Modal and Temporal Logics for Processes

Colin Stirling*
Dept. of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ, UK
email: cps@uk.ac.ed.dcs

Preface

In the following we examine modal and temporal logics for processes. First in section 1 we introduce concurrent processes as terms of an algebraic language comprising a few basic operators (as developed by Milner, Hoare and others). Their behaviours are described using transitions. Families of transitions can be arranged as labelled graphs, concrete summaries of the behaviour of processes. Various combinations of processes and their resulting behaviour as determined by the transition rules are reviewed.

In section 2 a simple modal logic, a slight extension of Hennessy-Milner logic, is introduced for describing the capabilities of processes. An important discussion is when two processes may be deemed, for all practical purposes, to have the same behaviour. We discuss bisimulation equivalence as the discriminating power of the modal logic is tied to it. This equivalence is initially presented in terms of games.

More generally practitioners have found it useful to be able to express temporal properties (such as liveness and safety) of concurrent systems. A logic expressing temporal notions provides a framework for the precise formalization of such specifications. Formulas of the modal logic are not rich enough to express such temporal properties. So extra operators, extremal fixed points, are added in section 3. The result is a very expressive temporal logic, (a slight extension of) modal mu-calculus.

The modal and temporal logics provide a repository of useful properties. However it is also very important to be able to verify that an agent has or lacks a particular property. This is the topic of section 4. First we show that this property checking can be understood in terms of game playing. We then present sound and complete tableau proof systems for proving temporal properties of processes. The proof method is illustrated on examples.

Acknowledgements Earlier versions of these notes have been presented previously at the 4th European Summer School in Logic, Language, and Information, University of Essex, 1992; at the Tempus Summer School for Algebraic and Categorical Methods in Computer Science, Masaryk University, Brno, 1993; and the

* Research in part supported by EPSRC grant GR/H16056 and ESPRIT BRA project Concur2.

Summer School in Logic Methods in Concurrency, Aarhus University, 1993. I would like to thank the organisers of these summerschools and workshops, and of The Banff higher order workshop. I would like to thank Julian Bradfield for use of his Tex tree constructor for building derivation trees, and Faron Moller and David Walker for comments on earlier drafts.

1 Processes

Process theory introduces processes as terms of an algebraic language comprising a few basic operators. Transitions of the form $E \xrightarrow{a} F$, that process E may become F by performing (participating in, or accepting) the action a, feature prominently underpinning the behavioural meaning of a process. Structured rules guide their derivation in the sense that the transitions from a compound process are determined by those from its components. Families of transitions can be arranged as labelled graphs, concrete summaries of the behaviour of processes. Here we first review various combinations of processes and their resulting behaviour as determined by the transition rules.

1.1 First examples

A simple example of a process (courtesy of Hoare [48]) is a clock that perpetually ticks: $Cl \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl$. We adopt the usual convention that names of actions, or events, such as $\mathtt{tick}$ are in lower case whereas names of processes have an initial capital letter. The defining expression for Cl invokes a prefix operator . which forms the process $a.E$ from the action a and the process E. The facility for defining processes $\stackrel{\mathrm{def}}{=}$, relating a process name with a process expression, is recursive as both occurrences of Cl name the same process. The behaviour of Cl is very elementary: it can only perform the action $\mathtt{tick}$ and in so doing becomes Cl again. This is deducible from the rules for transitions.

First is the rule or axiom for the prefix operator when a is any action and E any process.

$$\boxed{\mathrm{R}(.) \quad a.E \xrightarrow{a} E}$$

Its meaning is that process $a.E$ may *perform* (respond to, participate in, or accept) the action a and evolve into the process E. An instance is the transition: $\mathtt{tick}.Cl \xrightarrow{\mathtt{tick}} Cl$. Next is the transition rule for $\stackrel{\mathrm{def}}{=}$, which is presented as a *goal directed* inference rule.

$$\boxed{\mathrm{R}(\stackrel{\mathrm{def}}{=}) \quad \frac{P \xrightarrow{a} F}{E \xrightarrow{a} F} \; P \stackrel{\mathrm{def}}{=} E}$$

Provided that E may become F by performing a and that the side condition $P \stackrel{\mathrm{def}}{=} E$ is fulfilled, it follows that $P \xrightarrow{a} F$.

Using these two rules we can show $Cl \xrightarrow{\texttt{tick}} Cl$ as follows:

$$\frac{Cl \xrightarrow{\texttt{tick}} Cl}{\texttt{tick}.Cl \xrightarrow{\texttt{tick}} Cl} \; Cl \stackrel{\mathrm{def}}{=} \texttt{tick}.Cl$$

This *proof* of the transition $Cl \xrightarrow{\texttt{tick}} Cl$ is presented with the desired conclusion as antecedent which follows from the axiom instance beneath it via an application of R($\stackrel{\mathrm{def}}{=}$). These two rules do not permit other derivable transitions from Cl.

The behaviour of Cl can be visually summarized as a transition graph as in figure 1. The ingredients of this graph (called a *labelled transition system*) are

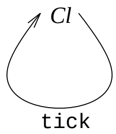


Fig. 1. A transition graph for Cl

process terms and directed labelled arcs between them. Each node is a process term, and one of the nodes is Cl (which can be thought of as the root). All the possible transitions from each node, those that are provable from the rules for transitions, are represented.

A second example, a very simple vending machine, is defined in figure 2. Here + (which has wider scope than .) is the choice operator from Milner's

$$\begin{aligned} Ven &\stackrel{\mathrm{def}}{=} \texttt{2p}.Ven_b + \texttt{1p}.Ven_l \\ Ven_b &\stackrel{\mathrm{def}}{=} \texttt{big}.\texttt{collect}_b.Ven \\ Ven_l &\stackrel{\mathrm{def}}{=} \texttt{little}.\texttt{collect}_l.Ven \end{aligned}$$

Fig. 2. A vending machine

CCS, Calculus of Communicating Systems, [61]. Initially Ven may accept a `2p` or `1p` coin; then a button, `big` or `little`, may be depressed depending on which coin was deposited; finally, after an item is collected the process reverts to its initial state Ven. Transition rules for +, justifying this description of Ven, are:

$$\boxed{\mathrm{R}(+) \quad \frac{E_1 + E_2 \xrightarrow{a} F}{E_1 \xrightarrow{a} F} \qquad \frac{E_1 + E_2 \xrightarrow{a} F}{E_2 \xrightarrow{a} F}}$$

The proof of the transition $Ven \xrightarrow{2p} Ven_b$ is[2]:

$$\dfrac{\dfrac{2\mathtt{p}.\,Ven_b \xrightarrow{2\mathrm{p}} Ven_b}{2\mathtt{p}.\,Ven_b + 1\mathtt{p}.\,Ven_l \xrightarrow{2\mathrm{p}} Ven_b}}{Ven \xrightarrow{2\mathrm{p}} Ven_b}$$

The final transition is an axiom instance; an application of the first R(+) rule to it yields the intermediate transition; and the goal therefore follows using the rule R($\stackrel{\mathrm{def}}{=}$). The transition graph for Ven is presented in figure 3.

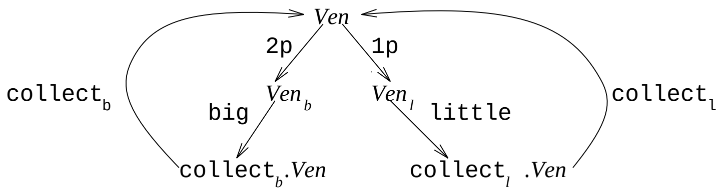


Fig. 3. A transition graph for Ven

A transition $E \xrightarrow{a} F$ can be viewed as an assertion which is derivable from the rules for transitions. To find out what transitions are possible from E it suffices to examine its main combinator and the possible transitions of its components. There is an analogy with rules for expression evaluation: for instance, to evaluate $(3 \times 2) + 4$ it suffices to evaluate the components 3×2 and 4, and then sum their values. Such families of rules give rise to a structured operational semantics in the style of [71]. However whereas the essence of an expression is to be evaluated the essence of a process is to *act*.

Families of processes can be defined using indexing. A simple case is the set of counters $\{Ct_i : i \in \mathbb{N}\}$ of figure 4. The counter Ct_3 can increase to

$$\begin{aligned} Ct_0 &\stackrel{\mathrm{def}}{=} \mathtt{up}.\,Ct_1 + \mathtt{round}.\,Ct_0 \\ Ct_{i+1} &\stackrel{\mathrm{def}}{=} \mathtt{up}.\,Ct_{i+2} + \mathtt{down}.\,Ct_i \end{aligned}$$

Fig. 4. A family of counters

Ct_4 by performing `up`, or decrease to Ct_2 by performing `down`. Each member Ct_i determines the same transition graph which contains an infinite number of

[2] In proofs of transitions we usually omit explicit mention of side conditions in the application of a rule such as R($\stackrel{\mathrm{def}}{=}$).

different nodes: such graphs are called *infinite state* graphs in contrast to the finite state graphs of figures 1 and 3.

The operator $+$ is frequently extended to indexed families $\sum\{E_i : i \in I\}$ where I is a set of indices.

$$\boxed{\mathrm{R}(\textstyle\sum) \quad \dfrac{\sum\{E_i : i \in I\} \stackrel{a}{\longrightarrow} F}{E_j \stackrel{a}{\longrightarrow} F} \; j \in I}$$

A special case is when the indexing set I is empty. By the rule $\mathrm{R}(\sum)$ such a process has no transitions as the subgoal can never be fulfilled. In CCS this nil process $\sum\{E_i : i \in \emptyset\}$ is abbreviated to $\mathbf{0}$ (and to STOP in Hoare's CSP, Communicating Sequential Processes, [48]). Thus the clock $\mathtt{tick}.\mathbf{0}$ can only do a single tick before terminating.

Actions can be viewed as ports or channels, means by which processes can interact. It is then also important to consider the passage of data between processes along these channels or through these ports. In CCS input of data at a port named a is represented by the prefix $a(x).E$ where $a(x)$ binds free occurrences of x in E. (In CSP $a(x)$ is written $a?x$.) Now a no longer names a single action but instead represents the set $\{a(v) : v \in D\}$ where D is the appropriate family of data values. The transition axiom for this prefix input form is:

$$\boxed{\mathrm{R}(\mathrm{in}) \quad a(x).E \stackrel{a(v)}{\longrightarrow} E\{v/x\} \quad v \in D}$$

where $E\{v/x\}$ is the process term which is the result of replacing all free occurrences of x in E with v[3]. Output at a port named a is represented in CCS by the prefix $\overline{a}(e).E$ where e is a data expression. The overbar $^{-}$ symbolizes output at the named port. (In CSP $\overline{a}(e)$ is written $a!e$.) The transition rule for this prefixing depends on extra machinery for expression evaluation. Assume that $\mathrm{Val}(e)$ is the data value in D (if there is one) which e evaluates to:

$$\boxed{\mathrm{R}(\mathrm{out}) \quad \overline{a}(e).E \stackrel{\overline{a}(v)}{\longrightarrow} E \quad \mathrm{Val}(e) = v}$$

The asymmetry between input and output is illustrated in the following process that copies a value from `in` and then sends it through `out`:

$$Cop \stackrel{\mathrm{def}}{=} \mathtt{in}(x).\overline{\mathtt{out}}(x).Cop$$

For any $v \in D$ the transition $Cop \stackrel{\mathtt{in}(v)}{\longrightarrow} \overline{\mathtt{out}}(v).Cop$ is derived as follows:

$$\frac{Cop \stackrel{\mathtt{in}(v)}{\longrightarrow} \overline{\mathtt{out}}(v).Cop}{\mathtt{in}(x).\overline{\mathtt{out}}(x).Cop \stackrel{\mathtt{in}(v)}{\longrightarrow} \overline{\mathtt{out}}(v).Cop}$$

[3] The process $a(x).E$ can be viewed as an abbreviation of the process $\sum\{a_v.E\{v/x\} : v \in D\}$, writing a_v instead of $a(v)$.

The subgoal is an instance of R(in), as $(\overline{\texttt{out}}(x).Cop)\{v/x\}$ is $\overline{\texttt{out}}(v).Cop$. This latter process has only one possible transition, $\overline{\texttt{out}}(v).Cop \xrightarrow{\overline{\texttt{out}}(v)} Cop$, an instance of R(out) as we assume that Val(v) is v. Whenever Cop inputs a value at `in` it immediately disgorges it through `out`. The size of the transition graph for Cop depends on the size of the data domain D and is finite state when D is a finite set.

Input actions and indexing can be mingled as in the following description of a family of registers where both i and x have type $\mathbb{N}$:

$$Reg_i \stackrel{\mathrm{def}}{=} \overline{\texttt{read}}(i).Reg_i + \texttt{write}(x).Reg_x$$

Reg_i can output the value i at the port `read`, or instead it can be updated by being written to at `write`.

We shall implicitly assume different expression types, such as boolean expressions. For instance when i is an integer, Val$(even(i))$ = true if i is even and is false otherwise. This allows us to use conditionals in the definition of a process as exemplified by S that sieves numbers.

$$S \stackrel{\mathrm{def}}{=} \texttt{in}(x).\mathbf{if}\ even(x)\ \mathbf{then}\ \overline{\texttt{out}}_e(x).S\ \mathbf{else}\ \overline{\texttt{out}}_o(x).S$$

Below are the transition rules for this conditional.

$$\boxed{\mathrm{R(if1)}\ \ \frac{\mathbf{if}\ b\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2 \xrightarrow{a} E'}{E_1 \xrightarrow{a} E'}\ \ \mathrm{Val}(b) = \mathrm{true}}$$

$$\boxed{\mathrm{R(if2)}\ \ \frac{\mathbf{if}\ b\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2 \xrightarrow{a} E'}{E_2 \xrightarrow{a} E'}\ \ \mathrm{Val}(b) = \mathrm{false}}$$

Example 1 Consider the following family of processes for $i \geq 1$:

$$T(i) \stackrel{\mathrm{def}}{=} \mathbf{if}\ even(i)\ \mathbf{then}\ \overline{\texttt{out}}(i).T(i/2)\ \mathbf{else}\ \overline{\texttt{out}}(i).T((3i+1)/2)$$

So $T(5)$ performs the sequence of transitions $T(5) \xrightarrow{\overline{\texttt{out}}(5)} T(8) \xrightarrow{\overline{\texttt{out}}(8)} T(4) \xrightarrow{\overline{\texttt{out}}(4)} T(2)$, and then cycles through the transitions $T(2) \xrightarrow{\overline{\texttt{out}}(2)} T(1) \xrightarrow{\overline{\texttt{out}}(1)} T(2)$. $\square$

1.2 Concurrent interaction

A compelling feature of process theory is its modelling of concurrent interaction. A prevalent approach is to appeal to handshake communication as primitive. At any one time only two processes may communicate at a port or along a channel. In CCS the resultant communication is a *completed* internal action. Each incomplete, or observable, action a has a partner $\overline{a}$, its co-action. Moreover the action $\overline{\overline{a}}$ is a which means that a is also the co-action of $\overline{a}$. The partner of a parametrized action $\texttt{in}(v)$ is $\overline{\texttt{in}}(v)$. Simultaneously performing an action and its co-action produces the internal action τ which is a complete action and so does not have a partner.

Concurrent composition of E and F is rendered as the process $E \mid F$. The crucial transition rule for $\mid$ which conveys communication is:

$$\boxed{\mathrm{R}(\mid\mathrm{com})\quad \frac{E \mid F \xrightarrow{\tau} E' \mid F'}{E \xrightarrow{a} E' \qquad F \xrightarrow{\overline{a}} F'}}$$

Consider a potential user of the copier Cop of the previous section who first writes a file before sending it through the port `in`:

$$User \stackrel{\mathrm{def}}{=} \mathtt{write}(x).\,User_x \qquad User_x \stackrel{\mathrm{def}}{=} \overline{\mathtt{in}}(x).\,User$$

As soon as $User$ has written the file v it becomes the process $User_v$ which can communicate with Cop. Rule $\mathrm{R}(\mid\mathrm{com})$ licenses the following proof[4]:

$$\frac{Cop \mid User_v \xrightarrow{\tau} \overline{\mathtt{out}}(v).\,Cop \mid User}{\dfrac{Cop \xrightarrow{\mathtt{in}(v)} \overline{\mathtt{out}}(v).\,Cop}{\mathtt{in}(x).\overline{\mathtt{out}}(x).\,Cop \xrightarrow{\mathtt{in}(v)} \overline{\mathtt{out}}(v).\,Cop} \quad \dfrac{User_v \xrightarrow{\overline{\mathtt{in}}(v)} User}{\overline{\mathtt{in}}(v).\,User \xrightarrow{\overline{\mathtt{in}}(v)} User}}$$

Through this communication the value v is sent from the user to the copier. Data is thereby passed from one process to another. When the actions a and $\overline{a}$ do not involve values, the resulting communication is a synchronization.

Various users can share the copying resource, $Cop \mid (User_{v1} \mid User_{v2})$ admits two users, but only one at a time is allowed to employ it. So other transition rules for $\mid$ are needed, permitting components to proceed without communicating. These rules are:

$$\boxed{\mathrm{R}(\mid)\quad \frac{E \mid F \xrightarrow{a} E' \mid F}{E \xrightarrow{a} E'} \qquad \frac{E \mid F \xrightarrow{a} E \mid F'}{F \xrightarrow{a} F'}}$$

In the first of these rules the process F does not contribute to the action a which E performs. An example derivation is:

$$\frac{Cop \mid (User_{v1} \mid User_{v2}) \xrightarrow{\tau} \overline{\mathtt{out}}(v1).\,Cop \mid (User \mid User_{v2})}{\dfrac{Cop \xrightarrow{\mathtt{in}(v1)} \overline{\mathtt{out}}(v1).\,Cop}{\mathtt{in}(x).\overline{\mathtt{out}}(x).\,Cop \xrightarrow{\mathtt{in}(v1)} \overline{\mathtt{out}}(v1).\,Cop} \quad \dfrac{User_{v1} \mid User_{v2} \xrightarrow{\overline{\mathtt{in}}(v1)} User \mid User_{v2}}{\dfrac{User_{v1} \xrightarrow{\overline{\mathtt{in}}(v1)} User}{\overline{\mathtt{in}}(v1).\,User \xrightarrow{\overline{\mathtt{in}}(v1)} User}}}$$

The goal transition reflects a communication between Cop and $User_{v1}$, and so $User_{v2}$ does not contribute to it. The process $Cop \mid (User_{v1} \mid User_{v2})$ is not forced

[4] We assume that $\mid$ has greater scope than the other process operators: the process $\overline{\mathtt{out}}(v).\,Cop \mid User$ is therefore the parallel composition of $\overline{\mathtt{out}}(v).\,Cop$ and $User$.

to engage in communication, it may instead perform an input $\mathtt{in}(v)$ action or an output action $\overline{\mathtt{in}}(v1)$ or $\overline{\mathtt{in}}(v2)$.

The behaviour of the users sharing the copier is not affected by order of parallel subcomponents or by placement of brackets. Both $(Cop \mid User_{v1}) \mid User_{v2}$ and $User_{v1} \mid (Cop \mid User_{v2})$ have the same capabilities as $Cop \mid (User_{v1} \mid User_{v2})$. Therefore in the sequel we omit brackets between multiple concurrent processes[5].

The parallel operator is expressively powerful. It can be used to describe infinite state systems without invoking infinite indices or value spaces. A simple example is the counter Cnt given by $Cnt \stackrel{\mathrm{def}}{=} \mathtt{up}.(Cnt \mid \mathtt{down}.\mathbf{0})$. Cnt can perform $\mathtt{up}$ and become $Cnt \mid \mathtt{down}.\mathbf{0}$ which in turn can also perform $\mathtt{up}$ and thereby becomes $Cnt \mid \mathtt{down}.\mathbf{0} \mid \mathtt{down}.\mathbf{0}$, and so on.

Figure 5 offers an alternative pictorial representation of the copier Cop and its user process $User$. Such diagrams are called *flow graphs* by Milner [61] (and should be distinguished from behaviour graphs). A flow graph summarizes the potential movement of information flowing into and out of ports, and also ex-

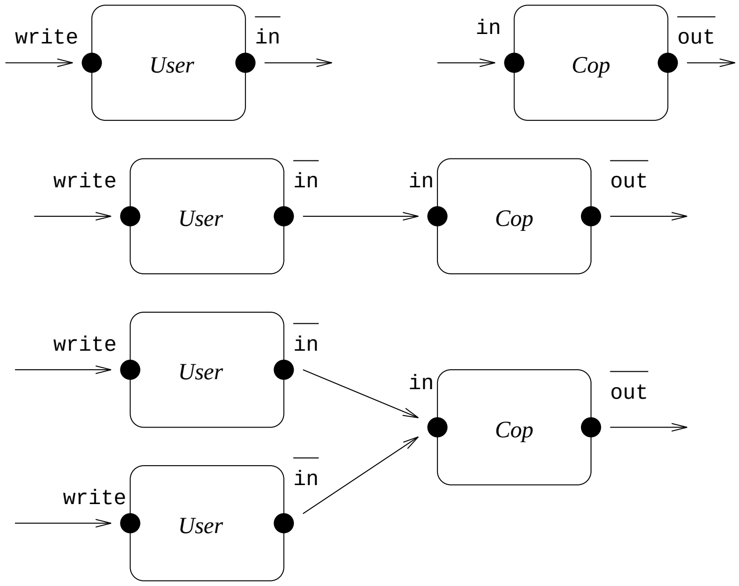


Fig. 5. The flow graphs of $User$, Cop, $Cop \mid User$ and $Cop \mid User \mid User$

hibits the ports through which a process is in principle willing to communicate. In the case of $User$ the incoming arrow to the port labelled $\mathtt{write}$ represents input whereas the outgoing arrow from $\overline{\mathtt{in}}$ symbolizes output. The flow graph for $Cop \mid User$ has the crucial feature that there is a potential linkage between the output port $\mathtt{in}$ of $User$ and its input in Cop permitting information to circulate

[5] See Section 2.4 for a technical justification for this.

from *User* to *Cop* when communication takes place. However this port is still available for other users: both users in $Cop|User|User$ are able to communicate, at different times, with *Cop*.

