitely many exceptions) is said to behave *unfair* to the neglected transitions. The assumption of *fairness* rules out this kind of runs. Details follow in Chapter 4.2.

Interleaving based formalisms mimic progress with a "weak" form of fairness: Each interleaving sequence superfluously establishes an order on concurrent transitions. Selection of this order is then confused with nondeterministic choice.

4.1 Progress

A concurrent run K *respects progress of an action* $\mathcal{A}$ iff for all suffixes L of K, all valuations $S' : var \to val$ and all $\mathcal{T} \subseteq \wp(var)$ holds: If $\mathcal{A}$ is enabled at the initial state of L, i.e. if there exists a step $U = (min(L), S', \mathcal{T})$ with $U \vDash \mathcal{A}$, then L can not be extended by the step U, i.e. there exists some $x \in var$ with $x \in \bigcup_{t \in \mathcal{T}} t$ and $L \vDash \Diamond \widetilde{x}$. x is called a *witness* for U.

For example, with



(4.1)

L can not be extended by U_1 because of witness y, and likewise not by U_2 because of witness z. Let $\mathcal{A} = (x' = 2 \wedge y' = 1 \wedge \widetilde{xy}) \vee (x' = 4 \wedge z' = 5 \wedge \widetilde{xz})$. Then $U_1 \vDash \mathcal{A}$ and $U_2 \vDash \mathcal{A}$, and apparently, L respects progress of $\mathcal{A}$.

K violates progress of $\mathcal{A}$ iff K does not respect progress of $\mathcal{A}$. For example, the above run L violates progress of $\mathcal{A} = (x = 1 \wedge x' = 2 \wedge \widetilde{x})$.

In the above definition there may be infinitely many valuations S' and $\mathcal{T}$, hence infinitely many steps U. But finitely many witnesses suffice: To each witness not occuring in $\mathcal{A}$ there must exist one in $\mathcal{A}$. Hence the above definition is equivalent to the following:

> A concurrent run K respects progress of an action $\mathcal{A}$ iff for all suffixes L of K holds: There exists a finite subset $X \subseteq var$ of variables such that for all $S' : var \to val$ and all $\mathcal{T} \subseteq \wp(var)$ holds: if $(min(L), S', \mathcal{T}) \vDash \mathcal{A}$ then $X \cap (\bigcup_{t \in \mathcal{T}} t) \neq \emptyset$ and $L \vDash \bigwedge_{x \in X} \Diamond \widetilde{x}$.

Now, let X be any subset of variables occurring in $\mathcal{A}$. Using (3.14) let

$$\mathcal{A}_X \triangleq Involved(\mathcal{A}, X) \wedge \bigwedge_{x \in X} \Diamond \widetilde{x}. \tag{4.2}$$

Then a concurrent run K respects progress of an action $\mathcal{A}$ iff to each suffix L of K there exists a finite set $X \subseteq var$ such that $L \vDash \mathcal{A}_X$.
Finally, let $\{X_1, \ldots, X_l\} = \wp(Z)$ with Z the variables occurring in $\mathcal{A}$. Let

$$progress\ (\mathcal{A}) \triangleq \Box(\bigvee_{i=1}^{l} \mathcal{A}_{X_i}).$$

Then a concurrent run K respects progress of $\mathcal{A}$ iff

$$K \vDash\ progress\ (\mathcal{A}).$$

Progress properties can usually be specified by very simple formulas shaped

$$progress\,(\widetilde{x}) \tag{4.3}$$

for program variables x. (4.3) then reduces to

$$\Box \Diamond \widetilde{x}. \tag{4.4}$$

For example, we turn back to the clocks of Chapter 3.5. Perfect, never stucking clocks can easily be specified by the additional requirement of progress for the variables *hr* and *min*. Here the progressing clocks:

$$\mathrm{HR}_2 \triangleq \mathrm{HR}_1 \wedge progress\,(hr). \tag{4.5}$$

$$\mathrm{MIN}_2 \triangleq \mathrm{MIN}_1 \wedge progress\,(min). \tag{4.6}$$

$$\mathrm{HR{-}MIN}_2 \triangleq \mathrm{HR}_2 \wedge \mathrm{MIN}_2 \wedge \mathrm{SYNC}_1. \tag{4.7}$$

In the last definition, HR_2 may even be replaced by HR_1 because $progress\,(hr)$ follows from $progress\,(min)$ and SYNC_1.