Consider now the situation where a user has private access to a copier. In process calculi this is modelled using an abstraction or encapsulation operator which conceals ports or channels. In CCS there is a *restriction* operator $\backslash J$ where J ranges over families of *incomplete* actions (thereby excluding τ). Let K be the set $\{\mathtt{in}(v) : v \in D\}$ where D is the space of values that could be accessed through `in`. In the process $(Cop|User)\backslash K$ the port `in` is inaccessible from the outside. Its flow graph is pictured in figure 6 where the linkage without names at the ports represents that they are concealed from other users. This flow graph can therefore be simplified as in the second diagram of figure 6.

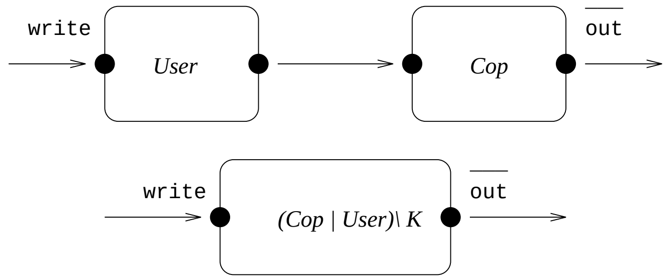


Fig. 6. Flow graphs of $(Cop|User)\backslash K$

The visual effect of $\backslash J$ on flow graphs is underpinned by its transition rule, where the set $\overline{J}$ is $\{\overline{a} : a \in J\}$.

$$\boxed{\mathrm{R}(\backslash) \quad \frac{E\backslash J \xrightarrow{a} F\backslash J}{E \xrightarrow{a} F} \; a \notin J \cup \overline{J}}$$

The behaviour of $E\backslash J$ is part of that of E. The presence of $\backslash K$ prevents *Cop* in $(Cop|User)\backslash K$ from ever doing an `in` transition except in the context of a communication with *User*. This therefore enforces communication between these two components. After an initial write transition, $(Cop|User)\backslash K \xrightarrow{\mathtt{write}(v)} (Cop|User_v)\backslash K$, the only available transition is the communication whose proof is:

$$\cfrac{(Cop|User_v)\backslash K \xrightarrow{\tau} (\overline{\mathtt{out}}(v).Cop|User)\backslash K}{\cfrac{Cop|User_v \xrightarrow{\tau} \overline{\mathtt{out}}(v).Cop|User}{\cfrac{Cop \xrightarrow{\mathtt{in}(v)} \overline{\mathtt{out}}(v).Cop}{\mathtt{in}(x).\overline{\mathtt{out}}(x).Cop \xrightarrow{\mathtt{in}(v)} \overline{\mathtt{out}}(v).Cop} \qquad \cfrac{User_v \xrightarrow{\overline{\mathtt{in}}(v)} User}{\overline{\mathtt{in}}(v).User \xrightarrow{\overline{\mathtt{in}}(v)} User}}}$$

The second user in $(Cop|User)\backslash K|User$ has no access to the copier. As the operator $\backslash J$ is intended to conceal ports, we shall usually abbreviate any set of actions of the form $\{a(v) : v \in D\}$ within a restriction to $\{a\}$: for instance the restriction $\backslash K$ above is more succinctly expressed as $\backslash\{\mathtt{in}\}$.

Process descriptions can become quite large, especially when a process consists of multiple components in parallel. So we shall employ abbreviations of process expressions using the relation $\equiv$ where $E \equiv F$ means that E abbreviates F.

Example 1 The mesh of abstraction and concurrency is further revealed in the following finite state example (without data) of a level crossing of figure 7 from [19] consisting of three components, the road, rail, and the signal. The actions

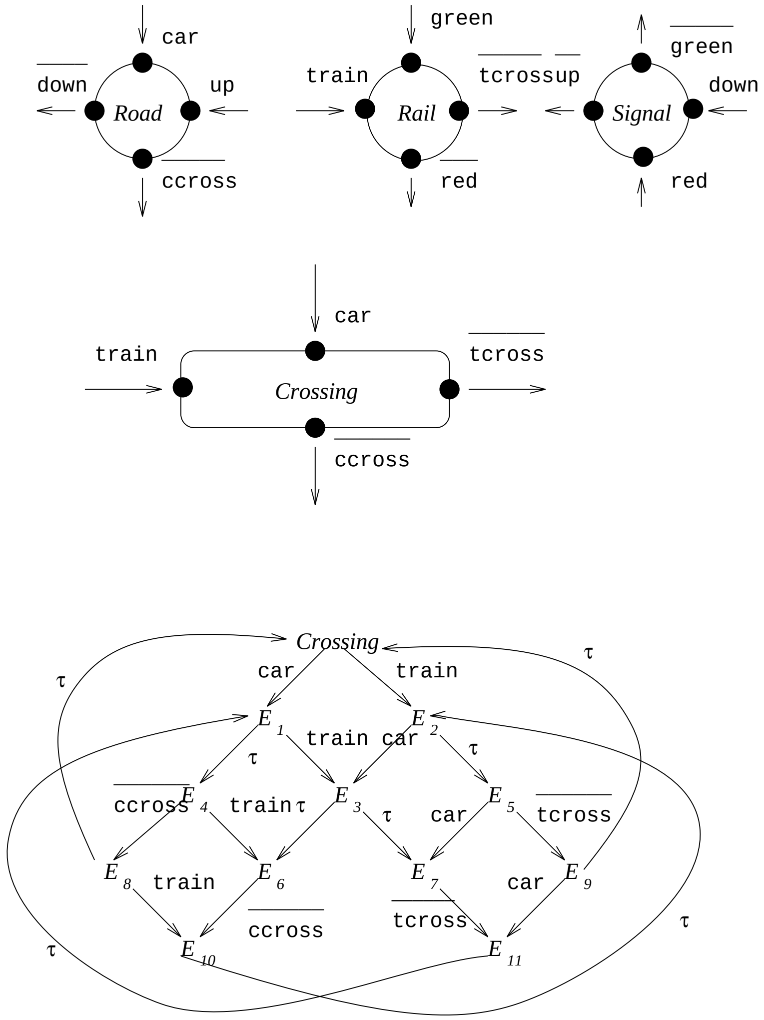
$$
\begin{array}{lcl}
Road & \stackrel{\mathrm{def}}{=} & \mathtt{car}.\mathtt{up}.\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.Road \\
Rail & \stackrel{\mathrm{def}}{=} & \mathtt{train}.\mathtt{green}.\overline{\mathtt{tcross}}.\overline{\mathtt{red}}.Rail \\
Signal & \stackrel{\mathrm{def}}{=} & \overline{\mathtt{green}}.\mathtt{red}.Signal + \overline{\mathtt{up}}.\mathtt{down}.Signal \\
 & & \\
Crossing & \equiv & (Road \mid Rail \mid Signal)\backslash\{\mathtt{green}, \mathtt{red}, \mathtt{up}, \mathtt{down}\}
\end{array}
$$

Fig. 7. A level crossing

$\mathtt{car}$ and $\mathtt{train}$ represent the approach of a car and a train, $\mathtt{up}$ is the gates opening for the car, $\overline{\mathtt{ccross}}$ is the car crossing, $\mathtt{down}$ closes the gates, $\mathtt{green}$ is the receipt of a green signal by the train, $\overline{\mathtt{tcross}}$ is the train crossing, and $\mathtt{red}$ automatically sets the light red. Unlike most crossings it keeps the barriers down except when a car actually approaches and tries to cross. The flow graphs of the components and the overall system are depicted in figure 8 as is its transition graph. Both the *Rail* and the *Road* are simple cyclers, only able to perform a determinate sequence of actions repeatedly. □

An important arena for the use of process descriptions is modelling protocols [70]. An example is the process *Protocol* of figure 9 taken from [81] which models an extremely simple communications protocol that allows a message to be lost during transmission. Its flow graph is the same as that of *Cop* earlier, and the size of its transition graph depends on the space of messages. The sender transmits any message it receives at the port in to the medium. In turn the medium may transmit the message to the receiver, or instead the message may be lost, an action modelled as the silent τ action, in which case the medium sends a timeout signal to the sender and the message is retransmitted. On receiving a message the receiver outputs it at the port out and then sends an acknowledgement directly to the sender (which we assume can not be lost). Having received the acknowledgement, the sender may again receive a message at port in.

Although the flow graphs for *Protocol* and for *Cop* are the same their levels



$$
\begin{aligned}
K \; &= \{\mathtt{green}, \mathtt{red}, \mathtt{up}, \mathtt{down}\} \\
E_0 \; &\equiv \mathit{Crossing} \\
E_1 \; &\equiv (\mathtt{up}.\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.\mathit{Road} \mid \mathit{Rail} \mid \mathit{Signal})\backslash K \\
E_2 \; &\equiv (\mathit{Road} \mid \mathtt{green}.\overline{\mathtt{tcross}}.\overline{\mathtt{red}}.\mathit{Rail} \mid \mathit{Signal})\backslash K \\
E_3 \; &\equiv (\mathtt{up}.\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.\mathit{Road} \mid \mathtt{green}.\overline{\mathtt{tcross}}.\overline{\mathtt{red}}.\mathit{Rail} \mid \mathit{Signal})\backslash K \\
E_4 \; &\equiv (\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.\mathit{Road} \mid \mathit{Rail} \mid \mathtt{down}.\mathit{Signal})\backslash K \\
E_5 \; &\equiv (\mathit{Road} \mid \overline{\mathtt{tcross}}.\overline{\mathtt{red}}.\mathit{Rail} \mid \mathtt{red}.\mathit{Signal})\backslash K \\
E_6 \; &\equiv (\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.\mathit{Road} \mid \mathtt{green}.\overline{\mathtt{tcross}}.\overline{\mathtt{red}}.\mathit{Rail} \mid \mathtt{down}.\mathit{Signal})\backslash K \\
E_7 \; &\equiv (\mathtt{up}.\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.\mathit{Road} \mid \overline{\mathtt{tcross}}.\overline{\mathtt{red}}.\mathit{Rail} \mid \mathtt{red}.\mathit{Signal})\backslash K \\
E_8 \; &\equiv (\overline{\mathtt{down}}.\mathit{Road} \mid \mathit{Rail} \mid \mathtt{down}.\mathit{Signal})\backslash K \\
E_9 \; &\equiv (\mathit{Road} \mid \overline{\mathtt{red}}.\mathit{Rail} \mid \mathtt{red}.\mathit{Signal})\backslash K \\
E_{10} \; &\equiv (\overline{\mathtt{down}}.\mathit{Road} \mid \mathtt{green}.\overline{\mathtt{tcross}}.\overline{\mathtt{red}}.\mathit{Rail} \mid \mathtt{down}.\mathit{Signal})\backslash K \\
E_{11} \; &\equiv (\mathtt{up}.\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.\mathit{Road} \mid \overline{\mathtt{red}}.\mathit{Rail} \mid \mathtt{red}.\mathit{Signal})\backslash K
\end{aligned}
$$

Fig. 8. Flow graphs of the crossing and its components, and its transition graph

$$
\begin{aligned}
\mathit{Sender} &\stackrel{\mathrm{def}}{=} \mathtt{in}(x).\overline{\mathtt{sm}}(x).\mathit{Send1}(x) \\
\mathit{Send1}(x) &\stackrel{\mathrm{def}}{=} \mathtt{ms}.\overline{\mathtt{sm}}(x).\mathit{Send1}(x) + \mathtt{ok}.\mathit{Sender} \\
\mathit{Medium} &\stackrel{\mathrm{def}}{=} \mathtt{sm}(y).\mathit{Med1}(y) \\
\mathit{Med1}(y) &\stackrel{\mathrm{def}}{=} \overline{\mathtt{mr}}(y).\mathit{Medium} + \tau.\overline{\mathtt{ms}}.\mathit{Medium} \\
\mathit{Receiver} &\stackrel{\mathrm{def}}{=} \mathtt{mr}(x).\overline{\mathtt{out}}(x).\overline{\mathtt{ok}}.\mathit{Receiver}
\end{aligned}
$$

$$
\mathit{Protocol} \equiv (\mathit{Sender} \mid \mathit{Medium} \mid \mathit{Receiver})\backslash\{\mathtt{sm}, \mathtt{ms}, \mathtt{mr}, \mathtt{ok}\}
$$

Fig. 9. A simple protocol

of detail are very different. It turns out that these two processes are observationally equivalent in the sense defined later. As process descriptions they are very different. *Cop* is close to a specification as its desired behaviour is given merely in terms of what it does, or how it may react. In contrast *Protocol* is closer to an implementation as it is defined in terms of how it is built from simpler components.

Example 2 The slot machine in figure 10 is an infinite state system (from [19]). Its flow graph is also depicted there. A coin is input (the action `slot`) and then after some silent activity either a loss is output or a winning sum of money. It consists of three components, IO which handles the taking and paying out money, B_n a bank holding n pounds, and D the wheel-spinning decision component. □

$$
\begin{aligned}
IO &\stackrel{\mathrm{def}}{=} \mathtt{slot}.\overline{\mathtt{bank}}.(\mathtt{lost}.\overline{\mathtt{loss}}.IO + \mathtt{release}(y).\overline{\mathtt{win}}(y).IO) \\
B_n &\stackrel{\mathrm{def}}{=} \mathtt{bank}.\overline{\mathtt{max}}(n+1).\mathtt{left}(y).B_y \\
D &\stackrel{\mathrm{def}}{=} \mathtt{max}(z).(\overline{\mathtt{lost}}.\overline{\mathtt{left}}(z).D + \textstyle\sum\{\overline{\mathtt{release}}(y).\overline{\mathtt{left}}(z-y).D \mid 1 \le y \le z\})
\end{aligned}
$$

$$
SM_n \equiv (IO \mid B_n \mid D)\backslash\{\mathtt{bank}, \mathtt{max}, \mathtt{left}, \mathtt{release}\}
$$

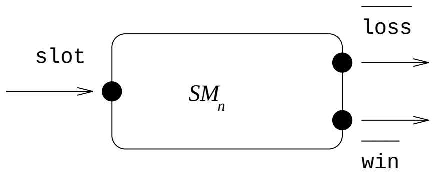


Fig. 10. A slot machine

1.3 Silent transitions

A natural extension of the transition relations $\xrightarrow{a}$ is to finite length sequences of actions or *traces* $a_1 \ldots a_n$. Assume that w ranges over such sequences (with ε as the empty trace). The notation $E \xrightarrow{w} F$ represents that E may perform the trace w and become F. The next transition rule captures this generalization:

$$\boxed{\mathrm{R(tr)} \quad E \xrightarrow{\varepsilon} E \qquad \frac{E \xrightarrow{aw} F}{E \xrightarrow{a} E' \quad E' \xrightarrow{w} F}}$$

For instance, the crossing of figure 7 has the following cyclic behaviour when w is $\mathtt{train}\,\tau\,\overline{\mathtt{tcross}}\,\tau$: $\mathit{Crossing} \xrightarrow{w} \mathit{Crossing}$.

There is an important difference between the completed internal τ action and incomplete actions. An incomplete action is *observable* in the sense that it can be interacted with in a parallel context. Assume that E may at some time perform the action $\mathtt{ok}$, and that F is a resource. Within the process $(E \mid \overline{\mathtt{ok}}.F)\backslash\{\mathtt{ok}\}$ accessibility to this resource is triggered only when E performs $\mathtt{ok}$. Here observation of $\mathtt{ok}$ is the same as the release of the resource. The silent action τ can not be observed in this fashion.

Consequently an important abstraction of the behaviour of processes is away from silent activity. Consider a slightly different copier $C \stackrel{\mathrm{def}}{=} \mathtt{in}(x).\overline{\mathtt{out}}(x).\overline{\mathtt{ok}}.C$, and a user U, defined as $\mathtt{write}(x).\overline{\mathtt{in}}(x).\mathtt{ok}.U$, who writes a file before sending it through $\mathtt{in}$ and then waits for an acknowledgement. The behaviour of $(C \mid U)\backslash\{\mathtt{in}, \mathtt{ok}\}$ is very similar to that of $\mathit{Ucop} \stackrel{\mathrm{def}}{=} \mathtt{write}(x).\overline{\mathtt{out}}(x).\mathit{Ucop}$. Their only difference in capability is internal activity. Both are initially only able to perform a write action. But afterwards *Ucop* outputs immediately whereas the other process must first perform a communication before outputing and then τ again before a second write can happen. When abstracting from silent actions this difference disappears: outwardly both processes repeatedly write and output. Therefore we now define another family of transition relations between processes which epitomizes observable behaviour.

Given a trace w the subsequence of w restricted to the set of actions J, denoted by $w \upharpoonright J$, is the remainder of w when actions outside of J are erased. For instance $(\mathtt{train}\,\tau\,\overline{\mathtt{tcross}}\,\tau) \upharpoonright \{\overline{\mathtt{tcross}}\} = \overline{\mathtt{tcross}}$, and $\tau\,\tau \upharpoonright \{\overline{\mathtt{tcross}}\} = \varepsilon$. Associated with any trace w is its *observable* trace, the subsequence $w \upharpoonright \mathcal{O}$ where $\mathcal{O}$ is a universal set of *observable* actions (containing at least all the actions mentioned in this work apart from τ). For example the observable trace derived from $\mathtt{train}\,\tau\,\overline{\mathtt{tcross}}\,\tau$ is $\mathtt{train}\,\overline{\mathtt{tcross}}$. Observable traces are either empty or built from observable actions. Consequently we can introduce the notion that E may perform the observable trace w and evolve into F: we use the standard notation $E \stackrel{w}{\Longrightarrow} F$ to represent this. A transition rule for such traces utilizes R(tr) and $\upharpoonright$:

$$\boxed{\mathrm{R(Tr)} \quad \frac{E \stackrel{u}{\Longrightarrow} F}{E \xrightarrow{w} F} \quad u = w \upharpoonright \mathcal{O}}$$

Although observable traces abstract from silent activity this does not mean that internal actions can not contribute to differences in observable capability. Let Ven' be a vending machine which is very similar to Ven of Figure 2 except that the initial $\texttt{2p}$ action is prefaced by the silent action τ: $Ven' \stackrel{\text{def}}{=} \tau.\texttt{2p}.Ven_b + \texttt{1p}.Ven_l$. Ven and Ven' can perform the same observable traces. However Ven' is *unstable* as it may lose its initial capability for accepting $\texttt{1p}$, by silently evolving to $\texttt{2p}.Ven_b$. Assume that Use_1 is the user, $Use_1 \stackrel{\text{def}}{=} \overline{\texttt{1p}}.\overline{\texttt{little}}.Use_1$ who is only interested in inserting the smaller coin. The process $(Ven'|Use_1)\backslash\{\texttt{1p}, \texttt{2p}, \texttt{little}\}$ may deadlock before a coin is inserted because $(Ven'|Use_1)\backslash\{\texttt{1p}, \texttt{2p}, \texttt{little}\} \stackrel{\varepsilon}{\Longrightarrow} (\texttt{2p}.Ven_b|Use_1)\backslash\{\texttt{1p}, \texttt{2p}, \texttt{little}\}$. This can not happen in the case of $(Ven|Use_1)\backslash\{\texttt{1p}, \texttt{2p}, \texttt{little}\}$.

Trace transitions were constructed from the transitions of their component actions. Similarly observable traces can also be built from their observable components. The extended observable transition $Crossing \stackrel{\texttt{train}\,\overline{\texttt{tcross}}}{\Longrightarrow} Crossing$ is the result of gluing together the two transitions $Crossing \stackrel{\texttt{train}}{\Longrightarrow} E$ and $E \stackrel{\overline{\texttt{tcross}}}{\Longrightarrow} Crossing$ when the intermediate state E is is either E_2 or E_5 of figure 8. Following [61] we therefore define a new family of transition relations which underpin observable traces. A label in this family is either the empty sequence ε or an observable action a:

$$\text{R}(\stackrel{\varepsilon}{\Longrightarrow}) \quad E \stackrel{\varepsilon}{\Longrightarrow} E \qquad \frac{E \stackrel{\varepsilon}{\Longrightarrow} F}{E \stackrel{\tau}{\longrightarrow} E' \quad E' \stackrel{\varepsilon}{\Longrightarrow} F}$$

$$\text{R}(\stackrel{a}{\Longrightarrow}) \qquad \frac{E \stackrel{a}{\Longrightarrow} F}{E \stackrel{\varepsilon}{\Longrightarrow} E' \quad E' \stackrel{a}{\longrightarrow} F' \quad F' \stackrel{\varepsilon}{\Longrightarrow} E'}$$

The observable behaviour of a process can also be visually encapsulated as a graph. As in section 1.1 the ingredients of this graph are process terms related by directed labelled arcs. Each arc is either $\stackrel{\varepsilon}{\Longrightarrow}$ or $\stackrel{a}{\Longrightarrow}$ where a is observable. Consequently there are *two* behaviour graphs associated with any process. Although both graphs contain the same vertices, they differ in their labelled arcs. Observable graphs are more complex than their counterparts built from the single arrow transitions. However the abundance of arcs may result in redundant vertices, for when minimized with respect to appropriate notions of observational equivalence, the graphs may be dramatically simplified as their vertices are fused. In this way the observable transitions of a process offer an abstraction of their behaviour.

1.4 Renaming and linking

The processes Cop and $User$ are essentially one place buffers, taking in a value and later expelling it. Assume that B is a canonical buffer, $B \stackrel{\text{def}}{=} \texttt{i}(x).\overline{\texttt{o}}(x).B$. Cop is the process B when port $\texttt{i}$ is $\texttt{in}$ and $\texttt{o}$ is $\texttt{out}$ and $User$ is B when $\texttt{i}$ is write and o is in. So these are simple variants of B when ports are renamed. Relabelling of ports or actions can be made explicit as in CCS through a renaming combinator.

The crux of renaming is a function mapping actions into actions. Not all functions are suitable for relabelling. A renaming function f should respect complements, for any observable a the actions $f(a)$ and $f(\overline{a})$ must be co-actions, that is $f(\overline{a}) = \overline{f(a)}$. For simplicity f should also preserve values passing between ports, $f(a(v))$ must be an action $b(v)$ with the same value, and moreover for any other value w the action $f(a(w))$ should be $b(w)$. Finally a renaming function f should conserve the silent action, $f(\tau) = \tau$. Associated with any function f obeying these conditions is the renaming operator $[f]$ which when applied to a process E is written as $E[f]$: this is the process E whose actions are relabelled according to f. Following [61] a renaming function f is usually abbreviated to its essential part: when the a_i are distinct obervable actions $b_1/a_1, \ldots, b_n/a_n$ represents the function f which renames a_i to b_i, and leaves any other action c unchanged, so $f(c) = c$. For instance *Cop* abbreviates the process $B[\mathtt{in}/\mathtt{i}, \mathtt{out}/\mathtt{o}]$: here we maintain the convention that in stands for the family $\{\mathtt{in}(v) : v \in D\}$ and i for $\{\mathtt{i}(v) : v \in D\}$, and so in/i symbolizes the function which maps $\mathtt{i}(v)$ to $\mathtt{in}(v)$ for each v.

The transition rule for renaming is

$$\boxed{\mathrm{R}([f]) \quad \frac{E[f] \xrightarrow{a} F[f]}{E \xrightarrow{b} F} \; a = f(b)}$$

which is used in deriving the following transitions.

$$B[\mathtt{in}/\mathtt{i}, \mathtt{out}/\mathtt{o}] \xrightarrow{\mathtt{in}(v)} (\overline{\mathtt{o}}(v).B)[\mathtt{in}/\mathtt{i}, \mathtt{out}/\mathtt{o}] \xrightarrow{\overline{\mathtt{out}}(v)} B[\mathtt{in}/\mathtt{i}, \mathtt{out}/\mathtt{o}]$$

An important feature of processes is that they can be built from simpler components. Consider for instance how to construct an n-place buffer, following [61], by linking together n instances of B in parallel. The flow graph of n copies of B is pictured in figure 11. For this to become an n-place buffer we need to *link*

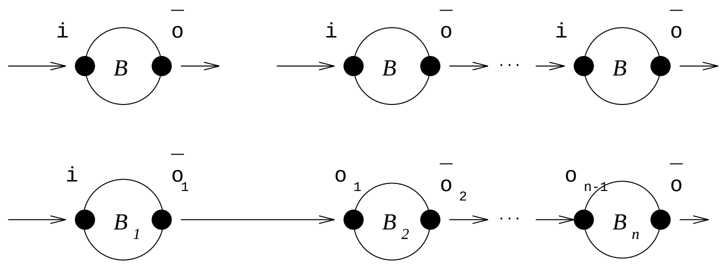


Fig. 11. Flow graph of n instances of B, and $B_1|\ldots|B_n$.

and then internalize the contiguous $\overline{\mathtt{o}}$ and $\mathtt{i}$ ports. Renaming permits linking as the following variants of B show:

$$B_1 \equiv B[\mathtt{o}_1/\mathtt{o}] \quad B_n \equiv B[\mathtt{o}_{n-1}/\mathtt{i}] \quad B_{j+1} \equiv B[\mathtt{o}_j/\mathtt{i}, \mathtt{o}_{j+1}/\mathtt{o}] \quad 1 \leq j < n-1$$

The flow graph of $B_1|\ldots|B_n$ is also depicted in figure 11 with the intended links, and the n-place buffer arises by their internalization, $(B_1|\ldots|B_n)\backslash\{\mathtt{o}_1,\ldots,\mathtt{o}_{n-1}\}$.

Let Cy' be a cycler of length four (as are the *Road* and *Rail* components of the crossing of section 1.2) $Cy' \stackrel{\mathrm{def}}{=} a.c.b.d.Cy'$. Its flow graph is illustrated in figure 12: in this case the flow graph is very close to the transition graph, and so we have circled the a label to indicate that it is initially active.

A more involved example from [61] is the construction of a scheduler from small cycling components. Assume n tasks when $n > 1$, and that action a_i initiates the ith task whereas b_i signals its completion. The scheduler timetables the order of task initiation, ensuring that the sequence of actions $a_1 \ldots a_n$ is performed cyclically starting with a_1. The tasks may terminate in any order, but a task can not be restarted until its previous performance has finished. So the scheduler must guarantee that the actions a_i and b_i happen alternately for each i.

A first attempt builds the required process from a ring of Cy' cyclers. Let Cy'_i for $1 \leq i \leq n$ be the following processes:

$$\begin{aligned} Cy'_1 &\equiv Cy'[a_1/a, c_1/c, b_1/b, \overline{c}_n/d] \\ Cy'_i &\equiv d.Cy'[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d] \quad 1 < i \leq n \end{aligned}$$

When n is four the flow graph of the process $Cy'_1|Cy'_2|Cy'_3|Cy'_4$ with initial states marked is depicted in figure 12. Next the c_i actions are internalized. Let $Sched'_4$ abbreviate the process $(Cy'_1|Cy'_2|Cy'_3|Cy'_4)\backslash\{c_1,\ldots,c_4\}$. The flow graph shows how the tasks must be initiated cyclically (when the c_i actions are internalized): for example a_3 can only happen once a_1 and then a_2 have both happened. Moreover no task can be reinitiated until its previous performance has terminated: action a_3 can not recur until b_3 has happened. However $Sched'_4$ does not permit all possible acceptable behaviour: a simple case is that action b_4 depends on b_1, so task four can not terminate before the initial task.

The solution in [61] is to redefine the cycler as $Cy \stackrel{\mathrm{def}}{=} a.c.(b.d.Cy + d.b.Cy)$, and to use the same renaming functions: let $Cy_i \equiv d.Cy[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d]$ for $1 < i \leq n$, and let Cy_1 be $Cy[a_1/a, c_1/c, b_1/b, \overline{c}_n/d]$. The scheduler, $Sched_n$, is the process $(Cy_1|\ldots|Cy_n)\backslash\{c_1,\ldots,c_n\}$.

1.5 More combinations of processes

In previous sections we have emphasized the process combinators of CCS. However there is a variety of process calculi dedicated to precise modelling of actual systems. Besides CCS and CSP there is ACP due to Bergstra and Klop [14, 9], Hennessy's EPL [45], MEIJE defined by Austry, Boudol and de Simone [7, 74],

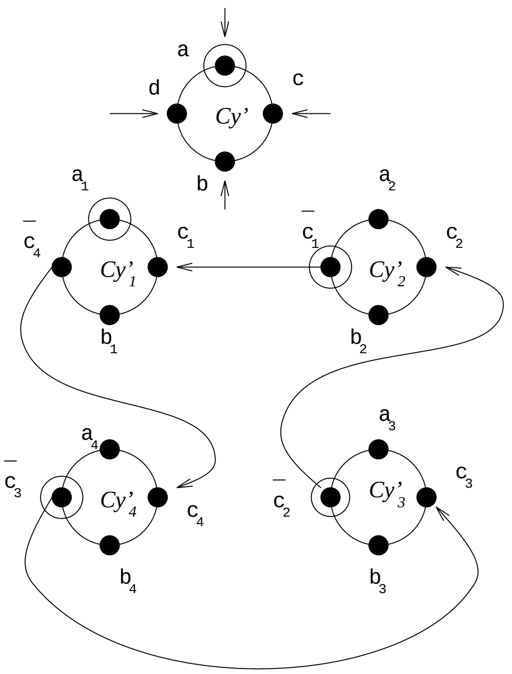


Fig. 12. Flow graph of Cy' and $Cy'_1 | Cy'_2 | Cy'_3 | Cy'_4$

Milner's SCCS [60], and Winskel's general process algebra [84]. Although the behavioural meaning of all the operators of these calculi can be presented as inference rules, their conception reflects different concerns, see [10]. ACP is primarily algebraic, highlighting equations[6]. CSP was devised with a distinguished model in mind, the failures model[7], and MEIJE was introduced as a very expressive calculus, initiating general results about families of rules that can be used to define process operators [44]. Moreover users of the process notation can introduce their own operators according to the application at hand.

Numerous parallel operators are proposed within the calculi mentioned above. Their transition rules are of two kinds. First, where $\times$ is parallel, is a synchronization rule:

$$\frac{E \times F \xrightarrow{ab} E' \times F'}{E \xrightarrow{a} E' \quad F \xrightarrow{b} F'} \ \ldots$$

Here ab is the concurrent product of the component actions a and b, and ... may be filled in with a side condition: in the case of $|$ of Section 1.2 the actions a and b must be co-actions, and their concurrent product is the silent action. Other rules permit components to act alone.

[6] See section 2.7.
[7] See section 2.2 for the notion of failure.

$$\frac{E \times F \xrightarrow{a} E' \times F}{E \xrightarrow{a} E'} \;\ldots \qquad \frac{E \times F \xrightarrow{a} E \times F'}{F \xrightarrow{a} F'} \;\ldots$$

In the case of $|$ there are no side conditions when applying these rules. This general format covers a variety of parallel operators. At one extreme is the case when $\times$ is a synchronous parallel (as in SCCS), when only the synchronization rule applies thereby forcing maximal concurrent interaction: in this case we can think of $\times$ as a pure product on transition graphs. At the other extreme is a pure interleaving operator when the synchronization rule never applies. In between, lie the parallel operators of ACP, CCS, CSP and MEIJE.

A different conception of synchronization underlies the parallel operator of CSP (when data is not passed). Synchronization is *sharing* the same action. Actions now do not have partner co-actions as multiple parallel processes may synchronize. Each process instance in CSP has an associated alphabet consisting of those actions it is willing to engage in. Two processes must synchronize on common actions, belonging to both component alphabets. An alternative presentation which does not require alphabets is to introduce a family of binary parallel operators $\|_K$ indexed by a set K of actions which have to be shared. Rules for $\|_K$ are:

$$\frac{E\|_K F \xrightarrow{a} E'\|_K F'}{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'} \; a \in K$$

$$\frac{E\|_K F \xrightarrow{a} E'\|_K F}{E \xrightarrow{a} E'} \; a \notin K \qquad \frac{E\|_K F \xrightarrow{a} E\|_K F'}{F \xrightarrow{a} F'} \; a \notin K$$

The operator $\|_K$ enforces synchronization. In CCS enforced synchronization is achieved using $|$ and the restriction operator $\backslash K$. Restriction also provides a mechanism for abstracting from observable actions by making them silent. Similarly there is a useful abstraction, or hiding, operator in CSP which we represent as $\backslash\backslash K$. Its rules are

$$\frac{E\backslash\backslash K \xrightarrow{a} F\backslash\backslash K}{E \xrightarrow{a} F} \; a \notin K \qquad \frac{E\backslash\backslash K \xrightarrow{\tau} F\backslash\backslash K}{E \xrightarrow{a} F} \; a \in K$$

Example 1 Recall the example of the scheduler, $Sched_n$, of the previous section. When J is $\{b_1, \ldots, b_n\}$, the observable behaviour of $Sched_n\backslash\backslash J$ is the infinite repetition of the sequence $a_1, \ldots, a_n$ of actions. □

There is a variety of extensions to these basic calculi for modelling real time phenomena such as timeouts expressed using either action duration or delay intervals between actions [67, 64], priorities among actions or among processes [30, 24], and the description of stochastic behaviour using probabilistic instead of non-deterministic choice [56]. A particularly exciting extension is to allow ports to be passed permitting mobile processes which can alter their potential links with each other [62].

Processes can also be used to capture foundational models of computation such as Turing machines, or parallel random-access machines. This remains true for the following restricted process language where P ranges over process names, a over actions, and I over *finite* sets of indices:

$$E ::= P \mid \sum\{a_i.E_i \,:\, i \in I\} \mid E_1 \mid E_2 \mid E\backslash\{a\}$$

A (closed) process is given via a *finite* family of definitions $\{P_i \stackrel{\text{def}}{=} E_i \,:\, 1 \leq i \leq n\}$ where all the process names in each E_i belong to the set $\{P_1, \ldots, P_n\}$. Although process expressions such as the counter Ct_0 (figure 4) the register Reg_0 (section 1.1) and the slot machine SM_0 (figure 10) are excluded because their definitions appeal to infinite sets of indices, their behaviours can be described within this restricted process language. Consider, for example, the following finite reformulation [80] of the counter $Count$:

$$\begin{aligned} Count &\stackrel{\text{def}}{=} \mathtt{round}.Count + \mathtt{up}.(Count_1 \mid a.Count)\backslash\{a\} \\ Count_1 &\stackrel{\text{def}}{=} \mathtt{down}.\overline{a}.\mathbf{0} + \mathtt{up}.(Count_2 \mid b.Count_1)\backslash\{b\} \\ Count_2 &\stackrel{\text{def}}{=} \mathtt{down}.\overline{b}.\mathbf{0} + \mathtt{up}.(Count_1 \mid a.Count_2)\backslash\{a\} \end{aligned}$$

The reader is invited to draw its observable transition graph and compare it with that for Ct_0 of section 1.1.

It is interesting to speculate on whether there should be a "canonical" process calculus, and what the criteria could be for such a description: is there a concurrent version of the Church-Turing thesis for sequential programs? See [1] for some discussion.

2 Modalities and Capabilities

Various examples of processes have been presented from a simple clock to a sophisticated scheduler. In each case a process was given as a term drawn from a process language built from a small number of operators. Behaviour is determined by the transition rules for these process combinators. These rules may involve side conditions relying on extra information as is the case when data is involved. The ingredients of a process description are combinators, predicates, and rules which allow us to deduce transitional behaviour.

In this section we consider various abstractions from the behaviour of processes. Already we have contrasted finite state from infinite state processes: the size being determined by the behaviour graph generated from a process. We have also distinguished observable transitions, marked by the thicker transition arrows $\stackrel{a}{\Longrightarrow}$. Clearly an important discussion surrounds the question of when two processes may be deemed, for all practical purposes, to have the same behaviour. Such an abstraction can be presented by defining an appropriate equivalence relation between processes. A more abstract approach is to examine behavioural properties, and to consider equivalence in terms of having the same pertinent properties. Consequently we first define simple logics which can capture process capabilities.

2.1 Hennessy-Milner logic

We now introduce a *modal* logic for describing local capabilities of processes. Formulas are built from boolean connectives and modal operators $[K]$ ("box K") and $\langle K\rangle$ ("diamond K") where K is a *set* of actions:

$$\boxed{\Phi \ ::= \ \mathtt{tt} \ \mid \ \mathtt{ff} \ \mid \ \Phi_1 \wedge \Phi_2 \ \mid \ \Phi_1 \vee \Phi_2 \ \mid \ [K]\Phi \ \mid \ \langle K\rangle\Phi}$$

This modal logic slightly generalizes Hennessy-Milner logic [46, 47], as sets of actions instead of single actions appear in the modalities.

For any modal formula Φ we define when a process E has, or *satisfies*, the property Φ, in symbols $E \models \Phi$, and if E fails to have the property Φ we write $E \not\models \Phi$. The satisfaction relation $\models$ between processes and formulas is defined inductively on the structure of formulas:

$$\begin{array}{|l|}\hline
E \models \mathtt{tt} \\
E \not\models \mathtt{ff} \\
E \models \Phi \wedge \Psi \text{ iff } E \models \Phi \text{ and } E \models \Psi \\
E \models \Phi \vee \Psi \text{ iff } E \models \Phi \text{ or } E \models \Psi \\
E \models [K]\Phi \ \text{ iff } \forall F \in \{E' : E \xrightarrow{a} E' \text{ and } a \in K\}.\ F \models \Phi \\
E \models \langle K\rangle\Phi \ \text{ iff } \exists F \in \{E' : E \xrightarrow{a} E' \text{ and } a \in K\}.\ F \models \Phi \\
\hline\end{array}$$

The meanings of modalized formulas appeal to the transition behaviour of a process: E has the property $[K]\Phi$ if after every performance of any action in K each resultant process has Φ, and it satisfies $\langle K\rangle\Phi$ if E can evolve to some process obeying Φ by performing an action in K[8]. To reduce the number of brackets in modalities we write $[a_1, \ldots, a_n]$ and $\langle a_1, \ldots, a_n\rangle$ instead of $[\{a_1, \ldots, a_n\}]$ and $\langle\{a_1, \ldots, a_n\}\rangle$.

The simple modal formula $\langle\mathtt{tick}\rangle\mathtt{tt}$ expresses a *capability* for performing the action $\mathtt{tick}$, $E \models \langle\mathtt{tick}\rangle\mathtt{tt}$ iff $\exists F \in \{E' : E \xrightarrow{\mathtt{tick}} E'\}$, which the clock Cl from section 1.1 has. In contrast $[\mathtt{tick}]\mathtt{ff}$ expresses an *inability* to perform $\mathtt{tick}$, $E \models [\mathtt{tick}]\mathtt{ff}$ iff $\{E' : E \xrightarrow{\mathtt{tick}} E'\} = \emptyset$. So Ven has this property but Cl fails to have it. Such basic properties can be embedded within modal operators and between boolean connectives. For instance E can perform the trace $a_1 \ldots a_n$ just in case it has the corresponding property $\langle a_1\rangle \ldots \langle a_n\rangle\mathtt{tt}$. The formula $[\mathtt{tick}](\langle\mathtt{tick}\rangle\mathtt{tt} \wedge [\mathtt{tock}]\mathtt{ff})$[9] expresses the property that after any $\mathtt{tick}$ action it is possible to perform $\mathtt{tick}$ again but not possible to perform $\mathtt{tock}$.

$[K]\mathtt{ff}$ expresses an inability to initially perform any action in K. In the case of the vending machine Ven a button can not be depressed (before money is deposited), so $Ven \models [\mathtt{big}, \mathtt{little}]\mathtt{ff}$. Other interesting properties of Ven are:

[8] A more succinct presentation is, $\Phi \ ::= \ \mathtt{tt} \ \mid \ \neg\Phi \ \mid \ \Phi_1 \wedge \Phi_2 \ \mid \ [K]\Phi$ where $\mathtt{ff} \stackrel{\text{def}}{=} \neg\mathtt{tt}$, $\Phi_1 \vee \Phi_2 \stackrel{\text{def}}{=} \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, and $\langle K\rangle\Phi \stackrel{\text{def}}{=} \neg[K]\neg\Phi$, and where the semantic clause for $\neg$ is : $E \models \neg\Phi$ iff $E \not\models \Phi$.

[9] We assume that $\wedge$ and $\vee$ have wider scope than the modalities $[K]$, $\langle K\rangle$, and that brackets are introduced to resolve any further ambiguities as to the structure of a formula: consequently, $\wedge$ is the main connective of the subformula $\langle\mathtt{tick}\rangle\mathtt{tt} \wedge [\mathtt{tock}]\mathtt{ff}$.

- $\mathit{Ven} \models [2\mathtt{p}]([\mathtt{little}]\mathtt{ff} \wedge \langle\mathtt{big}\rangle\mathtt{tt})$: after $\mathtt{2p}$ is deposited the little button cannot be depressed whereas the big one can.
- $\mathit{Ven} \models [\mathtt{1p}, \mathtt{2p}][\mathtt{1p}, \mathtt{2p}]\mathtt{ff}$: after a coin is entrusted no other coin ($\mathtt{2p}$ or $\mathtt{1p}$) may be deposited.
- $\mathit{Ven} \models [\mathtt{1p}, \mathtt{2p}][\mathtt{big}, \mathtt{little}]\langle\mathtt{collect}_b, \mathtt{collect}_l\rangle\mathtt{tt}$: after a coin is deposited and a button is depressed, an item can be collected.

Verifying that *Ven* has these properties is undemanding. Their proofs merely appeal to the inductive definition of the satisfaction relation between processes and formulas. For instance $\mathit{Ven} \models [\mathtt{1p}, \mathtt{2p}][\mathtt{1p}, \mathtt{2p}]\mathtt{ff}$ iff $\mathit{Ven}_b \models [\mathtt{1p}, \mathtt{2p}]\mathtt{ff}$ and $\mathit{Ven}_l \models [\mathtt{1p}, \mathtt{2p}]\mathtt{ff}$, and clearly both of these hold. Similarly establishing that *Ven* lacks a property, such as $\langle\mathtt{1p}\rangle\langle\mathtt{1p}, \mathtt{big}\rangle\mathtt{tt}$, is equally routine. Notice that it may not be necessary to *construct* or even to know a *full* transition graph for a process when showing that it has or fails to have a property.

Actions in the modalities may contain values. For instance the register Reg_5 from section 1.1 has the property $\langle\overline{\mathtt{read}}(5)\rangle\mathtt{tt} \wedge [\{\overline{\mathtt{read}}(k) \, : \, k \neq 5\}]\mathtt{ff}$.