4.2 Fairness

A fundamentally different notion is *fairness*. Fairness is crucial when a step U infinitely often competes with other steps for some resource, and U is never granted. The corresponding formula is

$$fair(\mathcal{A}) \triangleq \Box\Diamond\mathcal{A} \vee \Diamond\Box\neg Enabled\,\mathcal{A}$$

5 The Expressive Power of TLDA

TLDA is more expressive than any formalism based on sequences of states, such as TLA.

5.1 An example

As an example consider a system Σ with the two concurrent runs of Fig.1.(⋆) These runs are in fact the behaviors of a "real" system: Figs.2 and 3 represent Σ as a high-level Petri net and a low level Petri net, respectively.

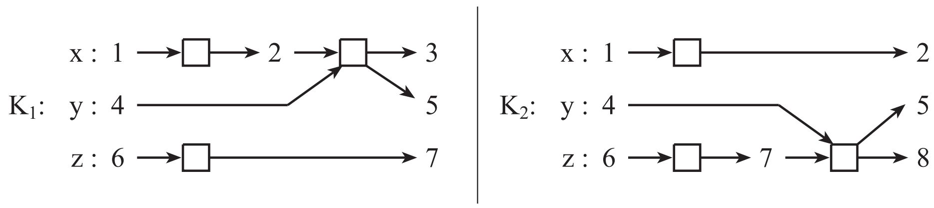


Fig. 1. Two concurrent runs

As an intuitive interpretation of Σ, conceive x and z as two agents, coincidently starting operation. In case they jointly reach state S_0 with $S_0(x) = 2$, $S_0(y) = 4$ and $S_0(z) = 7$ they compete for a common ressource (represented by "$y = 4$"). Choice is decided nondeterministically in S_0, resulting in two different runs, K_1 and K_2.

In case state S_1 with $S_1(x) = 3$, $S_1(y) = 5$ and $S_1(z) = 6$ is eventually taken in K_1, state S_0 is apparently never taken during the course of S_1. This likewise applies in K_2 to state S_2 with $S_2(x) = 1$, $S_2(y) = 5$ and $S_2(z) = 8$. Hence, in all behaviors of Σ, it remains unspecified whether S_0 is eventually reached, and whether nondeterministic choice had to be decided at all.

Systems with this kind of unspecified choice are more difficult to model, to analyse and to implement than systems with "objective" nondeterminism.

(⋆) We owe this example to Amir Pnueli [Pnu92].

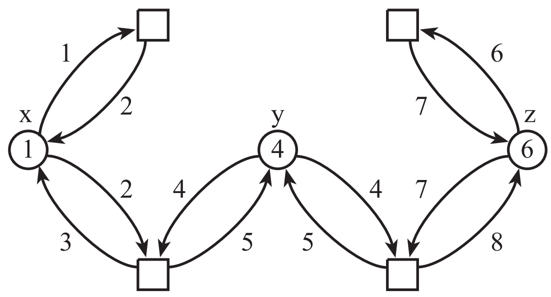


Fig. 2. High level Petri net with behaviors as in Fig.1

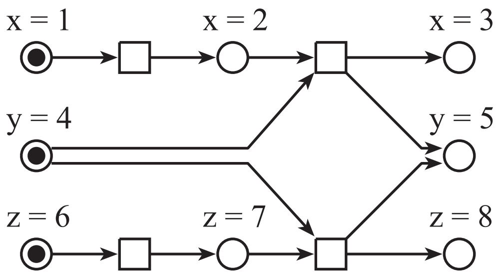


Fig. 3. Low level Petri net with behaviors of Fig.1

It is easy to express in TLDA that one of the states S_1 or S_2 was reachable in either run of Σ.

From

$$\begin{aligned} \text{init} &\triangleq x = 1 \wedge y = 4 \wedge z = 6, \\ p_1 &\triangleq x = 3 \wedge y = 5 \wedge z = 6, \\ p_2 &\triangleq x = 1 \wedge y = 5 \wedge z = 8 \end{aligned} \tag{5.1}$$

and

$$p_0 \triangleq \text{init} \mapsto (p_1 \vee p_2)$$

follows $K_1 \models p_0$ as well as $K_2 \models p_0$.

For TLDA formula p define

$\Sigma \models p$ iff $K \models p$ for each occurrence run K of Σ.

Then $\Sigma \models p_0$.