Assume that $\mathcal{A}$ is a universal set of actions including τ: so $\mathcal{A} = \mathcal{O} \cup \{\tau\}$ where $\mathcal{O}$ is described in section 1.3. We let $-K$ abbreviate the set $\mathcal{A} - K$, and within modalities we write $-a_1, \ldots, a_n$ for $-\{a_1, \ldots, a_n\}$. Moreover we assume that $-$ abbreviates the set $-\emptyset$ (which is just $\mathcal{A}$). Consequently a process E has the property $[-]\Phi$ when each F in the set $\{E' \, : \, E \xrightarrow{a} E' \text{ and } a \in \mathcal{A}\}$ has the feature Φ. The modal formula $[-]\mathtt{ff}$ expresses *deadlock* or *termination*.

Within this modal logic we can also express immediate *necessity* or *inevitability*. The property that only a can be performed, that it *must* be the next action is given by the formula $\langle-\rangle\mathtt{tt} \wedge [-a]\mathtt{ff}$. The conjunct $\langle-\rangle\mathtt{tt}$ affirms that some action is possible while $[-a]\mathtt{ff}$ states that every action except a is impossible. After $\mathtt{2p}$ is deposited *Ven* must perform $\mathtt{big}$, and so $\mathit{Ven} \models [2\mathtt{p}](\langle-\rangle\mathtt{tt} \wedge [-\mathtt{big}]\mathtt{ff})$.

2.2 More modal logics

Process activity is delineated by the two kinds of transition relation distinguished by the thickness of their arrows $\longrightarrow$ and $\Longrightarrow$. The latter captures the performance of observable transitions as $\stackrel{a}{\Longrightarrow}$ permits silent activity before and after a happens: the relation $\stackrel{a}{\Longrightarrow}$ was (see section 1.3) defined in terms of $\xrightarrow{a}$ and the relation $\stackrel{\varepsilon}{\Longrightarrow}$ indicating zero or more silent actions.

The modal logic of the previous Section does not express observable capabilities of processes as silent actions are not accorded a special status. To overcome this it suffices to introduce two new modalities $[\![\ ]\!]$ and $\langle\!\langle\ \rangle\!\rangle$:

$$\boxed{\begin{array}{l} E \models [\![\ ]\!]\Phi \text{ iff } \forall F \in \{E' \, : \, E \stackrel{\varepsilon}{\Longrightarrow} E'\}.\ F \models \Phi \\ E \models \langle\!\langle\ \rangle\!\rangle\Phi \text{ iff } \exists F \in \{E' \, : \, E \stackrel{\varepsilon}{\Longrightarrow} E'\}.\ F \models \Phi \end{array}}$$

These operators are *not* definable within the modal logic of the previous section. Supplementary modalities $[\![K]\!]$ and $\langle\!\langle K\rangle\!\rangle$ are now definable when K is a subset of the *observable* actions $\mathcal{O}$ (which is just the set $\mathcal{A} - \{\tau\}$):

$$[\![K]\!]\Phi \stackrel{\mathrm{def}}{=} [\![\,]\!]\,[K]\,[\![\,]\!]\Phi \qquad \langle\!\langle K\rangle\!\rangle\Phi \stackrel{\mathrm{def}}{=} \langle\!\langle\,\rangle\!\rangle\langle K\rangle\langle\!\langle\,\rangle\!\rangle\Phi$$

The derived meanings of these modalities appeal to the observable transition relations $\stackrel{a}{\Longrightarrow}$ in the same way that their counterparts $[K]$ and $\langle K\rangle$ appeal to $\stackrel{a}{\longrightarrow}$. We write $[\![a_1,\ldots,a_n]\!]$ and $\langle\!\langle a_1,\ldots,a_n\rangle\!\rangle$ instead of $[\![\{a_1,\ldots,a_n\}]\!]$ and $\langle\!\langle\{a_1,\ldots,a_n\}\rangle\!\rangle$.

The simple modal formula $\langle\!\langle \mathtt{tick}\rangle\!\rangle\mathtt{tt}$ expresses the observable capability for performing the action $\mathtt{tick}$ while $[\![\mathtt{tick}]\!]\,\mathtt{ff}$ expresses an inability to tick after any amount of internal activity. Both clocks $Cl \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl$ and $Cl' \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl' + \tau.\mathbf{0}$ have the property $\langle\!\langle \mathtt{tick}\rangle\!\rangle\mathtt{tt}$ but Cl' may at any time silently stop ticking, and therefore it also has the property $\langle\!\langle \mathtt{tick}\rangle\!\rangle\,[\![\mathtt{tick}]\!]\,\mathtt{ff}$.

Example 1 *Crossing* $\models [\![\mathtt{car}]\!]\,[\![\mathtt{train}]\!]\,(\langle\!\langle\overline{\mathtt{tcross}}\rangle\!\rangle\mathtt{tt} \vee \langle\!\langle\overline{\mathtt{ccross}}\rangle\!\rangle\mathtt{tt})$, but it fails $[\![\mathtt{car}]\!]\,[\![\mathtt{train}]\!]\,(\langle\!\langle\overline{\mathtt{tcross}}\rangle\!\rangle\mathtt{tt} \wedge \langle\!\langle\overline{\mathtt{ccross}}\rangle\!\rangle\mathtt{tt})$. □

Modal formulas can also be used to express notions that are basic to the theory of CSP [48]. A process may perform the observable trace $a_1\ldots a_n$ provided that it has the property $\langle\!\langle a_1\rangle\!\rangle\ldots\langle\!\langle a_n\rangle\!\rangle\mathtt{tt}$. The formula $[\![K]\!]\,\mathtt{ff}$ expresses that the observable set of actions K is a *refusal*: $E \models [\![K]\!]\,\mathtt{ff}$ just in case E can not silently evolve into a process which can perform an action in K. The pair $(a_1\ldots a_n, K)$ is a *failure* for a process if it has the property $\langle\!\langle a_1\rangle\!\rangle\ldots\langle\!\langle a_n\rangle\!\rangle([\tau]\mathtt{ff} \wedge [\![K]\!]\,\mathtt{ff})$: E satisfies this formula if it can perform the observable trace $a_1\ldots a_n$ and become *stable* (unable to perform τ) and also be unable to perform any observable action in K.

Recall that $\mathcal{O}$ is a universal set of observable actions (which does not contain τ). Let $[\![-K]\!]$ abbreviate $[\![\mathcal{O}-K]\!]$ and similarly for $\langle\!\langle -K\rangle\!\rangle$. Moreover let $[\![-]\!]$ and $\langle\!\langle -\rangle\!\rangle$ be abbreviations for $[\![\mathcal{O}]\!]$ and $\langle\!\langle\mathcal{O}\rangle\!\rangle$. This means that the modal formula $[\![-]\!]\,\mathtt{ff}$ expresses an inability to perform an *observable* action: so the process Div has this property when $Div \stackrel{\mathrm{def}}{=} \tau.Div$.

The modal logic of the previous section permits the expression of immediate necessity or inevitability. The property that a process must perform a next was given by the formula $\langle -\rangle\mathtt{tt} \wedge [-a]\mathtt{ff}$. However the similarly structured formula $\langle\!\langle -\rangle\!\rangle\mathtt{tt} \wedge [\![-a]\!]\,\mathtt{ff}$ does not preclude the possibility that the observable action a becomes excluded through silent activity. For instance both clocks Cl and Cl' earlier have the feature $\langle\!\langle -\rangle\!\rangle\mathtt{tt} \wedge [\![-\mathtt{tick}]\!]\,\mathtt{ff}$: Cl' has this property because it is able to perform an observable transition (so satisfies $\langle\!\langle -\rangle\!\rangle\mathtt{tt}$) and is unable to perform any observable action other than $\mathtt{tick}$ (and so satisfies $[\![-\mathtt{tick}]\!]\,\mathtt{ff}$). But Cl' may silently break down and therefore be unable to tick. This shortcoming can be surmounted with the following strengthened formula for expressing immediate necessity, $[\![\,]\!]\,\langle\!\langle -\rangle\!\rangle\mathtt{tt} \wedge [\![-\mathtt{tick}]\!]\,\mathtt{ff}$. Now $Cl' \not\models [\![\,]\!]\,\langle\!\langle -\rangle\!\rangle\mathtt{tt}$ because of the silent transition $Cl' \stackrel{\varepsilon}{\Longrightarrow} \mathbf{0}$. But there is still a question mark over this formula as an expression of necessity. Let Cl_1 be a further clock, $Cl_1 \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl_1 + \tau.Cl_1$, which satisfies $[\![-\mathtt{tick}]\!]\,\mathtt{ff}$ as its only observable transition is that of $\mathtt{tick}$. However it also has the property $[\![\,]\!]\,\langle\!\langle -\rangle\!\rangle\mathtt{tt}$ as the set $\{F : Cl_1 \stackrel{\varepsilon}{\Longrightarrow} F\}$ contains the sole element Cl_1 which obeys $\langle\!\langle - \rangle\!\rangle \mathtt{tt}$. But interpreting this as the inevitability that `tick` must be performed fails to take into account the possibility that Cl_1 perpetually engages in internal activity and therefore never ticks.

A process *diverges* if it is able to perform internal actions for ever: we write $E \uparrow$ if E diverges following [45], and $E \downarrow$ if E *converges* (fails to diverge). So $Cl_1 \uparrow$ whereas $Cl' \downarrow$. Convergence and divergence are not definable in the modal logics introduced so far. Consequently we introduce another modality $[\![\downarrow]\!]$, similar to $[\![\ ]\!]$, except it contains information about convergence:

$$\boxed{E \models [\![\downarrow]\!]\Phi \ \text{ iff } \ E\downarrow \ \text{ and } \forall F \in \{E' : E \stackrel{\varepsilon}{\Longrightarrow} E'\}.\ F \models \Phi}$$

From this we can define the modality $[\![\downarrow K]\!]$ as $[\![\downarrow]\!]\,[K]\,[\![\ ]\!]$:

$$E \models [\![\downarrow K]\!]\Phi \ \text{ iff } \ E\downarrow \ \text{ and } \forall F \in \{E' : E \stackrel{a}{\Longrightarrow} E' \text{ and } a \in K\}.\ F \models \Phi$$

Other mixtures are also definable: $[\![K \downarrow]\!]$ as $[\![\ ]\!]\,[K]\,[\![\downarrow]\!]$; and $[\![\downarrow K \downarrow]\!]$ as $[\![\downarrow]\!]\,[K]\,[\![\downarrow]\!]$. Thus the stronger necessity $[\![\downarrow]\!]\,\langle\!\langle - \rangle\!\rangle \mathtt{tt} \wedge [\![-\mathtt{tick}]\!]\,\mathtt{ff}$ excludes divergence.

Features of processes concerning divergence, appealed to in definitions of behavioural refinement [45], can also be expressed as modal formulas in this extended modal logic. For instance the strong property that there can not be divergence throughout the observable trace $a_1 \ldots a_n$ is given as $[\![\downarrow a_1 \downarrow]\!] \ldots [\![\downarrow a_n \downarrow]\!]\,\mathtt{tt}$.

2.3 Process equivalences

Process expressions are intended to be used for describing actual systems. However our discussion has omitted criteria for when two such expressions may be said to describe the *same* system. Alternatively we can consider grounds for differentiating process descriptions. Undoubtedly the clock Cl and the vending machine Ven of section 1.1 are different. They are intended as models of distinct kinds of objects. Moreover at every level they differ, their algebraic expressions, their action names, their flow graphs, and their transition graphs. A concrete manifestation of these differences is their initial capabilities. The clock Cl can perform the observable action `tick` whereas Ven can not: so Cl has the properties $\langle \mathtt{tick} \rangle \mathtt{tt}$ and $\langle\!\langle \mathtt{tick} \rangle\!\rangle \mathtt{tt}$ which Ven fails to have.

Syntactic differences alone should not be sufficient grounds for distinguishing processes. It is important to allow the possibility that two process descriptions may be equivalent even though they differ markedly in their level of detail. An example is the two descriptions of a counter Ct_0 of figure 4 and $Count$ of section 1.5. An account of process equivalence has practical significance when we view process expressions both as *specifications* and as *implementations* depending on how detailed they are. Ct_0 can be viewed as a specification (even the requirement specification) of a counter whereas the finite description $Count$ with its very different structure could be seen as an implementation. Similarly the buffer Cop can be seen as a simple specification of the process $Protocol$ of section 1.2. In this context, a theory of process equivalence could tell us when an implementation *meets* its specification[10].

Not only do Ct_0 and *Count* have the same initial capabilities, but also this feature is preserved as observable actions are performed. There is a similarity between their observable transition graphs, a resemblance which is not immediately easy to define. A simpler case is the following two clocks, $Cl \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl$ and $Cl_2 \stackrel{\mathrm{def}}{=} \mathtt{tick}.\mathtt{tick}.Cl_2$, which appear to be equivalent descriptions of clocks even though they have different transition graphs. Clearly Cl and $\mathtt{tick}.Cl$ count as equivalent expressions. Also replacing a subexpression by an equivalent expression should preserve equivalence. If we therefore replace Cl with $\mathtt{tick}.Cl$ in the expression $\mathtt{tick}.Cl$, the resulting expression $\mathtt{tick}.\mathtt{tick}.Cl$ should be equivalent to Cl. Since particular names of processes are unimportant this should imply that Cl and Cl_2 are also equivalent.

The extensionality principle here is that an equivalence should be a *congruence*, preserved by the various process combinators. For example if the decision component D of the slot machine SM_n breaks down then replacing it with an equivalent component should not affect the overall behaviour of the system (up to equivalence). In principle, an equivalence as a congruence offers the possibility for structured reasoning about combinations of processes, and also the possibility that equivalence proofs may be founded on equational reasoning.

Clearly if processes have different initial capabilities then they should not be deemed equivalent. This can be strengthened to initial necessities: that if one process must perform an action whereas another need not then they should be distinguished. We can imagine taking some simple properties as the basic distinguishable features. Two processes can then be counted as equivalent if when they are placed in any context the resultant processes have the same basic properties, thereby guaranteeing a congruence.

As an example consider the following three vending machines,

$$\begin{aligned}
Ven_1 &\stackrel{\mathrm{def}}{=} \mathtt{1p}.\mathtt{1p}.(\mathtt{tea}.Ven_1 + \mathtt{coffee}.Ven_1)\\
Ven_2 &\stackrel{\mathrm{def}}{=} \mathtt{1p}.(\mathtt{1p}.\mathtt{tea}.Ven_2 + \mathtt{1p}.\mathtt{coffee}.Ven_2)\\
Ven_3 &\stackrel{\mathrm{def}}{=} \mathtt{1p}.\mathtt{1p}.\mathtt{tea}.Ven_3 + \mathtt{1p}.\mathtt{1p}.\mathtt{coffee}.Ven_3
\end{aligned}$$

which have the same (observable) traces. Assume a user who only wishes to drink a single tea by offering coins and having done so expresses visible satisfaction as the action $\overline{\mathtt{ok}}$: $Use \stackrel{\mathrm{def}}{=} \overline{\mathtt{1p}}.\overline{\mathtt{1p}}.\overline{\mathtt{tea}}.\overline{\mathtt{ok}}.\mathbf{0}$. For each of the three vending machines we can build the process $(Ven_i \mid Use)\backslash K$ where K is the set $\{\mathtt{1p}, \mathtt{tea}, \mathtt{coffee}\}$. In the first case the user must express satisfaction: $(Ven_1 \mid Use_2)\backslash K \models [\![\ ]\!]\, \langle\!\langle - \rangle\!\rangle \mathtt{tt} \wedge [[-\overline{\mathtt{ok}}]]\ \mathtt{ff}$. But in the other two cases this is not so. They may evolve silently, $(Ven_i \mid Use)\backslash K \stackrel{\varepsilon}{\Longrightarrow} (\mathtt{coffee}.Ven_i \mid \overline{\mathtt{tea}}.\overline{\mathtt{ok}}.\mathbf{0})$ and thereby be deadlocked. Therefore it is clear that Ven_1 should not be equivalent to Ven_2 or Ven_3. With respect to the failures model of CSP [48] and the testing framework of [45] Ven_2 and

[10] Similar comments could be made about *refinement* where we would expect an ordering on processes.

Ven_3 are equivalent. Finer equivalences distinguish them on the basis that once a coin has been inserted in Ven_3, unlike in Ven_2, any possible successful collection of tea must already be decided. Imagine that after a single coin has been inserted the resulting state is copied for a number of users: in the case of Ven_3 all these users must express satisfaction or all of them must deadlock. Let Rep be a process operator that replicates successor processes

$$\frac{Rep(E) \xrightarrow{a} E' | E'}{E \xrightarrow{a} E'}$$

So $(Rep(Ven_2 | Use))\backslash K \models \langle\langle \overline{\texttt{ok}} \rangle\rangle \, [[\overline{\texttt{ok}}]] \, \texttt{ff}$, but $(Rep(Ven_3 | Use))\backslash K$ fails to have this property.

Therefore one problem with this notion of equivalence is that it is very sensitive to the family of operators that are allowed within process expressions. This has prompted research into general classes of operators and the congruences they induce, see for instance [44].

A different approach is to consider equivalences induced by families of modal formulas. Assume Γ is a set of modal formulas, the subset of Γ true of E is given by $\| E \|^{\Gamma}$, the set $\{\Phi \in \Gamma \, : \, E \models \Phi\}$. Each set induces an equivalence relation $\equiv_{\Gamma}$ on processes: $E \equiv_{\Gamma} F$ iff $\| E \|^{\Gamma} = \| F \|^{\Gamma}$.

Example 1 If Γ consists of all formulas of the form $\langle a_1 \rangle \ldots \langle a_n \rangle \texttt{tt}$, for $n \geq 0$, then $\equiv_{\Gamma}$ is traces equivalence. Instead if Γ contains all formulas of the form $\langle\langle a_1 \rangle\rangle \ldots \langle\langle a_n \rangle\rangle ([\![K]\!] \, \texttt{ff} \wedge [\tau]\texttt{ff})$ then $\equiv_{\Gamma}$ is now failures equivalence [48]. □

2.4 Bisimulations and games

Equivalences for CCS processes appeal to *bisimulation* relations [60, 69] based on the idea that an observer can repeatedly interact with a process, and that two processes are equivalent if no matter how often they are interacted with differences can not be detected. Before introducing bisimulations we first formalize the idea of interaction using games.

A pair of processes (E_0, F_0) can be viewed as an interactive game to be played by two participants or observers, player I and player II. A *play* of the game (E_0, F_0) is a finite or infinite sequence of the form: $(E_0, F_0) \ldots (E_i, F_i) \ldots$. Player I attempts to show that a conspicuous difference between the initial processes is detectable whereas player II wishes to establish that they can not be distinguished. Suppose an initial part of a play is $(E_0, \Phi_0), ldots, (E_j, \Phi_j)$. The next pair (E_{j+1}, F_{j+1}) is decided as follows: first player I chooses a transition $E_j \xrightarrow{a} E_{j+1}$ from E_j or a transition $F_j \xrightarrow{a} F_{j+1}$ from F_j, and then player II chooses a corresponding transition, one with the same label, from the other process $F_j \xrightarrow{a} F_{j+1}$ or $E_j \xrightarrow{a} E_{j+1}$.

If a player is unable to make a move then the other player *wins* that play of the game. Player II loses when no corresponding transition is available in response to a move from player I. This happens in the configuration (E_j, F_j) when one of these processes is able to perform an action which the other can not; a manifest difference is then detectable. Player I loses in the configuration

(E_j, F_j) when both these processes are deadlocked. Player II also wins in two other circumstances. If there is a repeat configuration, when (E_j, F_j) has already occurred earlier in the play, then player I loses. Secondly, player II counts as winning any infinite length play. In both these cases player I has been unable to expose a difference between the initial processes. Given these winning conditions, an infinite length play is only possible when at least one of the initial processes is infinite state.

Example 1 Player II must win any play of the game (Cl, Cl_2) when $Cl \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl$ and $Cl_2 \stackrel{\mathrm{def}}{=} \mathtt{tick}.\mathtt{tick}.Cl_2$. Any play has to progress through the steps (Cl, Cl_2) $(Cl, \mathtt{tick}.Cl_2)$ (Cl, Cl_2) thereby ensuring that player I loses because of the repeat configuration. It does not matter which moves either player makes as player II is guaranteed to win. In the case of the game (Cl, Cl_3) when Cl_3 is the clock $Cl_3 \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl_3 + \mathtt{tick}.\mathbf{0}$ there are plays that player I wins and plays that player II wins. However player I has the power to win any play by initially choosing the transition $Cl_3 \xrightarrow{\mathtt{tick}} \mathbf{0}$. Similarly player II is able to win any play of the game (Cl_3, Cl_3) just by copying any move that player I makes in the other process. $\square$

A player is said to have a *winning strategy* for the interactive game (E_0, F_0) if she is able to win any play irrespective of whatever moves her opponent makes. It turns out that for any pair of processes one of the players has a winning strategy.

Example 2 The different choice points in the vending machines Ven_2 and Ven_3 of the previous section ensure that player I has a winning strategy for the game (Ven_2, Ven_3). $\square$

Example 3 Player II has a winning strategy for the game (B, B') where B is $\mathtt{in}.B \mid \mathtt{out}.B$ and $B' \stackrel{\mathrm{def}}{=} \mathtt{in}.B' + \mathtt{out}.B'$. In this case any play has to be of infinite length, and player II's strategy resides with the fact that she is always able to respond to any move from player I. An example play consisting of $\mathtt{in}$ moves only is, $(B, B')(B|\mathtt{out}.B, B')(B|\mathtt{out}.B|\mathtt{out}.B, B')\ldots$, which illustrates that an infinite length play requires one of the processes to have an infinite state space. $\square$

In the circumstance that player II has a winning strategy for the game (E, F) we say that process E is *game equivalent* to process F. This means that player I is therefore unable to distinguish E and F. Examples of game equivalent processes are Cl and Cl_2 of example 1, and B and B' of example 3. Game equivalence is indeed an equivalence relation. An indirect proof follows from the following alternative characterization.

A winning strategy for player II consists of always being able to match player I's moves. Underpinning this is a *bisimulation* relation between processes.

Definition 1 A binary relation $\mathcal{R}$ between processes is a *bisimulation* relation just in case whenever $E\mathcal{R}F$ then for all $a \in \mathcal{A}$,

1. if $E \xrightarrow{a} E'$ then $F \xrightarrow{a} F'$ and $E'\mathcal{R}F'$ for some F'
2. if $F \xrightarrow{a} F'$ then $E \xrightarrow{a} E'$ and $E'\mathcal{R}F'$ for some E'

This definition places the hereditary conditions of 1 and 2 on when a relation between processes counts as a bisimulation: for instance the empty relation and the identity relation are bisimulations. A bisimulation relation is a witness for player II's winning strategy. Two processes E, F are *bisimulation equivalent* (or *bisimilar*), written $E \sim F$, if there is a bisimulation that relates them. Equivalently, as Proposition 1 tells us, $E \sim F$ just in case these processes are game equivalent.

Proposition 1 *E is game equivalent to F iff $E \sim F$.*

Parallel composition is both commutative and associative with respect to bisimulation equivalence (as is $+$): this permits us to drop bracketing in the case of a process description with multiple parallel components. Bisimulation equivalence is also a *congruence* with respect to *all* the process combinators introduced in previous sections (including the operator *Rep*).

Bisimulation equivalence is a very fine equivalence between processes, reflecting the idea that in the presence of concurrency a more intensional description of behaviour is needed than for instance the set of completed traces of a process. Although this equivalence is undecidable for full CCS, an interesting question is when it is decidable. Clearly this is the case for finite state processes. However surprizingly it remains true for families of infinite state processes including context free processes [8, 27].

2.5 Bisimulations and modal properties

There is an intimate relationship between bisimulation equivalence and modal properties. Let Γ be the family of all modal formulas built from the boolean connectives and any of the modal operators $\{[K], \langle K\rangle, [\![-]\!], \langle\!\langle -\rangle\!\rangle, [\![\downarrow]\!]\}$ from sections 2.1 and 2.2. First there is the result that bisimulation equivalence preserves modal properties[11].

Proposition 1 *If $E \sim F$ then $E \equiv_\Gamma F$.*

Consequently when processes are bisimilar they have the very same capabilities and necessities.

The converse of Proposition 1 holds for a restricted set of processes. A process E is immediately image-finite if for each $a \in \mathcal{A}$ the set $\{F : E \xrightarrow{a} F\}$ is finite. And E is *image-finite* if every member of $\{F : \exists w \in \mathcal{A}^*.\, E \xrightarrow{w} F\}$ is immediately image-finite.

Proposition 2 *If E and F are image-finite and $E \equiv_\Gamma F$ then $E \sim F$.*

Clearly Proposition 1 remains true for any subset of the properties Γ1. Proposition 2 is still true for the subset of modal formulas defined in section 2.1, when

[11] Recall from section 2.3 that $E \equiv_\Gamma F$ iff $\{\Phi \in \Gamma : E \models \Phi\} = \{\Phi \in \Gamma : F \models \Phi\}$.

the only modalities are $[K]$ and $\langle K \rangle$. In fact it still holds if the sets K are restricted to be of size one. These two results under this restriction are known as the modal *characterization* of bisimulation equivalence, due to Hennessy and Milner [46, 47]. This characterization not only reinforces the naturalness of bisimulation equivalence but also suggests that modal logic is a very appropriate logic for processes[12].

The need for the restriction of image-finiteness in Proposition 2 can be seen in the following example: let $Cl^{i+1} \stackrel{\text{def}}{=} \texttt{tick}.Cl^i$, $Cl^0 \stackrel{\text{def}}{=} \mathbf{0}$, and $Cl \stackrel{\text{def}}{=} \texttt{tick}.Cl$. Let E be the process $\sum\{Cl^i : i \geq 0\}$, and let F be $E + Cl$. It follows that $E \not\sim F$ but $E \equiv_\Gamma F$.

Let M_∞ be the following modal logic where I ranges over arbitrary finite and infinite indexing families:

$$\Phi ::= \bigwedge\{\Phi_i : i \in I\} \mid \bigvee\{\Phi_i : i \in I\} \mid [K]\Phi \mid \langle K \rangle \Phi$$

E has the property $\bigwedge\{\Phi_i : i \in I\}$ if E has the property Φ_j for every $j \in I$ (and $E \models \bigvee\{\Phi_i : i \in I\}$ iff $E \models \Phi_j$ for some $j \in I$). The atomic formula $\texttt{tt}$ can be defined as $\bigwedge\{\Phi_i : \in \emptyset\}$ and $\texttt{ff}$ as $\bigvee\{\Phi_i : i \in \emptyset\}$. The following unrestricted characterization result now holds: $E \sim F$ iff $E \equiv_{M_\infty} F$.

2.6 Weak bisimulations

In the previous two sections we have seen that there is a close relationship between winning strategies, bisimulation relations, and modal properties. Two processes are game equivalent if they are related by a bisimulation and they then enjoy the same modal properties. Moreover if they are image-finite and have the same modal properties then they are bisimiliar and also game equivalent. However not one of this trio abstracts from the silent action τ, as each appeals to the family of transition relations $\{\stackrel{a}{\longrightarrow} : a \in \mathcal{A}\}$. By consistently replacing this set with the family of weak transitions, these notions can abstract from τ. Therefore we can define *weak* games which use the thicker transition relations $\stackrel{a}{\Longrightarrow}$, $a \in \mathcal{O} \cup \{\varepsilon\}$ and *weak* bisimulation relations, and *weak* modal logic whose modalities are restricted to the set $\{[\![K]\!], [\![\ ]\!], \langle\!\langle K \rangle\!\rangle, \langle\!\langle\ \rangle\!\rangle\}$.

The next move in a play $(E_0, F_0) \ldots (E_j, F_j)$ of the weak game is: player I chooses E_j (or F_j) and a transition $E_j \stackrel{a}{\Longrightarrow} E_{j+1}$ (or $F_j \stackrel{a}{\Longrightarrow} F_{j+1}$) when $a \in \mathcal{O} \cup \{\varepsilon\}$ and then player II chooses a corresponding transition from the other process $F_j \stackrel{a}{\Longrightarrow} F_{j+1}$ (or $E_j \stackrel{a}{\Longrightarrow} E_{j+1}$). If player II becomes stuck, unable to make a move, then player I *wins* that play. (Note that player I can not become stuck as the empty transition $E \stackrel{\varepsilon}{\Longrightarrow} E$ is always available.) If a play continues forever or a repeat configuration occurs then player II wins it.

A player has a *winning strategy* from (E_0, F_0) if she is able to win any play from this pair. Two processes E_0 and F_0 are *weak game equivalent* if player II

[12] In fact bisimulations occur in the theory of modal logic as zig-zag relations in [15].

has a winning strategy from (E_0, F_0). Pairs of processes that are weakly equivalent are $(C \mid U)\backslash\{\mathtt{in}, \mathtt{ok}\}$ and $Ucop$ from section 1.3, Cop and $Protocol$, and the counters Ct_0 and $Count$.

Underpinning weak game equivalence is the existence of a weak bisimulation relation, which is defined as in the previous section except with respect to the thicker transitions $\stackrel{a}{\Longrightarrow}$ when $a \in \mathcal{O} \cup \{\varepsilon\}$. Two processes are *weakly bisimilar*, or *observationally equivalent*, given by the relation $\approx$, if they are tied by a weak bisimulation relation. Such a relation is a witness for player II's winning strategy. There is also an intimate relationship between weak bisimulation equivalence and weak modal properties. Let Γ be the family of modal formulas built from the boolean connectives and any of the modal operators $\{[\![\ ]\!], \langle\!\langle\ \rangle\!\rangle, [\![K]\!], \langle\!\langle K \rangle\!\rangle\}$ from section 2.2.

Proposition 1 *If* $E \approx F$ *then* $E \equiv_\Gamma F$.

Notice that this result is *not* true if we include the modalities from section 2.1, or the divergence sensitive modality $[\![\downarrow]\!]$. The converse of Proposition 1 holds for *weakly* image-finite processes.

Weak bisimilarity is not however a congruence because of the initial preemptive power of τ: $E \approx \tau.E$ but in general $F + E \not\approx F + \tau.E$. So in CCS [47, 61] the equivalence used is the largest subset of $\approx$ which is also a congruence. It turns out this equivalence, denoted by $E \approx^c F$, can also be described using transitions: $E \approx^c F$ iff $E \approx F$ and the conditions

$$\begin{array}{l} \text{if } E \stackrel{\tau}{\longrightarrow} E' \text{ then } \exists F_1, F'.\, F \stackrel{\tau}{\longrightarrow} F_1 \stackrel{\varepsilon}{\Longrightarrow} F'.\ E' \approx F' \\ \text{if } F \stackrel{\tau}{\longrightarrow} F' \text{ then } \exists E_1, E'.\, E \stackrel{\tau}{\longrightarrow} E_1 \stackrel{\varepsilon}{\Longrightarrow} E'.\ E' \approx F' \end{array}$$

If E and F are stable then $E \approx F$ implies $E \approx^c F$. There is a finer weak bisimulation equivalence called *branching bisimulation equivalence*. It also has a logical characterization [35]. Notice that weak bisimilarity and its congruence are not sensitive to divergence. So they do not preserve the strong necessity properties discussed in Section 2.2. However it is possible to define equivalences that take divergence into account [45, 48, 83].

2.7 Equivalence checking

A simple proof that two processes are strong or weak bisimulation equivalent is to exhibit the appropriate bisimulation relation. Alternatively proofs can utilize equational reasoning. There is an assortment of algebraic theories of processes depending on the congruence and the process combinators: for details see [45, 48, 9, 61]. We now present a proof in the equational theory for CCS that a simplified slot machine without data values meets its specification. The equivalence shown is the weak bisimulation congruence defined in the previous section.

Important CCS laws which are appealed to in the proof are:

$$\begin{array}{rclrcl} a.\tau.x & = & a.x & x + \tau.x & = & \tau.x \\ x + \mathbf{0} & = & x & (x+y)\backslash K & = & x\backslash K + y\backslash K \\ (a.x)\backslash K & = & a.(x\backslash K) \text{ if } a \notin K \cup \overline{K} & & & \\ (a.x)\backslash K & = & \mathbf{0} \text{ if } a \in K \cup \overline{K} & & & \end{array}$$

The last four are clear from the behavioural meanings of the operators. The first two are important τ-laws. A rule schema, called an *expansion law* by Milner [61], relating concurrency and choice is:

$$\begin{array}{l} \text{if } x = \sum\{a_i.x_i : 1 \le i \le n\} \quad \text{and} \quad y = \sum\{b_j.y_j : 1 \le j \le m\} \quad \text{then} \\ x \mid y = \sum\{a_i.(x_i \mid y) : 1 \le i \le n\} + \sum\{b_j.(x \mid y_j) : 1 \le j \le m\} + \\ \qquad \sum\{\tau.(x_i \mid y_j) : a_i = \overline{b}_j\} \end{array}$$

This is clearly justified by the transition rules for $\mid$. The schema is easily extended to more processes in parallel, $x_1 \mid \ldots \mid x_n$ provided that each x_i is of the form $\sum\{a_j.y_j : 1 \le j \le n_i\}$.

We also need proof rules for recursion. First, if $P \stackrel{\mathrm{def}}{=} E$ then $P = E$. Secondly and more significantly, if $E\{Q/P\} = F$ and $P \stackrel{\mathrm{def}}{=} E$ and $Q \stackrel{\mathrm{def}}{=} F$ and P and Q are guarded[13] in E and F then $P = F$.

The slot machine SM without data values and its specification SM' appear in figure 13. The proof that $SM = SM'$ proceeds using the generalized expansion

$$\begin{array}{ll} IO \stackrel{\mathrm{def}}{=} \mathtt{slot}.IO_1 & IO_1 \stackrel{\mathrm{def}}{=} \overline{\mathtt{bank}}.IO_2 \\ IO_2 \stackrel{\mathrm{def}}{=} \mathtt{lost}.\overline{\mathtt{loss}}.IO + \overline{\mathtt{release}}.\overline{\mathtt{win}}.IO & \\ B \stackrel{\mathrm{def}}{=} \mathtt{bank}.B_1 & B_1 \stackrel{\mathrm{def}}{=} \overline{\mathtt{max}}.\mathtt{left}.B \\ D \stackrel{\mathrm{def}}{=} \mathtt{max}.D_1 & D_1 \stackrel{\mathrm{def}}{=} \overline{\mathtt{lost}}.\overline{\mathtt{left}}.D + \overline{\mathtt{release}}.\overline{\mathtt{left}}.D \end{array}$$

$$\begin{array}{l} SM' \stackrel{\mathrm{def}}{=} \mathtt{slot}.(\tau.\overline{\mathtt{loss}}.SM' + \tau.\overline{\mathtt{win}}.SM') \\ SM \equiv (IO \mid B \mid D)\backslash K \quad \text{where } K = \{\mathtt{bank}, \mathtt{max}, \mathtt{left}, \mathtt{release}\} \end{array}$$

Fig. 13. A simplified slot machine

law and laws for $\backslash K$ and $\mathbf{0}$:

$$\begin{array}{rl} SM & = (\mathtt{slot}.(IO_1 \mid B \mid D) + \mathtt{bank}.(IO \mid B_1 \mid D) + \mathtt{max}.(IO \mid B \mid D_1))\backslash K \\ & = \mathtt{slot}.(IO_1 \mid B \mid D)\backslash K + \mathbf{0} + \mathbf{0} \quad = \quad \mathtt{slot}.(IO_1 \mid B \mid D)\backslash K \end{array}$$

Let $SM_1 \equiv (IO_1 \mid B \mid D)\backslash K$. Similarly, $SM_1 = \tau.\tau.(\tau.SM_2 + \tau.SM_3)$ where

$$SM_2 \equiv (\overline{\mathtt{loss}}.IO \mid \mathtt{left}.B \mid \overline{\mathtt{left}}.D)\backslash K \qquad SM_3 \equiv (\overline{\mathtt{win}}.IO \mid \mathtt{left}.B \mid \overline{\mathtt{left}}.D)\backslash K$$

The τ-laws are used in the following chain of reasoning:

---

[13] P is guarded in E if its occurrences in E are within the scope of a prefix a. operator when a is observable: this then guarantees that the equation $P \stackrel{\mathrm{def}}{=} E$ has a unique solution up to $\approx^c$.

$$
\begin{aligned}
SM_2 &= \overline{\texttt{loss}}.(IO \mid \texttt{left}.B \mid \overline{\texttt{left}}.D)\backslash K + \tau.(\overline{\texttt{loss}}.IO \mid B \mid D)\backslash K \\
&= \overline{\texttt{loss}}.\big(\tau.SM + \texttt{slot}.(IO_1 \mid \texttt{left}.B \mid \overline{\texttt{left}}.D)\backslash K\big) + \tau.\overline{\texttt{loss}}.SM \\
&= \overline{\texttt{loss}}.\,(\tau.SM + \texttt{slot}.\tau.(IO_1 \mid B \mid D)\backslash K) + \tau.\overline{\texttt{loss}}.SM \\
&= \overline{\texttt{loss}}.\,(\tau.SM + \texttt{slot}.(IO_1 \mid B \mid D)\backslash K) + \tau.\overline{\texttt{loss}}.SM \\
&= \overline{\texttt{loss}}.(\tau.SM + SM) + \tau.\overline{\texttt{loss}}.SM \\
&= \overline{\texttt{loss}}.\tau.SM + \tau.\overline{\texttt{loss}}.SM \quad = \quad \tau.\overline{\texttt{loss}}.SM
\end{aligned}
$$

Also $SM_3 = \tau.\overline{\texttt{win}}.SM$. Backtracking and substituting equals for equals gives: $SM = \texttt{slot}.SM_1 = \texttt{slot}.\tau.\tau.(\tau.\tau.\overline{\texttt{loss}}.SM + \tau.\tau.\overline{\texttt{win}}.SM)$. So by the τ-laws $SM = \texttt{slot}.(\tau.\overline{\texttt{loss}}.SM + \tau.\overline{\texttt{win}}.SM)$. Finally, the result follows by the recursion rules (via the substitution of SM' for SM).

3 Temporal Properties

Modal formulas are able to express local capabilities and necessities of processes. However they can not express enduring capabilities (such as action `tick` is always possible) or long term inevitabilities (that `tick` must eventually happen). Such features especially in the guise of safety and liveness properties have been found to be very useful when specifying the behaviour of systems. We show that these properties are (fixed point) solutions to recursive modal equations, and so we add recursion to modal logic to provide a very rich temporal logic.

3.1 Modal properties revisited

A property separates a set of processes into the two disjoint subsets consisting of those with the property and those without: for example, $\langle \texttt{tick} \rangle \texttt{tt}$ divides $\{Cl_1, \texttt{tock}.Cl_1\}$ into the two subsets $\{Cl_1\}$ and $\{\texttt{tock}.Cl_1\}$ when $Cl_1 \stackrel{\text{def}}{=} \texttt{tick}.\texttt{tock}.Cl_1$. Let us introduce a notation for the subset of a non-empty family of processes $\mathcal{E}$ which have the modal property Φ:

$$
\boxed{\, \|\Phi\|^{\mathcal{E}} \stackrel{\text{def}}{=} \{E \in \mathcal{E} \,:\, E \models \Phi\} \,}
$$

Any modal formula Φ partitions $\mathcal{E}$ into the two subsets $\|\Phi\|^{\mathcal{E}}$ and $\mathcal{E} - \|\Phi\|^{\mathcal{E}}$.

The set $\|\Phi\|^{\mathcal{E}}$ is definable directly by induction on the structure of Φ provided that $\mathcal{E}$ obeys a closure condition described later. First for the boolean connectives:

$$
\boxed{
\begin{aligned}
\|\texttt{tt}\|^{\mathcal{E}} &= \mathcal{E} \\
\|\texttt{ff}\|^{\mathcal{E}} &= \emptyset \\
\|\Phi \wedge \Psi\|^{\mathcal{E}} &= \|\Phi\|^{\mathcal{E}} \cap \|\Psi\|^{\mathcal{E}} \\
\|\Phi \vee \Psi\|^{\mathcal{E}} &= \|\Phi\|^{\mathcal{E}} \cup \|\Psi\|^{\mathcal{E}}
\end{aligned}
}
$$

When $\# \in \{[K], \langle K \rangle, [\![\;]\!], \langle\!\langle\;\rangle\!\rangle, [\![\downarrow]\!]\}$ the definition of the subset of processes with the property $\#\Phi$ appeals to process transformers $\|\,\#\,\|^{\mathcal{E}}$ mapping subsets of $\mathcal{E}$ into subsets of $\mathcal{E}$:

$$\boxed{\|\#\Phi\|^{\mathcal{E}} \;=\; \|\,\#\,\|^{\mathcal{E}}\|\Phi\|^{\mathcal{E}}}$$

In the case of $[K]$ and $\langle K \rangle$ these transformers are:

$$\begin{aligned}
\|[K]\|^{\mathcal{E}} &= \lambda X \subseteq \mathcal{E}.\{F \in \mathcal{E} \,:\, \text{if } F \xrightarrow{a} E \text{ and } a \in K \text{ then } E \in X\}\\
\|\langle K \rangle\|^{\mathcal{E}} &= \lambda X \subseteq \mathcal{E}.\{F \in \mathcal{E} \,:\, \exists E \in X.\, \exists a \in K.\; F \xrightarrow{a} E\}
\end{aligned}$$

Of course we would like the direct inductive definition of $\|\Phi\|^{\mathcal{E}}$ to coincide with its indirect definition. But this is only guaranteed for particular sets of processes $\mathcal{E}$, namely *transition closed* sets: a set of processes $\mathcal{E}$ is transition closed if whenever $E \in \mathcal{E}$ and $E \xrightarrow{a} F$ then also $F \in \mathcal{E}$. In the sequel we use $\mathcal{P}$ to range over non-empty transition closed sets. Notice that the set of processes of a transition diagram is transition closed. Therefore we also introduce the notation $\mathcal{P}(E)$ to be the *smallest* transition closed set containing E. If a set is transition closed then it is also closed with respect to the thicker transitions $\stackrel{a}{\Longrightarrow}$ and $\stackrel{\varepsilon}{\Longrightarrow}$. In the following result we assume that $\|\Phi\|^{\mathcal{P}}$ is determined by its inductive definition.

Proposition 1 *If $E \in \mathcal{P}$ then $E \in \|\Phi\|^{\mathcal{P}}$ iff $E \models \Phi$.*

An important feature of modal logic is that its operators are monotonic with respect to subset inclusion. This feature will be exploited when we develop the temporal logic later.