As a variant consider a system Σ', exhibiting six concurrent runs $K_1{}', \ldots, K_6{}'$, as in Fig.4. These runs are likewise the behaviors of a "real" system: Fig.5 and 6 represent Σ' as a high-level Petri net and a low-level Petri net, respectively.

The above formula p_0 does not hold in Σ'; it doesn't hold e.g. in $K_1{}'$.

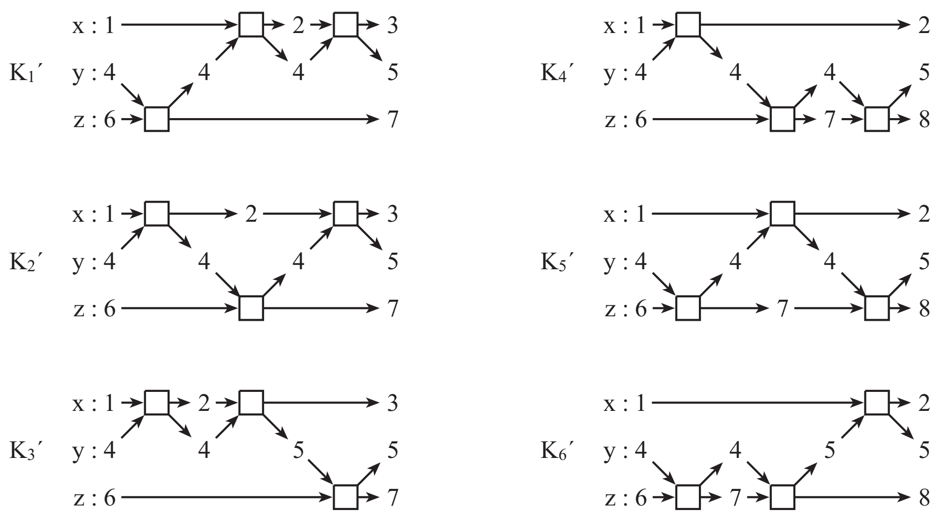


Fig. 4. The six behaviors of Σ'

A canonical procedure turns concurrent runs into state sequences: Extend the partial order of transitions to total orders. The resulting state sequences are *consistent with* the concurrent run. Fig.7 shows the result of this procedure for

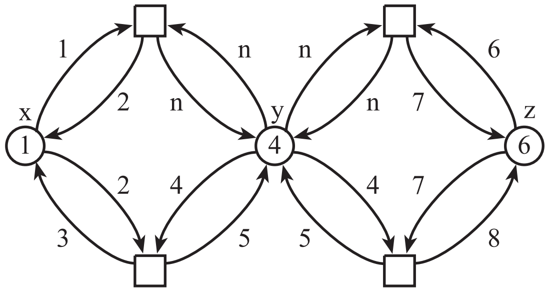


Fig. 5. High-level Petri Net with behaviors of Fig.4

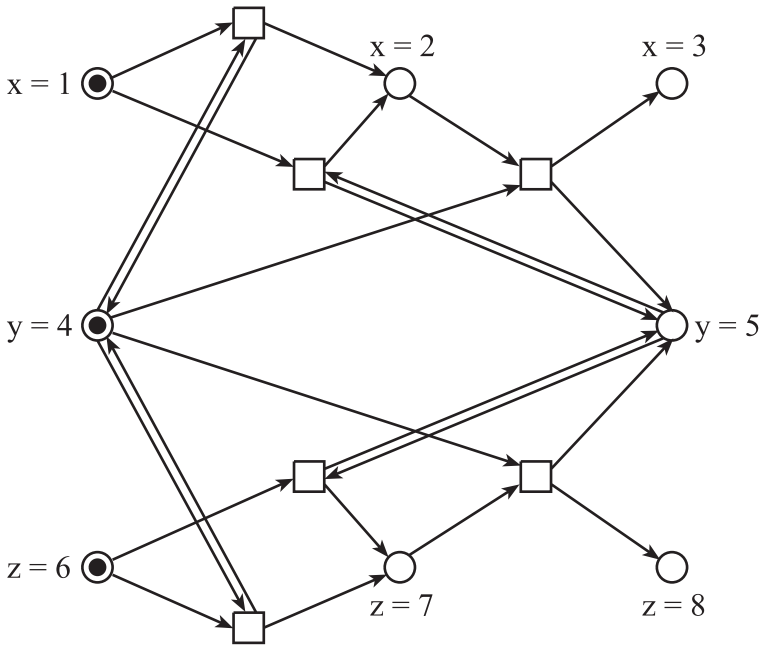


Fig. 6. Low level Petri Net with behavior of Fig.4

the two concurrent runs of Fig.1, i.e. for system Σ. The same six state sequences results from the concurrent runs ${K_1}', \ldots, {K_6}'$ of Fig.4, i.e. for Σ'.