Proposition 2 *If $\mathcal{E} \subseteq \mathcal{F}$ and $\# \in \{[K], \langle K \rangle, [\![\;]\!], \langle\!\langle\;\rangle\!\rangle, [\![\downarrow]\!]\}$ then $\|\Phi\|^{\mathcal{P}} \cap \mathcal{E} \subseteq \|\Phi\|^{\mathcal{P}} \cap \mathcal{F}$, $\|\Phi\|^{\mathcal{P}} \cup \mathcal{E} \subseteq \|\Phi\|^{\mathcal{P}} \cup \mathcal{F}$, and $\|\,\#\,\|^{\mathcal{P}}\,\mathcal{E} \subseteq \|\,\#\,\|^{\mathcal{P}}\,\mathcal{F}$.*

A property on a transition closed set of processes is that subset which has it. Consider the family of clocks $\mathcal{P} = \{Cl^i, Cl \,:\, i \geq 0\}$, where $Cl \stackrel{\text{def}}{=} \texttt{tick}.Cl$ and $Cl^{i+1} \stackrel{\text{def}}{=} \texttt{tick}.Cl^i$ for every $i \geq 0$. What distinguishes Cl from the other members of $\mathcal{P}$ is its long term capability for ticking endlessly. Each Cl^i can only tick i times before stopping. This property divides $\mathcal{P}$ into the two subsets $\{Cl\}$ and $\mathcal{P} - \{Cl\}$. But this feature can not be captured by any single modal formula[14], as the next result shows.

Proposition 3 *For any modal Φ, if $Cl \in \|\Phi\|^{\mathcal{P}}$ then there is a $j \geq 0$ such that for all $k \geq j$, $Cl^k \in \|\Phi\|^{\mathcal{P}}$.*

3.2 Processes and their runs

Proposition 3 of the previous section shows that modal formulas are not very expressive. Although able to describe immediate capabilities and necessities, they

[14] Of course, for each Cl^i there is the formula $\langle\texttt{tick}\rangle^{i+1}\texttt{tt}$ which it fails and which holds of Cl.

can not capture long-standing features of processes. We can contrast the immediate capability for ticking with the enduring capability for ticking forever, and the urgent inevitability that tick must happen next with the lingering inevitability that tick eventually happens.

Another abstraction from behaviour is that of a *run* of a process E_0 which is a finite or infinite length sequence of transitions of the form, $E_0 \xrightarrow{a_0} E_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} E_n \xrightarrow{a_n} \ldots$. If a run has finite length then its final process is unable to perform a transition. Thus a run of E_0 can be viewed as a computation of E_0, a maximal performance of actions.

The difference between Cl and the clocks Cl^i of the previous section is that Cl has a perpetual run consisting wholly of tick actions whereas any run from Cl^i terminates. A capability for ticking forever requires there to be a never-ending run of ticks, and that tick eventually happens depends upon there being a tick transition in every run.

Many significant properties of systems can be understood as features of their runs. Especially important is a classification of properties into *safety* and *liveness*, due to Lamport [53]. A safety property states that *nothing bad ever happens* while a liveness property expresses that *something good does eventually happen*. In terms of runs a safety property is that no run contains the bad feature whereas a liveness property is the characteristic that every run contains the good trait.

Example 1 A property distinguishing each clock Cl^i from Cl is eventual termination. The good characteristic is expressed by the formula $[-]\texttt{ff}$. On the other hand this can also be viewed as defective, as exhaustion of the clock. In which case Cl has the safety property of absence of deadlock which Cl^i fails. □

Example 2 The level crossing of figure 7 should have the crucial safety property that it is never possible for a train and a car to cross at the same time. In terms of runs this means that no run of *Crossing* passes through a process that can immediately perform both $\overline{\texttt{tcross}}$ and $\overline{\texttt{ccross}}$, and so the bad feature is expressed by the formula $\langle\overline{\texttt{tcross}}\rangle\texttt{tt} \wedge \langle\overline{\texttt{ccross}}\rangle\texttt{tt}$. □

Liveness and safety properties of a process concern all its runs. Therefore we can weaken them to features of some runs. A *weak* safety property states that something bad does not happen in some run, and a *weak* liveness property asserts that something good may happen, that it eventually happens in some run.

Example 3 A weak liveness property of the slot machine SM_n is that it may eventually pay out a windfall of a million pounds: the good thing is given by the formula $\langle\overline{\texttt{win}}(10^6)\rangle\texttt{tt}$. □

There is middle ground between all and some runs. Often we are only interested in liveness and safety in the case of a subset of the runs, those that obey a particular condition.

Example 4 A desirable property of the level crossing is that whenever a car approaches eventually it crosses: any run containing the action `car` also contains the later action $\overline{\texttt{ccross}}$. □

Sometimes these conditions are complex and depend on assumptions outwith the possible behaviour of the process itself. For example the protocol of figure 9 fails to have the property that whenever a message is input eventually it is output because of runs where a message is forever lost and is forever retransmitted. However we may assume that the medium must eventually pass a message to the receiver, and that therefore these runs are precluded.

Cyclic properties are also pertinent. The clock Cl_1 of the previous section performs $\texttt{tick}$ immediately followed by $\texttt{tock}$ cyclically starting with $\texttt{tick}$: any run from Cl_1 involves the infinite sequence of actions $(\texttt{tick}\,\texttt{tock})^\omega$.

Example 5 The scheduler of section 1.4 ensures that the sequence of actions $a_1 \ldots a_n$ is performed cyclically starting with a_1. In this example other actions (silent actions and the task termination actions b_i) may be interspersed between any of the pairs $a_i a_{i+1}$. □

The cyclicity of Cl_1 ensures that the number of $\texttt{tock}$ actions never exceeds the number of $\texttt{tick}$ occurrences in any run. This paves the way for richer properties. The process *Cop* from Section 1.1 is a single place buffer with a very simple cyclic behaviour. Consider instead an unbounded buffer. An essential feature is that the number of out actions never exceeds the number of in actions, which need not be achieved cyclically.

Not all properties can be ascribed to process runs. A simple example is that the behaviour graph of the following clock $Cl' \stackrel{\mathrm{def}}{=} \texttt{tick}.Cl' + \texttt{tock}.Cl'$ can be unfolded into a recurring infinite binary tree. Moreover embedding liveness and safety properties within their weaker versions can not be so described.

There is a variety of logics called *temporal* logics for expressing properties of runs directly: see [39, 59, 77] for surveys. Traditional temporal logics studied in philosophy and linguistics are not able to express cyclic properties (and more generally are not able to capture regular behaviour) and so have been extended [86].

In this work we suppress the notion of a run. It is of course a very useful abstraction but not as a primitive. As we noted in section 2.4, processes that may perform the same sequences of completed traces should not be deemed equivalent. Moreover it is not clear what the notion of run should be when silent actions are abstracted away. Instead we note that a run is simply a *subset* of a transition closed set.

All the properties mentioned in this section are not expressible in the modal logics we introduced earlier. These properties capture features of processes that can be quite crucial. We will now find appropriate closure conditions on sets of processes which define them by appealing to *inductive* definitions built out of modal logic. The idea is that a long term capability is just a particular closure of an immediate capability.

3.3 Modal equations and fixed points

Definitional equality, $\stackrel{\text{def}}{=}$, is indispensable for describing perpetual processes as in the simple case of the uncluttered clock Cl that ticks forever. Imagine adding this facility to modal logic, following Larsen [54]. For instance the modal equation $Z \stackrel{\text{def}}{=} \langle \mathtt{tick} \rangle \mathtt{tt}$ stipulates that Z expresses the same property, an ability to perform $\mathtt{tick}$, as $\langle \mathtt{tick} \rangle \mathtt{tt}$. On any transition closed set this characteristic is given by the set $\| \langle \mathtt{tick} \rangle \mathtt{tt} \|$.

A more intricate modal equation is $Z \stackrel{\text{def}}{=} \langle \mathtt{tick} \rangle Z$ where both occurrences of Z select the same trait. Unlike the use of $\stackrel{\text{def}}{=}$ in the definition of a process, a recursive modal equation may express various properties on a set $\mathcal{P}$, each of which is a subset $\mathcal{E}$ of processes obeying the condition $\mathcal{E} = \| \langle \mathtt{tick} \rangle \| \ \mathcal{E}$. Any such solution $\mathcal{E}$ is a *fixed point* of the function $f = \lambda X \subseteq \mathcal{P}.\ \| \langle \mathtt{tick} \rangle \| \ X$ as $f(\mathcal{E}) = \mathcal{E}$. The equality can be bisected: $\mathcal{E}$ is a *prefixed* point of f, that is $f(\mathcal{E}) \subseteq \mathcal{E}$, and it is a *postfixed* point of f which is the other half, $\mathcal{E} \subseteq f(\mathcal{E})$. These halves can be viewed as *closure* conditions on any potential solution set $\mathcal{E}$:

PRE if $F \in \mathcal{E}$ and $E \in \mathcal{P}$ and $E \xrightarrow{\mathtt{tick}} F$ then $E \in \mathcal{E}$
POST if $E \in \mathcal{E}$ then $E \xrightarrow{\mathtt{tick}} F$ for some $F \in \mathcal{E}$

One solution is the empty set as it trivially fulfills both conditions. When $\mathcal{P}$ is $\{Cl\}$, the other subset $\{Cl\}$ also obeys both because of the transition $Cl \xrightarrow{\mathtt{tick}} Cl$. In this instance both candidate solutions are successful fixed points, and they can be ordered by subset, $\emptyset \subseteq \{Cl\}$, offering smallest and largest solutions. In the case that $\mathcal{P}$ is generated by the more sonorous clock $Cl_1 \stackrel{\text{def}}{=} \mathtt{tick}.\mathtt{tock}.Cl_1$ that alternately ticks and tocks, there are more candidates for solutions but besides $\emptyset$ all the rest fail PRE or fail POST.

With respect to any transition closed set the equation $Z \stackrel{\text{def}}{=} \langle \mathtt{tick} \rangle Z$ has both a *least* and a *greatest* solution (which may coincide) relative to the subset ordering. The general result guaranteeing this is due to Tarski and Knaster. It shows that the least solution is the intersection of all prefixed points, of all those subsets obeying PRE, and that the greatest solution is the union of all postfixed points, of all those subsets fulfilling POST. The result applies to arbitrary monotonic functions from subsets of $\mathcal{P}$ to subsets of $\mathcal{P}$. Recall that $2^{\mathcal{P}}$ is just the set of all subsets of $\mathcal{P}$, and that a function $g : 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ is monotonic with respect to $\subseteq$ if $\mathcal{E} \subseteq \mathcal{F}$ implies $g(\mathcal{E}) \subseteq g(\mathcal{F})$.

Proposition 1 *If $g : 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ is monotonic with respect to $\subseteq$ then g*
i. *has a least fixed point with respect to $\subseteq$ given by $\bigcap\{\mathcal{E} \subseteq \mathcal{P} : g(\mathcal{E}) \subseteq \mathcal{E}\}$,*
ii. *has a greatest fixed point with respect to $\subseteq$ given by $\bigcup\{\mathcal{E} \subseteq \mathcal{P} : \mathcal{E} \subseteq g(\mathcal{E})\}$.*

As we saw in Proposition 2 of section 3.1, the transformer $\| \langle K \rangle \|$ is monotonic with respect to $\subseteq$, and therefore Proposition 1 applies to the equation

$Z \stackrel{\text{def}}{=} \langle \texttt{tick} \rangle Z$ with respect to $\mathcal{P}$, showing that it has extremal solutions. Relinquishing the equational format we let $\mu Z.\, \langle \texttt{tick} \rangle Z$ express the property given by the least solution, and we let $\nu Z.\, \langle \texttt{tick} \rangle Z$ express the property determined by the largest solution. But what properties are expressed by these formulas? The least case is of little import as it expresses the same property as $\texttt{ff}$: irrespective of $\mathcal{P}$, the empty set obeys condition PRE. Much more stimulating is that $\nu Z.\, \langle \texttt{tick} \rangle Z$ expresses the long-standing capability for performing the action $\texttt{tick}$ forever. Let $\mathcal{E} \subseteq \mathcal{P}$ consist of all those processes E_0 that have an infinite length run of the form $E_0 \xrightarrow{\texttt{tick}} E_1 \xrightarrow{\texttt{tick}} \ldots$. It is clear that $\mathcal{E}$ obeys POST, and that it is the largest such set. As shown in section 3.1 this capability is *not* expressible within the earlier modal logics. More generally, $\nu Z.\, \langle K \rangle Z$ expresses a capability for performing K actions forever. Two special cases are striking, $\nu Z.\, \langle - \rangle Z$ expresses a capacity for never-ending behaviour and $\nu Z.\, \langle \tau \rangle Z$ captures divergence, the ability to engage in infinite internal chatter.

Proposition 1 applies to any modal equation $Z \stackrel{\text{def}}{=} \Psi$ when Ψ is built from modal operators, boolean connectives, the constants $\texttt{tt}$ and $\texttt{ff}$, and Z: this follows directly from Proposition 2 of section 3.1. A more composite equation schema where Φ does not contain Z is, $Z \stackrel{\text{def}}{=} \Phi \vee \langle K \rangle Z$. Any solution $\mathcal{E} \subseteq \mathcal{P}$ divides into the following two closure conditions:

PRE if $E \in \mathcal{P}$ and ($E \models \Phi$ or $E \xrightarrow{a} F$ for some $a \in K$ and $F \in \mathcal{E}$) then $E \in \mathcal{E}$
POST if $E \in \mathcal{E}$ then $E \models \Phi$ or $E \xrightarrow{a} F$ for some $a \in K$ and $F \in \mathcal{E}$

Every subset $\mathcal{E}$ fulfilling PRE must contain those processes in $\mathcal{P}$ with the property Φ, and also includes those processes that fail Φ, but are able to perform a K action and become a process having Φ, and so on. Therefore a process E_0 has the property $\mu Z. \Phi \vee \langle K \rangle Z$ if it has a run of the form $E_0 \xrightarrow{a_0} E_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} E_n \xrightarrow{a_n} \ldots$ with $E_n \models \Phi$ for some n and where each a_j, $j < n$, belongs to K, that is if E_0 is able to perform K actions *until* Φ holds. The maximal solution also includes the extra possibility of performing K actions forever without Φ ever becoming true.

Two general cases of $\mu Z. \Phi \vee \langle K \rangle Z$ are worth noting. When K is the complete set of actions $\mu Z. \Phi \vee \langle K \rangle Z$ expresses weak liveness, that Φ is eventually true in *some* run, and when K is the singleton set $\{\tau\}$ it expresses that after some silent activity Φ is true, that is $\langle\!\langle\ \rangle\!\rangle \Phi$. Recall that the modality $\langle\!\langle\ \rangle\!\rangle$ is not definable within the modal logic of section 2.1.

Another useful composite schema (assuming again that Φ does not contain Z) is $Z \stackrel{\text{def}}{=} \Phi \wedge \langle K \rangle Z$. The least solution is of no interest as it is expressed by $\texttt{ff}$. The maximal solution over $\mathcal{P}$ is the union of all subsets $\mathcal{E}$ obeying the following condition POST

POST if $E \in \mathcal{E}$ then $E \models \Phi$ and $E \xrightarrow{a} F$ for some $F \in \mathcal{E}$ and $a \in K$

which requires there to be a perpetual run involving only K actions and with Φ true throughout. A slight weakening is that Φ holds throughout a maximal performance of K actions, as expressed by $\nu Z.\Phi \wedge (\langle K\rangle Z \vee [K]\mathtt{ff})$, and when K is the set of all actions it expresses a weak safety property.

The *complement* of weak liveness is safety. A process fails $\mu Z.\Phi \vee \langle -\rangle Z$ if Φ never becomes true in any run. Similarly the complement of weak safety is liveness. A process lacks $\nu Z.\Phi \wedge (\langle -\rangle Z \vee [-]\mathtt{ff})$ if in every run eventually Φ is false. Complements are expressible when negation is freely admitted into formulas with its intended meaning, $\| \neg\Phi \|^{\mathcal{P}}$ is the set of processes $\mathcal{P} - \| \Phi \|^{\mathcal{P}}$. But then not every modal equation has extremal solutions, a simple instance is $Z \stackrel{\mathrm{def}}{=} \neg Z$. It fails the monotonicity requirement of Proposition 1. However if we restrict the form of an equation $Z \stackrel{\mathrm{def}}{=} \Phi$ so that every free occurrence of Z in Φ lies within the scope of an *even* number of negations then monotonicity is guaranteed.

However the complement of a formula is also in the logic without the explicit presence of negation. Let Φ^c be the complement of Φ, which is defined inductively as follows

$$\begin{array}{llll}
\mathtt{tt}^c & = \mathtt{ff} & \mathtt{ff}^c & = \mathtt{tt} \\
(\Phi \wedge \Psi)^c & = \Phi^c \vee \Psi^c & (\Phi \vee \Psi)^c & = \Phi^c \wedge \Psi^c \\
([K]\Phi)^c & = \langle K\rangle \Phi^c & (\langle K\rangle\Phi)^c & = [K]\Phi^c \\
(\nu Z.\Phi)^c & = \mu Z.\Phi^c & (\mu Z.\Phi)^c & = \nu Z.\Phi^c
\end{array}$$

The complement here is very simple, one merely replaces every operator by its dual: for instance $(\langle \mathtt{tick}\rangle\mathtt{tt} \wedge \langle \mathtt{tock}\rangle\mathtt{tt})^c$ is $[\mathtt{tick}]\mathtt{ff} \vee [\mathtt{tock}]\mathtt{ff}$. The fixed point operators are also duals of each other. We assume here that Z is the only free variable and that Z^c is Z. An expected consequence is that $\| \Phi^c \|^{\mathcal{P}} = \mathcal{P} - \| \Phi \|^{\mathcal{P}}$.

If a property is an extremal solution to the equation $Z \stackrel{\mathrm{def}}{=} \Phi$ then its complement is the dual solution to $Z \stackrel{\mathrm{def}}{=} \Phi^c$. Consider convergence $E \downarrow$, which holds if E can not perform silent actions for ever. The formula $\nu Z.\langle\tau\rangle Z$ expresses its complement, divergence, and so convergence is captured by $\mu Z.[\tau]Z$. Generalizing, the formula $\mu Z.[-]Z$ expresses that all behaviour is finite.

A strong invariance property is that Φ holds throughout every ongoing performance of K actions. This fails precisely when a process may perform K actions continuously until Φ^c holds. Failure here is given by the formula $\mu Z.\Phi^c \vee \langle K\rangle Z$ whose complement is therefore $\nu Z.\Phi \wedge [K]Z$. A particular case is that $[\![\ ]\!]\Phi$ is the formula $\nu Z.\Phi \wedge [\tau]Z$. Safety properties, that nothing bad ever happens, fall under this format.

A strong until property is that every continuous performance of K actions eventually leads to the holding of Φ. This is therefore the complement of $\nu Z.\Phi^c \wedge (\langle K\rangle Z \vee [K]\mathtt{ff})$ which is $\mu Z.\Phi \vee ([K]Z \wedge \langle K\rangle\mathtt{tt})$. Liveness properties, that something good must eventually happen, have this form (when K is $\mathcal{A}$). An example is that the slot machine must eventually output (winnings or an indication of loss). However a better description is that whenever a coin is input the slot machine must eventually output, a property expressed using both fixed point operators. Embedding fixed point operators within each other goes beyond the simple equational format which motivated their introduction.

3.4 Modal Mu-Calculus

Instead of appealing to modal equations to express temporal properties we add to modal logic propositional variables ranged over by Z and the extremal *fixed point* operators νZ and μZ. As before assume that K ranges over subsets of $\mathcal{A}$. The formulas of the resulting logic are:

$$\boxed{\Phi \ ::= \ Z \ \mid \ \Phi_1 \wedge \Phi_2 \ \mid \ \Phi_1 \vee \Phi_2 \ \mid \ [K]\Phi \ \mid \ \langle K \rangle \Phi \ \mid \ \nu Z.\Phi \ \mid \ \mu Z.\Phi}$$

The constant formulas $\mathtt{tt}$ and $\mathtt{ff}$ are definable as $\nu Z.\,Z$ and $\mu Z.\,Z$. However when describing properties we will freely use these constants.

In the sequel we let σ range over the set $\{\mu, \nu\}$. A fixed point formula has the form $\sigma Z.\Phi$ in which σZ *binds* free occurrences of Z in Φ, and an occurrence of Z is *free* if it is not within the scope of a binder σZ. We assume that σZ has wider scope than the boolean connectives $\vee$ and $\wedge$. Formulas may contain multiple occurrences of fixed point operators, as in $\nu Z.\,\mu Y.\,\nu X.\,[a]((\langle b \rangle X \wedge Z) \vee [K]Y)$. Also σZ may bind more than one occurrence of Z, as in $\nu Z.\,\langle \mathtt{tick} \rangle Z \wedge \langle \mathtt{tock} \rangle Z$.

By explicitly employing negation we obtain a more succinct version of the logic:

$$\Phi \ ::= \ Z \ \mid \ \neg\Phi \ \mid \ \Phi_1 \wedge \Phi_2 \ \mid \ [K]\Phi \ \mid \ \nu Z.\Phi$$

There is then the restriction, mentioned previously, on formulas of the form $\nu Z.\,\Phi$, that each free occurrence of Z in Φ must lie within the scope of an even number of negations. This condition guarantees monotonicity. The logic with negation slightly extends[15] modal mu-calculus, proposed originally by Kozen [52] (and also Pratt [72]). Its roots lie in more general program logics with extremal fixed points, developed by Park, De Bakker and De Roever, especially when formulated as relational calculi [11, 12, 68]. Kozen developed this logic as a natural extension of propositional dynamic logic. Larsen proposed that Hennessy-Milner logic with fixed points is useful for describing properties of processes [54]. Emerson and Clarke proposed the use of fixed points on top of a temporal logic for expressing properties of concurrent systems [36].

Clearly every formula in the fixed point modal logic above without negation has an equivalent form in the logic with negation. The converse is not quite true because a negated variable, $\neg Z$, does not have an equivalent form in the negation free fragment. However in applications this is not a real restriction. First note that any closed formula in the fixed point logic with negation has an equivalent form. When formulas do have free variables we employ valuations which are described below. Consequently we are free to introduce the understanding that a free variable Y has the meaning of the negation of a different free variable Z.

Assume a fixed transition closed set of processes $\mathcal{P}$. We wish to inductively define when the process $E \in \mathcal{P}$ has the temporal property Φ. The new problematic cases are the clauses for $E \models \sigma Z.\,\Psi$ which not only depend on properties

[15] as sets of labels may appear in the modalities instead of single labels

of sets of processes but also on interpreting the subformula Ψ which may contain free occurrences of Z. So the satisfaction relation $\models$ is defined indirectly, in terms of $\| \Phi \|$, the set of all processes in $\mathcal{P}$ with the property Φ. Subformulas containing free propositional variables are dealt with using *valuations*, functions ranged over by $\mathcal{V}$ which assign to each variable Z a subset $\mathcal{V}(Z)$ of processes in $\mathcal{P}$. A customary updating notation is also assumed: $\mathcal{V}[\mathcal{E}/Z]$ is the valuation $\mathcal{V}'$ which agrees with $\mathcal{V}$ on all variables except Z when $\mathcal{V}'(Z) = \mathcal{E}$.

The subset of processes in $\mathcal{P}$ satisfying an arbitrary formula Ψ relative to the valuation $\mathcal{V}$ is inductively defined as the set $\| \Psi \|_{\mathcal{V}}$ where for ease of notation we drop the superscript $\mathcal{P}$ which is assumed fixed throughout:

$$\boxed{\begin{array}{lcl} \| Z \|_{\mathcal{V}} & = & \mathcal{V}(Z) \\ \| \Phi \wedge \Psi \|_{\mathcal{V}} & = & \| \Phi \|_{\mathcal{V}} \cap \| \Psi \|_{\mathcal{V}} \\ \| \Phi \vee \Psi \|_{\mathcal{V}} & = & \| \Phi \|_{\mathcal{V}} \cup \| \Psi \|_{\mathcal{V}} \\ \| [K]\Phi \|_{\mathcal{V}} & = & \| [K] \| \; \| \Phi \|_{\mathcal{V}} \\ \| \langle K \rangle \Phi \|_{\mathcal{V}} & = & \| \langle K \rangle \| \; \| \Phi \|_{\mathcal{V}} \\ \| \nu Z.\, \Phi \|_{\mathcal{V}} & = & \bigcup \{ \mathcal{E} \subseteq \mathcal{P} \; : \; \mathcal{E} \subseteq \| \Phi \|_{\mathcal{V}[\mathcal{E}/Z]} \} \\ \| \mu Z.\, \Phi \|_{\mathcal{V}} & = & \bigcap \{ \mathcal{E} \subseteq \mathcal{P} \; : \; \| \Phi \|_{\mathcal{V}[\mathcal{E}/Z]} \subseteq \mathcal{E} \} \end{array}}$$

The subset of $\mathcal{P}$ with the property Z is that stipulated by the function $\mathcal{V}$. The semantic clauses for the boolean operators are as in section 3.1, except for the additional valuation component for understanding free variables. The meanings of the modal operators appeal to the transformers $\| [K] \|$ and $\| \langle K \rangle \|$ defined in section 3.1. (The derived clauses for the boolean constants are, $\| \mathtt{tt} \|_{\mathcal{V}} = \mathcal{P}$ and $\| \mathtt{ff} \|_{\mathcal{V}} = \emptyset$.)

It is straightforward to show that any formula Φ determines the monotonic function $\lambda \mathcal{E} \subseteq \mathcal{P}.\ \| \Phi \|_{\mathcal{V}[\mathcal{E}/Z]}$ with respect to the variable Z, the valuation $\mathcal{V}$, and the transition closed set $\mathcal{P}$. Hence the meanings of the fixed point formulas are instances of Proposition 1 of section 3.3: the greatest fixed point is given as the union of all postfixed points whereas the least fixed point is the intersection of all prefixed points. One consequence is that the meaning of $\sigma Z.\Phi$ is the same as its *unfolding* $\Phi\{\sigma Z.\Phi/Z\}$.

In the case of a closed formula[16] Φ, the subset $\| \Phi \|_{\mathcal{V}}$ is independent of the particular valuation $\mathcal{V}$, and so is the same as $\| \Phi \|_{\mathcal{V}'}$ for any other valuation $\mathcal{V}'$. Therefore when Φ is closed we let $\| \Phi \|$ be the subset of processes of $\mathcal{P}$ with the temporal property Φ relative to an arbitrary valuation, and we also use the notation $E \models \Phi$ to mean that $E \in \| \Phi \|$. More generally when Φ may contain free variables we write $E \models_{\mathcal{V}} \Phi$ whenever $E \in \| \Phi \|_{\mathcal{V}}$.

Example 1 Entangled fixed point formulas can be difficult to understand. Assume that D and D' are the following two processes: $D \stackrel{\rm def}{=} a.D'$ and $D' \stackrel{\rm def}{=} b.\mathbf{0} + a.D$. Let $\mathcal{P}$ be the transition closed set $\{D, D', \mathbf{0}\}$. $\| \nu Z.\, \mu Y.\, [a]((\langle b \rangle \mathtt{tt} \wedge Z) \vee Y) \|$ is $\mathcal{P}$, and $\| \mu Y.\, \nu Z.\, [a]((\langle b \rangle \mathtt{tt} \vee Y) \wedge Z) \|$ is $\{\mathbf{0}\}$. $\Box$

In section 2.2 we introduced further modal operators which are not definable in the modal logic of Section 2.1, namely $\langle\!\langle \ \rangle\!\rangle$, $[\![\ ]\!]$, and $[\![\downarrow]\!]$. In the presence of

[16] that is, a formula without free variables.

fixed points these modalities *are* definable as follows (where we asssume that Z is not free in Φ):

$$\boxed{\begin{array}{l} \langle\!\langle\ \rangle\!\rangle \Phi \stackrel{\mathrm{def}}{=} \mu Z.\, \Phi \vee \langle \tau \rangle Z \\ [\![\ ]\!] \Phi \stackrel{\mathrm{def}}{=} \nu Z.\, \Phi \wedge [\tau] Z \\ [\![\downarrow]\!] \Phi \stackrel{\mathrm{def}}{=} \mu Z.\, \Phi \wedge [\tau] Z \end{array}}$$

Therefore those derived modalities $\langle\!\langle K \rangle\!\rangle$, $[\![K]\!]$, $[\![\downarrow K]\!]$ are also definable. For instance, $[\![K]\!]\Phi$ was defined as $[\![\ ]\!]\,[K]\,[\![\ ]\!]\,\Phi$ which is the fixed point formula $\nu Z.\, [K](\nu Y.\, \Phi \wedge [\tau] Y) \wedge [\tau] Z$. Weak modal mu-calculus is the sublogic when the modalities are restricted to the subset $\{[\![\ ]\!], \langle\!\langle\ \rangle\!\rangle, [\![K]\!], \langle\!\langle K \rangle\!\rangle\}$ (and when $\tau \notin K$). This fixed point logic is an appropriate for expressing properties of weak transition systems (when τ is not observable).

An important feature of modal mu-calculus is that it has the *finite model property*: if a formula holds of a process then there is a finite state process satisfying it. A proof of this can be found in [79].

3.5 Approximants

At first sight there is a chasm between the meaning of a formula and techniques (other than exhaustive analysis) for actually finding the set of processes it defines. Example 1 of the previous section illustrates the delicacy of obtaining correct fixed points. There is however a more mechanical method, an iterative technique, due to Tarski and others, for discovering extremal fixed points. Let μg be the least and νg the greatest fixed point of the monotonic function g mapping subsets of $\mathcal{P}$ to subsets of $\mathcal{P}$.

Suppose we wish to determine the set νg, which we know is the union of all subsets $\mathcal{E}$ that obey $\mathcal{E} \subseteq g(\mathcal{E})$. Let $\nu^0 g$ be the full set $\mathcal{P}$, and let $\nu^{i+1} g$ be $g(\nu^i g)$. Clearly $g(\nu^0 g) \subseteq \nu^0 g$, that is $\nu^1 g \subseteq \nu^0 g$, and by monotonicity of g this implies that $g(\nu^1 g) \subseteq g(\nu^0 g)$, that is $\nu^2 g \subseteq \nu^1 g$. Consequently by repeated application of g it follows that $g(\nu^i g) \subseteq \nu^i g$ for each i, and so there is the possibly decreasing sequence of sets, $\nu^0 g \supseteq \nu^1 g \supseteq \ldots \supseteq \nu^i g \ldots$. The required set νg is a *subset* of each member of this sequence[17]. If $\nu^i g$ is equal to $\nu^{i+1} g$ then $\nu^i g$ is the set νg (and therefore also $\nu^j g$ is νg for every $j \geq i$). These observations suggest a strategy for discovering νg. Iteratively construct the sets $\nu^i g$ starting from $i = 0$, until $\nu^i g$ is the same as its successor $\nu^{i+1} g$. When $\mathcal{P}$ is a finite set containing n processes this iteration must terminate at, or before, $i = n$, and therefore νg is equal to $\nu^n g$.

Example 1 Let $\mathcal{P}$ be $\{Cl, \mathtt{tick}.\mathbf{0}, \mathbf{0}\}$, and let g be $\lambda \mathcal{E} \subseteq \mathcal{P}.\ \| \langle \mathtt{tick} \rangle Z \|_{\mathcal{V}[\mathcal{E}/Z]}$:

[17] By definition $\nu g \subseteq \nu^0 g$, and therefore $g^n(\nu g) \subseteq g^n(\nu^0 g)$ for any n, where $g^n(x)$ is the application of g to x n times. As νg is a fixed point, $g^n(\nu g) = \nu g$, and $g^n(\nu^0 g)$ is the set $\nu^n g$.

$$\begin{aligned}
\nu^0 g &= \mathcal{P} && = \{Cl, \mathtt{tick}.\mathbf{0}, \mathbf{0}\}\\
\nu^1 g &= \|\langle\mathtt{tick}\rangle Z\|_{\mathcal{V}[\nu^0 g/Z]} &&= \{Cl, \mathtt{tick}.\mathbf{0}\}\\
\nu^2 g &= \|\langle\mathtt{tick}\rangle Z\|_{\mathcal{V}[\nu^1 g/Z]} &&= \{Cl\}\\
\nu^3 g &= \|\langle\mathtt{tick}\rangle Z\|_{\mathcal{V}[\nu^2 g/Z]} &&= \{Cl\}
\end{aligned}$$

Stabilization occurs at the stage $\nu^2 g$ as this set is the same as $\nu^3 g$, and consequently is the same as $\nu^n g$ for all $n \geq 2$. Therefore νg is the singleton set $\{Cl\}$. □

When $\mathcal{P}$ is not a finite set of processes, we can still guarantee that νg is reachable by invoking ordinals as indices: $0, 1, \ldots, \omega, \omega{+}1, \ldots, \omega{+}\omega, \omega{+}\omega{+}1, \ldots$. Here ω is the initial *limit* ordinal (one without an immediate predecessor) while $\omega + 1$ is its successor[18]. The set $\nu^{\alpha+1} g$ is defined as $g(\nu^\alpha g)$ and $\nu^\lambda g$ when λ is a limit ordinal is $\bigcap\{\nu^\alpha g \,:\, \alpha < \lambda\}$. Therefore there is the possibly decreasing sequence $\nu^0 g \supseteq \ldots \supseteq \nu^\omega g \supseteq \nu^{\omega+1} g \supseteq \ldots$. The set νg is not only a subset of each member of this sequence, but also appears somewhere within it, and the first such point is not when the ordinal is a limit.

Example 2 Let $\mathcal{P}$ be the set $\{C, B_i \,:\, i \geq 0\}$ when C is the cell $C \stackrel{\text{def}}{=} \mathtt{in}(x).B_x$ with $x : \mathbb{N}$ and $B_{n+1} \stackrel{\text{def}}{=} \mathtt{down}.B_n$ for each n. Let g be $\lambda\mathcal{E} \subseteq \mathcal{P}.\ \|\langle-\rangle Z\|_{\mathcal{V}[\mathcal{E}/Z]}$. The fixed point νg is the empty set. (The formula $\nu Z.\,\langle-\rangle Z$ expresses a capability for infinite behaviour which every member of $\mathcal{P}$ lacks.) For each $j \geq 0$ the set $\nu^{j+1} g$ is $\{C, B_i \,:\, i \geq j+1\}$ and so in the case of the initial limit ordinal $\nu^\omega g$ is $\bigcap\{\nu^i g \,:\, i < \omega\}$ which is the singleton $\{C\}$, as each B_j fails to be in it. The next iterate is the fixed point: $\nu^{\omega+1} g = \|\langle-\rangle Z\|_{\mathcal{V}[\nu^\omega g/Z]}$ which is $\emptyset$. □

The situation for the least fixed point μg is dual. The required set is the intersection of all prefixed points, of all subsets $\mathcal{E} \subseteq \mathcal{P}$ with the feature that $g(\mathcal{E}) \subseteq \mathcal{E}$. Assume that $\mu^0 g$ is the empty set, and that $\mu^{i+1} g$ is the set $g(\mu^i g)$. Therefore there is a possibly increasing sequence of sets $\mu^0 g \subseteq \mu^1 g \subseteq \ldots \subseteq \mu^i g \ldots$ and μg is now a *superset* of each of $\mu^i g$. Again if $\mu^i g$ is equal to its successor $\mu^{i+1} g$ then $\mu^i g$ is the required fixed point μg. An iterative method for finding μg is to construct the set $\mu^i g$ starting with $\mu^0 g$ until it is the same as its successor. Again when $\mathcal{P}$ is finite and consists of n processes this iteration has to terminate at, or before, $\mu^n g$.

Example 3 Let g be $\lambda\mathcal{E} \subseteq \mathcal{P}.\ \|\,[\mathtt{tick}]\mathtt{ff} \vee \langle-\rangle Z\,\|_{\mathcal{V}[\mathcal{E}/Z]}$ when $\mathcal{P}$ be as in example 1.

$$\begin{aligned}
\mu^0 g &= \emptyset\\
\mu^1 g &= \|[\mathtt{tick}]\mathtt{ff} \vee \langle-\rangle Z\|_{\mathcal{V}[\mu^0 g/Z]} = \{\mathbf{0}\}\\
\mu^2 g &= \|[\mathtt{tick}]\mathtt{ff} \vee \langle-\rangle Z\|_{\mathcal{V}[\mu^1 g/Z]} = \{\mathtt{tick}.\mathbf{0}, \mathbf{0}\} \quad = \quad \mu^3 g
\end{aligned}$$

[18] All the processes considered in this work belong to a countable transition closed set $\mathcal{P}$, and so we only need to consider those ordinals whose cardinality is at most that of $\mathbb{N}$.

Stabilization occurs at $\mu^2 g$. $\square$

If $\mathcal{P}$ is not a finite set then we may need to invoke larger ordinals as indices. The set $\mu^{\alpha+1}g$ is $g(\mu^\alpha g)$, and $\mu^\lambda g$ is the union set $\bigcup\{\mu^\alpha g\ :\ \alpha < \lambda\}$ when λ is a limit ordinal. There is the possibly increasing sequence $\mu^0 g \subseteq \ldots \subseteq \mu^\omega g \subseteq \mu^{\omega+1} g \subseteq \ldots$. The set μg is not only a superset of each member of this sequence but also it occurs within it, and the first such time is not when the ordinal is a limit.

Example 4 Let $Cl' \stackrel{\text{def}}{=} \sum\{Cl^i\ :\ i \geq 0\}$ and $Cl^{i+1} \stackrel{\text{def}}{=} \mathtt{tick}.Cl^i$ for each i. The clock Cl' describes an arbitrary new clock, which will eventually break down. Let $\mathcal{P}$ be the set $\{Cl', Cl^i\ :\ i \geq 0\}$. As all behaviour is finite each process in $\mathcal{P}$ has the property $\mu Z.\,[\mathtt{tick}]Z$. Let g be the function $\lambda\mathcal{E} \subseteq \mathcal{P}.\ \|\,[\mathtt{tick}]Z\,\|_{\mathcal{V}[\mathcal{E}/Z]}$. The set $\mu^{i+1}g$ is $\{Cl^j\ :\ j < i+1\}$, and so in the limit $\mu^\omega g$ is $\bigcup\{\mu^i g\ :\ i < \omega\}$ which is $\{Cl^j\ :\ j \geq 0\}$. At the next iteration the required fixed point is reached: $\mu^{\omega+1}g = \mathcal{P}$. $\square$

The sets $\sigma^\alpha g$ are *approximants* for σg in that they verge towards it. Each $\nu^\alpha g$ approximates νg from above while $\mu^\alpha g$ approximates μg from below. In this way an extremal fixed point is the limit of a sequence of approximants. We now provide a more syntactic characterization of these fixed point sets in the extended modal logic M_∞ of section 2.5. If g is $\lambda\mathcal{E} \subseteq \mathcal{P}.\ \|\,\Phi\,\|_{\mathcal{V}[\mathcal{E}/Z]}$ then νg is the set $\|\,\nu Z.\Phi\,\|_{\mathcal{V}}$ and μg is $\|\,\mu Z.\Phi\,\|_{\mathcal{V}}$ (both with respect to $\mathcal{P}$). The initial approximant $\nu^0 g$ is just $\|\,\mathtt{tt}\,\|_{\mathcal{V}}$ and $\mu^0 g$ is $\|\,\mathtt{ff}\,\|_{\mathcal{V}}$. Therefore $\nu^1 g$ is $\|\,\Phi\,\|_{\mathcal{V}[\|\mathtt{tt}\|_{\mathcal{V}}/Z]}$ which is the set $\|\,\Phi\{\mathtt{tt}/Z\}\,\|_{\mathcal{V}}$: similarly $\mu^1 g$ is $\|\,\Phi\{\mathtt{ff}/Z\}\,\|_{\mathcal{V}}$. For each ordinal α we define $\sigma Z^\alpha.\Phi$ as a formula of the extended modal logic. As before let λ be a limit ordinal:

$$\begin{array}{llll}
\nu Z^0.\Phi & = \mathtt{tt} & \mu Z^0.\Phi & = \mathtt{ff} \\
\nu Z^{\alpha+1}.\Phi & = \Phi\{\nu Z^\alpha.\Phi/Z\} & \mu Z^{\alpha+1}.\Phi & = \Phi\{\mu Z^\alpha.\Phi/Z\} \\
\nu Z^\lambda.\Phi & = \bigwedge\{\nu Z^\alpha.\Phi\ :\ \alpha < \lambda\} & \mu Z^\lambda.\Phi & = \bigvee\{\mu Z^\alpha.\Phi\ :\ \alpha < \lambda\}
\end{array}$$

Proposition 1 *If g is $\lambda\mathcal{E} \subseteq \mathcal{P}.\ \|\,\Phi\,\|_{\mathcal{V}[\mathcal{E}/Z]}$ then $\sigma^\alpha g\ =\ \|\,\sigma Z^\alpha.\Phi\,\|_{\mathcal{V}}$ for all ordinals α.*

A simple consequence of this is that we can now give a more direct definition of $E \models_{\mathcal{V}} \Phi$ when Φ is a fixed point formula:

$$\boxed{\begin{array}{lll}
E \models_{\mathcal{V}} \nu Z.\Phi & \text{iff} & E \models_{\mathcal{V}} \nu Z^\alpha.\Phi \text{ for all } \alpha. \\
E \models_{\mathcal{V}} \mu Z.\Phi & \text{iff} & E \models_{\mathcal{V}} \mu Z^\alpha.\Phi \text{ for some } \alpha.
\end{array}}$$

Example 5 Let $\mathcal{P} = \{D, D', \mathbf{0}\}$ where D and D' are as in example 1 of the previous section, and let $\nu Z.\Psi$ be the formula $\nu Z.\,\mu Y.\,[a]((\langle b\rangle\mathtt{tt} \wedge Z) \vee Y)$ and $\mu Y.\Psi'$ be $\mu Y.\,\nu Z.\,[a]((\langle b\rangle\mathtt{tt} \vee Y) \wedge Z)$. D has the property $\nu Z.\Psi$ and D' fails $\mu Y.\Psi'$. As $|\mathcal{P}|$ is 3, stabilization occurs at or before approximant 3, and therefore $D \models \nu Z.\Psi$ iff $D \models \bigwedge\{\nu Z^i.\Psi\ :\ i \leq 3\}$. The formula $\nu Z^1.\Psi$ is $\mu Y.\,[a]((\langle b\rangle\mathtt{tt} \wedge \mathtt{tt}) \vee Y)$ which is equivalent to $\mu Y.\,[a](\langle b\rangle\mathtt{tt} \vee Y)$, and so $D \models \nu Z^1.\Psi$ iff $D \models \bigvee\{\mu Y^i.\,[a](\langle b\rangle\mathtt{tt} \vee Y)\ :\ i \leq 3\}$. Two of the approximants here are:

$$\begin{aligned}\mu Y^1.\,[a](\langle b\rangle\mathtt{tt}\vee Y) = [a](\langle b\rangle\mathtt{tt}\vee\mathtt{ff}) &= [a]\langle b\rangle\mathtt{tt}\\ \mu Y^2.\,[a](\langle b\rangle\mathtt{tt}\vee Y) = [a](\langle b\rangle\mathtt{tt}\vee[a]\langle b\rangle\mathtt{tt})&\end{aligned}$$

Clearly, $D \models [a]\langle b\rangle\mathtt{tt}$ and so D satisfies $\nu Z^1.\,\Psi$. Next, $\nu Z^2.\,\Psi$ is $\mu Y.\,[a]((\langle b\rangle\mathtt{tt}\wedge\nu Z^1.\,\Psi)\vee Y)$. So $D \models \nu Z^2.\,\Psi$ iff $D \models \bigvee\{\mu Y^i.\,[a]((\langle b\rangle\mathtt{tt}\wedge\nu Z^1.\,\Psi)\vee Y)\ :\ i\leq 3\}$. Now $D \models \mu Y^1.\,[a]((\langle b\rangle\mathtt{tt}\wedge\nu Z^1.\,\Psi)\vee Y)$ as $D \models [a](\langle b\rangle\mathtt{tt}\wedge\nu Z^1.\,\Psi)$. The argument that $D \models \nu Z^3.\,\Psi$ is similar.