So we end up with a situation of two systems, Σ and Σ', that can very well be distinguished with TLDA-formulas. As both systems yield the same state sequences, their different behavior can not be formulated in terms of state sequences. So, it can not be formulated in TLA.

$$
\begin{array}{ccccc c ccccc}
\multicolumn{5}{c}{\text{State sequences of } K_1} & \qquad & \multicolumn{5}{c}{\text{State sequences of } K_2} \\
\\
x & 1 & 1 & 2 & 3 & & x & 1 & 2 & 2 & 2 \\
y & 4 & 4 & 4 & 5 & & y & 4 & 4 & 4 & 5 \\
z & 6 & 7 & 7 & 7 & & z & 6 & 6 & 7 & 8 \\
\\
x & 1 & 2 & 2 & 3 & & x & 1 & 1 & 2 & 2 \\
y & 4 & 4 & 4 & 5 & & y & 4 & 4 & 4 & 5 \\
z & 6 & 6 & 7 & 7 & & z & 6 & 7 & 7 & 8 \\
\\
x & 1 & 2 & 3 & 3 & & x & 1 & 1 & 1 & 2 \\
y & 4 & 4 & 5 & 5 & & y & 4 & 4 & 5 & 5 \\
z & 6 & 6 & 6 & 7 & & z & 6 & 7 & 8 & 8 \\
 & & & \uparrow & & & & & & \uparrow & \\
 & & & p_1 & & & & & & p_2 &
\end{array}
$$

Fig. 7.

The above systems Σ and Σ' can apparently not be specified in TLA – hence it comes without surprise that TLA can not cope with properties such as init $\mapsto (p_1 \vee p_2)$.

With init as in (5.1), TLDA would specify Σ as

$$
\begin{array}{lll}
\text{init} & & \\
\wedge & \square & x = 1 \rightarrow (x' = 2 \wedge \neg \widetilde{y} \wedge \neg \widetilde{z}) \\
\wedge & \square & x = 2 \rightarrow (\neg \widetilde{x} \vee (x' = 3 \wedge \widetilde{xy} \wedge y = 4 \wedge y' = 5 \wedge \neg \widetilde{z})) \\
\wedge & \square & y = 4 \rightarrow (\neg \widetilde{y} \vee (y' = 5 \wedge ((\widetilde{xy} \wedge x = 2 \wedge x' = 3 \wedge \neg \widetilde{z}) \vee \\
 & & \qquad\qquad\qquad\qquad (\widetilde{yz} \wedge z = 7 \wedge z' = 8 \wedge \neg \widetilde{x})))) \\
\wedge & \square & z = 6 \rightarrow (z' = 7 \wedge \neg \widetilde{x} \wedge \neg \widetilde{y}) \\
\wedge & \square & z = 7 \rightarrow (\neg \widetilde{z} \vee (z' = 8 \wedge \widetilde{yz} \wedge y = 4 \wedge y' = 5 \wedge \neg \widetilde{x})).
\end{array}
\tag{5.2}
$$

A TLDA-specification of Σ' would be as follow (the second and the last line are identical in (5.2) and (5.3)):

$$
\begin{array}{lll}
\text{init} & & \\
\wedge & \Box & x=1 \rightarrow (\neg\widetilde{x} \ \vee \ (x'=2 \ \wedge \ \widetilde{xy} \ \wedge \ y'=y \ \wedge \ \neg\widetilde{z}) \\
\wedge & \Box & x=2 \rightarrow (\neg\widetilde{x} \ \vee \ (x'=3 \ \wedge \ \widetilde{xy} \ \wedge \ y=4 \ \wedge \ y'=5 \ \wedge \ \neg\widetilde{z})) \\
\wedge & \Box & y=4 \rightarrow (y'=4 \ \wedge \ ((\widetilde{xy} \ \wedge \ x=1 \ \wedge \ x'=2 \ \wedge \ \neg\widetilde{z}) \vee \\
& & \qquad\qquad\qquad (\widetilde{yz} \ \wedge \ z=6 \ \wedge \ z'=7 \ \wedge \ \neg\widetilde{x}))) \\
& & \qquad\quad \vee (y'=5 \ \wedge \ ((\widetilde{xy} \ \wedge \ x=2) \vee (\widetilde{yz} \ \wedge \ z=7))) \\
\wedge & \Box & z=6 \rightarrow (\neg\widetilde{z} \ \vee \ (z'=7 \ \wedge \ \widetilde{yz} \ \wedge \ y'=y \ \wedge \ \neg\widetilde{x}) \\
\wedge & \Box & z=7 \rightarrow (\neg\widetilde{z} \ \vee \ (z'=8 \ \wedge \ \widetilde{yz} \ \wedge \ y=4 \ \wedge \ y'=5 \ \wedge \ \neg\widetilde{x})).
\end{array}
\tag{5.3}
$$

5.2 Wrong arguments about Concurrent Runs

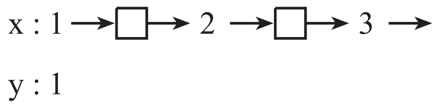
In the following quotes from literature, the terms "partial ordering" denotes what we call "concurrent run", and "interleaving" stand for "state sequence". Quoting [Lam94], "For reasoning about safety and liveness properties, partial ordering are completely equivalent to interleavings. In an interleaving model, a partial ordering of events corresponds to the set of total orderings consistent with it". This argument is wrong: The interleavings (state sequences) of a system usually stem from a *set* of partial orderings, not from a single one. This set of partial orderings can in general not be derived from the set of their interleavings. For example, the state sequences of Fig.7 may stem from the two concurrent runs of Fig.1, or from the six concurrent runs of Fig.4. Following [Lam94] "Asserting in a partial-ordering model that a property holds for all possible partial orderings of events is equivalent to asserting in an interleaving model that it holds for all possible sequences of events". This argument is wrong: The above discussed property init $\mapsto (p_1 \vee p_2)$ holds for all concurrent runs of Σ, but not for all state sequences that are consistent with them. We state this property more precisely in terms of state sequences: To each concurrent run of Σ there exists a consistent state sequence where the property holds.

Finally, Lamport concludes "Thus, a partial ordering semantics is not needed for reasoning about safety and liveness". This argument is wrong. Concurrent runs are the inevitable basis to describe many safety properties. The simplest of those properties reads "each concurrent run of Σ has a consistent state sequence with property p". The above property, init $\mapsto (p_1 \vee p_2)$, has this nature. Concurrent runs are likewise inevitable for many liveness properties. Most prominently, the property of *progress* as discussed in Chapter 4.

For example, the state sequence

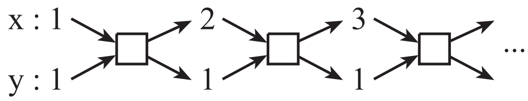
$$
\begin{array}{lcccc}
x: & 1 & 2 & 3 & 4 \\
y: & 1 & 1 & 1 & 1
\end{array}
\ \ldots
\tag{5.4}
$$

violates weak fairness for the transition $y := 2$. Likewise, the concurrent run



(5.5)

violates progress for $y : 1 \rightarrow \square \rightarrow 2$.
However, the concurrent run



(5.6)

respects progress for $y : 1 \rightarrow \square \rightarrow 2$.

Both, (5.5) and (5.6) evolve the same -unique- state sequence, (5.4). Hence, the notion of progress, intuitively most obvious, can not be represented by state sequences.

5.3 The notion of observation

A widespread argument states that concurrent runs and concurrency based properties such as confused nondeterminism or progress, were not observable. Assuming this a valid argument, observation, then, is an equivalence on behaviors. One then must be able to describe the different behaviors that then are equivalent under observation.

Observation is apparently not compatible with all relevant properties of concurrent systems. Apparently, it is not compatible with confused nondeterminism and with progress. Future insight into the basics and essentials of concurrent and distributed systems most likely will reveal further such properties.

References

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[Lam96] Leslie Lamport. Refinement in state-based formalisms. *SRC Technical Note 1996-001*, December 1996.

[Lam97] Leslie Lamport. Composition: A way to make proofs harder. In W.P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference*, LNCS 1536, pages 402–423. September 1997.

[Pnu92] Amir Pnueli. Private Communication, Dagstuhl, 1992.