To show that $D' \not\models \mu Y.\Psi'$ we need to establish that $D' \not\models \mu Y^i.\Psi'$ for all $i \leq 3$, and by definition D' fails it when i is 0. The next approximant $\mu Y^1.\Psi'$ is $\nu Z.\,[a]((\langle b\rangle\mathtt{tt}\vee\mathtt{ff})\wedge Z)$ which is equivalent to $\nu Z.\,[a](\langle b\rangle\mathtt{tt}\wedge Z)$. Clearly, D' fails $\nu Z^1.\,[a](\langle b\rangle\mathtt{tt}\wedge Z)$ as $D' \not\models [a]\langle b\rangle\mathtt{tt}$. So D' fails $\mu Y^1.\Psi'$. The other two cases are left as an exercise. □

When $\mathcal{P}$ is a finite set a crude measure of how much work needs to be done to compute an extremal fixed point is the number of approximants needed before reaching it. If the size of $\mathcal{P}$ is n then at most n are needed. However if we start with better initial approximations then less work will need to be done. See [57] for details of how this can be done in the case of formulas with multiple fixed points.

3.6 Preservation of bisimulation equivalence

Modal logic characterizes strong bisimulation equivalence as was shown in section 2.5. There are two halves to this result. First two bisimilar processes have the same modal properties (even when enriched with the modalities of section 2.2). Second two image-finite processes with the same modal properties are bisimilar. As modal logic is merely a sublogic of modal mu-calculus the second of these results remains true for this richer logic (and indeed for any extension of modal logic).

We may wonder if the restriction to image-finite processes is still necessary for this result given that the fixed points are expressible using infinitary conjunction and disjunction, and that infinitary modal logic characterizes bisimulation equivalence exactly. Recall the example of the two clocks in section 2.5 that showed that image-finiteness is essential. The clocks have the same modal properties but are not bisimilar. The presence of fixed points allows us to distinguish them as one of them has an infinite tick capability expressed by the formula $\nu Z.\,\langle\mathtt{tick}\rangle$ which the other lacks. The following more complex example due to Roope Kaivola shows the need for image-finiteness, or a weakened version of it, for modal mu-calculus. Let $\{Q_i\ :\ i\in I\}$ be the set of all finite state processes whose actions belong to $\{a,b\}$, and $P(n) \stackrel{\mathrm{def}}{=} a^n.b.P(n+1)$ and $P \stackrel{\mathrm{def}}{=} P(1)+R$ where $R \stackrel{\mathrm{def}}{=} \sum\{a.Q_i\ :\ i\in I\}$. Clearly, P and R are not bisimilar as this would require a finite state Q_j that is bisimilar to $b.P(2)$. But this is not possible (via the pumping lemma for regular languages). On the other hand, P and R have the same properties in Γ.

It also turns out that two bisimilar processes have the same modal mu-calculus properties provided they are expressed using *closed* formulas. Assume that Γ is the set of closed formulas, and recall that $E \equiv_\Gamma F$ means that E and F have the same properties drawn from Γ. Consequently, if $E \sim F$ then $E \equiv_\Gamma F$. Notice the significance of this result. Bisimilar processes not only have the same safety properties but also the same liveness, cyclic, and fairness properties when expressed using a closed formula. There is an indirect proof of this via M_∞, as modal mu-calculus is a sublogic of it. However here we will prove it directly. For we wish to expose some of the inductive structure of modal mu-calculus.

Let $\mathcal{P}$ be a fixed transition closed set of processes. A subset $\mathcal{E}$ of $\mathcal{P}$ is *bisimulation closed* if it obeys the condition: if $E \in \mathcal{E}$ and $F \in \mathcal{P}$ and $E \sim F$ then $F \in \mathcal{E}$. The main result is equivalent to the claim that for any closed Φ and set $\mathcal{P}$ the subset $\| \Phi \|$ is bisimulation closed. The next lemma states some straightforward features of bisimulation closure.

Lemma 1 *If $\mathcal{E}$ and $\mathcal{F}$ are bisimulation closed subsets of $\mathcal{P}$ then the sets $\mathcal{E} \cap \mathcal{F}$, $\mathcal{E} \cup \mathcal{F}$, $\| [K] \|^{\mathcal{P}} \mathcal{E}$, and $\| \langle K \rangle \|^{\mathcal{P}} \mathcal{E}$ are also bisimulation closed.*

Associated with any subset $\mathcal{E}$ of $\mathcal{P}$ are the following two subsets:

$$\begin{aligned} \mathcal{E}^d &= \{E \in \mathcal{E} : \text{ if } E \sim F \text{ and } F \in \mathcal{P} \text{ then } F \in \mathcal{E}\} \\ \mathcal{E}^u &= \{E \in \mathcal{P} : \ E \sim F \text{ and } F \in \mathcal{E}\} \end{aligned}$$

The set $\mathcal{E}^d$ is the *largest* bisimulation closed subset of $\mathcal{E}$ and $\mathcal{E}^u$ is the *smallest* bisimulation closed superset of $\mathcal{E}$, both with respect to $\mathcal{P}$.

Lemma 2 *For any subsets $\mathcal{E}$ and $\mathcal{F}$ of $\mathcal{P}$, the sets $\mathcal{E}^d$ and $\mathcal{E}^u$ are bisimulation closed and $\mathcal{E}^d \subseteq \mathcal{E} \subseteq \mathcal{E}^u$. Moreover if $\mathcal{E}$ is bisimulation closed then $\mathcal{E}^d = \mathcal{E}^u$, and if $\mathcal{E} \subseteq \mathcal{F}$ then $\mathcal{E}^d \subseteq \mathcal{F}^d$ and $\mathcal{E}^u \subseteq \mathcal{F}^u$.*

A valuation $\mathcal{V}$ mapping variables to subsets of $\mathcal{P}$ is bisimulation closed if for each variable Z the set $\mathcal{V}(Z)$ is bisimulation closed. Therefore we can associate the two important bisimulation closed valuations $\mathcal{V}^d$ and $\mathcal{V}^u$ with any valuation $\mathcal{V}$: for any variable Z the set $\mathcal{V}^d(Z) = (\mathcal{V}(Z))^d$ and $\mathcal{V}^u(Z) = (\mathcal{V}(Z))^u$. The desired result is a corollary of the following (when Φ may contain free variables).

Proposition 1 *If $\mathcal{V}$ is bisimulation closed then $\| \Phi \|_{\mathcal{V}}$ is bisimulation closed.*

This result tells us more than that bisimilar processes have the same properties when expressed using a closed formula. They also have the same properties expressed by open formulas provided that the meaning of free variables preserves bisimulation equivalence.

The proof of Proposition 1 also establishes that *weak* modal mu-calculus closed formulas (built from the modal operators $[\![K]\!]$, $[\![\ ]\!]$, $\langle\!\langle K \rangle\!\rangle$, and $\langle\!\langle\ \rangle\!\rangle$ with fixed points, and booleans) are preserved by the weak bisimulation equivalence.

3.7 Expressing properties

The expressive power of modal mu-calculus is intimately related to finite-state automata on infinite trees which is equiexpressive to the monadic second order theory of n successors [37, 49]. This is a very general and fundamental decidable theory to which many other decidability results in logic can be reduced [73]. Most propositional temporal and modal logics which are used in computer science are sublogics of mu-calculus, [52, 33, 77]. In this section we are concerned with the expression of properties of processes.

Safety and liveness properties can relate to *states* (process terms) or *actions*. In the former case we can think of safety as some bad state is never reached, such as a deadlocked process. If Φ is false only at these bad states then $\nu Z.\, \Phi \wedge [-]Z$ expresses this property.

Example 1 The safety property for the crossing of figure 7 is that it is never possible to reach a state where a train and a car are both able to cross. So the required safety property is $\nu Z.([\mathtt{tcross}]\mathtt{ff} \vee [\mathtt{ccross}]\mathtt{ff}) \wedge [-]Z$. □

Example 2 A safety property for the slot machine in figure 10 is that it never has a negative amount of money and therefore never pays out more than it contains. To express this we appeal to an open formula with a free variable Q_0. We assume a valuation $\mathcal{V}$ which assigns the set $\mathcal{P} - \{SM_j \,:\, j < 0\}$ to Q_0. Thus a bad state is one that fails the property Q_0. So this safety property is represented by $\nu Z.Q_0 \wedge [-]Z$ relative to $\mathcal{V}$. □

Safety can also be ascribed to actions, that a bad action a can never happen. This is represented by the formula $\nu Z.\, [a]\mathtt{ff} \wedge [-]Z$. But here there is no real distinction between action and state as this is equivalent to the property that a bad state obeying $\langle a\rangle\mathtt{tt}$ can never be reached. However there appears to be a distinction in the case of liveness. That a good state fulfilling Φ must eventually be reached is expressed by the formula $\mu Z.\, \Phi \vee (\langle -\rangle\mathtt{tt} \vee [-]Z)$. In contrast that *eventually action a happens* is given by the formula $\mu Z.\, \langle -\rangle\mathtt{tt} \wedge [-a]Z$ which says that performance of actions other than a is well founded, but not because of termination.

Liveness properties may be conditional on other actions happening first (or on prior states). For instance consider the property: if a ever happens then eventually b will happen later. That is, any run containing an a action must contain a later b action. This is expressed by the formula $\nu Z.\, [a](\mu Y.\, \langle -\rangle\mathtt{tt} \wedge [-b]Y) \wedge [-]Z$. Let this formula be $\Phi(a,b)$. Even more complex is the expression of liveness when fairness is assumed. For instance, the property that in any run if c and d happen infinitely often then so does e is given by the formula $\nu Z.\, \Psi \wedge [-]Z$ when Ψ is $\mu X.\, [c](\nu Y.\, [d](\nu Y_1.\, X \wedge [-e]Y_1) \wedge [-e]Y)$.

Example 3 The desirable liveness property for the crossing is that whenever a car approaches then eventually it crosses (and similarly for a train) is captured by the formula $\Phi(a,b)$ above when a is $\mathtt{car}$ and b is $\overline{\mathtt{ccross}}$. But this only holds if we assume that the signal is fair. So let Q be a propositional variable which is true when the crossing is in any state where *Rail* has the form $\mathtt{green}.\overline{\mathtt{tcross}}.\overline{\mathtt{red}}.Rail$ and let R hold when it is in a state where $Road$ has the form $\mathtt{up}.\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.Road$. The variables Q and R are probes in the sense of [82]. The liveness property is: for any run if Q is false infinitely often and R is false infinitely often then whenever a car approaches eventually it crosses. This is expressed by the open formula $\nu Y.[\mathtt{car}]\Psi_1 \wedge [-]Y$ when Ψ_1 is $\mu X.\nu Y_1.(Q \vee [-\overline{\mathtt{ccross}}](\nu Y_2.(R \vee X) \wedge [-\overline{\mathtt{ccross}}]Y_2)) \wedge [-\overline{\mathtt{ccross}}]Y_1$. □

Cyclic properties can also be captured. The clock $Cl_1 \stackrel{\mathrm{def}}{=} \mathtt{tick}.\mathtt{tock}.Cl_1$ must perform the action $\mathtt{tick}$ at every odd moment, $\nu Z.\,([-\mathtt{tick}]\mathtt{ff} \wedge \langle - \rangle \mathtt{tt}) \wedge [-][-]Z$. Weaker cyclic properties that allow other actions to intervene within a cycle can also be expressed. Recall the scheduler example of section 1.4. It has to ensure that the actions $a_1, \ldots, a_n$ happen cyclically starting with a_1, and that other actions may be interspersed between any pair $a_i\, a_{i+1}$. Such a property can be defined inductively as follows where the x_i range over distinct actions:

$$cycle(x_1 \ldots x_n) \stackrel{\mathrm{def}}{=} [x_1]\,cycle(x_2 \ldots x_n x_1) \wedge [-x_1]\,cycle(x_1 \ldots x_n) \wedge [-x_2, \ldots, x_n]\mathtt{ff}$$

A least fixed point interpretation here guarantees that the actions x_i will repeatedly happen (unlike the greatest fixed point interpretation which permits the intervening actions to happen forever). So this gives the property

$$cycle(x_1 \ldots x_n) \;= \mu Y.\,[x_1]\,cycle(x_2 \ldots x_n x_1) \wedge [-x_1]Y \wedge [-x_2 \ldots x_n]\mathtt{ff}$$

There are also many properties that are *not* expressible within the logic. A notable case is the following safety property of a buffer: it is never the case that the number of `out` actions exceeds the number of `in` actions. However a careful utilization of atomic sentences over sequences of transitions allows us to express such properties within the framework developed here.

4 Verifying Temporal Properties

A very rich temporal logic has been introduced which is able to describe useful properties of processes such as liveness, safety, cyclic, and fairness. The next step is to provide techniques for *verifying* that they have these features.

To show that a process has (or fails to have) a modal property we can appeal to the inductive definition of satisfaction between individual processes and formulas. This technique has the important feature that a proof of a property reduces to proofs of sub-properties via the inductive definition. A second key feature of modal property checking is that the complete transition graph of a process may not be needed as properties often just describe a part of the behaviour of a process.

However in the case of a temporal property the satisfaction relation is defined indirectly from *all* the processes in a transition closed set which have the property. Therefore a method for checking whether process E has Φ is first calculate a transition closed set containing E, next calculate $\| \Phi \|_{\mathcal{V}}$ with respect to this set, and finally check whether E belongs to $\| \Phi \|_{\mathcal{V}}$. This seems reasonable for determining whether a small process has a property. As a general method it is cumbersome and not feasible for processes that determine enormous let alone infinite-state transition graphs. Moreover picking out all processes in a transition closed set which have a weak liveness or safety property may involve considerable redundancy if the intention is to show that a *particular* process has the property: a case in point is the weak liveness feature of the slot machine that it may eventually pay out a windfall.

An alternative approach to showing that processes satisfy formulas is to appeal to their approximants, as described in section 3.5. Proofs will now require the use of induction over ordinals, and some care must be taken with limit ordinals. Moreover in the presence of embedded fixed points this requires embedded inductions too.

Discovering fixed point sets in general is not easy, and is therefore liable to lead to errors. Instead we would like simpler (and consequently safer) methods for checking temporal properties. Towards this end we first provide a different characterization of the satisfaction relation between a process and a formula in terms of games.

4.1 Games and constants

A property checking game is a pair (E, Φ) relative to a valuation $\mathcal{V}$ for which there are two players (as in section 2.4) players I and II. Player I attempts to show that E *fails* to have the property Φ whereas player II tries to prove otherwise. Unlike the equivalence games earlier players here do not necessarily take turns[19].

A play of the *property checking game* (E_0, Φ_0) relative to $\mathcal{V}$ is a finite or infinite sequence of the form, $(E_0, \Phi_0) \ldots (E_n, \Phi_n) \ldots$. The next move in a play, the step from (E_j, Φ_j) to (E_{j+1}, Φ_{j+1}) and which player makes it is determined by the main connective of Φ_j. An essential ingredient is the use of auxiliary propositional constants, ranged over by U, which are introduced as fixed point formulas are met. Suppose an initial part of a play is $(E_0, \Phi_0) \ldots (E_j, \Phi_j)$. The next step (E_{j+1}, Φ_{j+1}) is as below:

- if $\Phi_j = \Psi_1 \wedge \Psi_2$ $(\Psi_1 \vee \Psi_2)$ then player I (player II) chooses one of the conjuncts (disjuncts) Ψ_i: the process E_{j+1} is E_j and Φ_{j+1} is Ψ_i.
- if $\Phi_j = [K]\Psi$ $(\langle K \rangle \Psi)$ then player I (player II) chooses a transition $E_j \xrightarrow{a} E_{j+1}$ with $a \in K$ and Φ_{j+1} is Ψ.
- if $\Phi_j = \nu Z.\Psi$ $(\mu Z.\Psi)$ then player I (player II) chooses a *new* constant U and sets $U \stackrel{\text{def}}{=} \nu Z.\Psi$: the process E_{j+1} is E_j and Φ_{j+1} is U.
- if $\Phi_j = U$ and $U \stackrel{\text{def}}{=} \nu Z.\Psi$ $(U \stackrel{\text{def}}{=} \mu Z.\Psi)$ then player I (player II) unfolds the fixed point so Φ_{j+1} is $\Psi\{U/Z\}$ and E_{j+1} is E_j.

These rules are *backwards sound* with respect to the intentions of the players. If player I makes the move (E_{j+1}, Φ_{j+1}) from (E_j, Φ_j) and E_{j+1} *fails* to

[19] It is straightforward to reformulate their definition so that players must take turns.

satisfy Φ_{j+1} relative to $\mathcal{V}$ then also E_j fails to have Φ_j. In contrast when player II makes this move and $E_{j+1} \models_{\mathcal{V}} \Phi_{j+1}$ then also $E_j \models_{\mathcal{V}} \Phi_j$. In the case of a fixed point formula this is clear provided we understand the presence of a constant to be its defined equivalent. Formulas are no longer "pure" as they may contain constants. However we can recover a pure formula from an impure formula by replacing constants with their defined fixed points in reverse order of introduction: assuming that $U_1 \stackrel{\mathrm{def}}{=} \Psi_1 \ldots U_n \stackrel{\mathrm{def}}{=} \Psi_n$ is the sequence of declarations in order of introduction, the meaning of Ψ is just $\Psi\{\Psi_n/U_n\}\ldots\{\Psi_1/U_1\}$. Consequently the fixed point unfolding principle justifies the backwards soundness of the moves determined by the constants.

A player *wins* a play of a game in the circumstances described below:

Player II wins	**Player I wins**
1. Configuration (E,Φ) is reached and $\Phi = \mathtt{tt}$ or $\Phi = Z$ and $F \in \mathcal{V}(Z)$	$1'$. Configuration (E,Φ) is reached and $\Phi = \mathtt{ff}$ or $\Phi = Z$ and $F \notin \mathcal{V}(Z)$
2. Configuration $(E,[K]\Phi)$ is reached and $\{F : E \xrightarrow{a} F \text{ and } a \in K\} = \emptyset$	$2'$. Configuration $(E,\langle K\rangle\Phi)$ is reached and $\{F : E \xrightarrow{a} F \text{ and } a \in K\} = \emptyset$
3. Configuration (E,U) is repeated when $U \stackrel{\mathrm{def}}{=} \nu Z.\Phi$	3. Configuration (E,U) is repeated when $U \stackrel{\mathrm{def}}{=} \mu Z.\Phi$
4. U occurs infinitely often when $U \stackrel{\mathrm{def}}{=} \nu Z.\Phi$	$4'$. U occurs infinitely often when $U \stackrel{\mathrm{def}}{=} \mu Z.\Phi$

If the configuration reached is (E,U) when U abbreviates a maximal (minimal) fixed point formula, and this same configuration occurs earlier in the game play, then player II (I) wins. More generally as a play can have infinite length (but only when the initial process is infinite state) this repeat condition for winning is generalized for these plays. Player I (II) wins if there is a least (greatest) fixed point constant U which is traversed infinitely often. In any infinite length play there is exactly one constant that occurs infinitely often, and therefore just one of the players wins the play.

Lemma 1 *If* $(E_0,\Phi_0)\ldots(E_n,\Phi_n)\ldots$ *is an infinite length game play then there is exactly one constant* U *that for infinitely many* j, $\Phi_j = U$.

A player is said to have a *winning strategy* for a game if she is able to win any play of it. This means that she can always respond effectively to the moves her opponent makes. It turns out that player II has a winning strategy for (E,Φ) just in case E has the property Φ.

Theorem 1 $E \models_{\mathcal{V}} \Phi$ *iff player II has a winning strategy for* (E,Φ) relative to $\mathcal{V}$.

This result provides an alternative characterization of the satisfaction relation between processes and formulas. Game playing does not depend upon explicit calculation of fixed points. Also there is not an explicit appeal to induction over approximant indices. Indeed it is even open-ended as to knowing only part or all of the transition graph of a process. There is a further feature that we will comment on in section 4.6, the possibility of more sophisticated game playing where moves may also be guided by the algebraic structure of a process expression.

An infinite length game play must traverse a particular constant infinitely often, and so it follows that a play of (E, Φ) when E is finite-state must have finite length. Moreover for any such game there is an exponential upper bound on the number of different plays up to renaming of constants. Property checking of finite-state processes via game playing is therefore decidable. However this is not a very (time) efficient method[20]. In section 4.3 we provide less costly techniques based on games. For the remainder of this section we illustrate game playing.

Example 1 Player II has a winning strategy for the game $(Cl, \nu Z. \langle \texttt{tick} \rangle Z)$. The only possible play (up to renaming the constant) is $(Cl, \nu Z. \langle \texttt{tick} \rangle Z)(Cl, U)$ $(Cl, \langle \texttt{tick} \rangle U)(Cl, U)$. Player II also has a winning strategy for $(Cl_3, \nu Z. \langle \texttt{tick} \rangle Z)$ with respect to the slightly different clock $Cl_3 \stackrel{\text{def}}{=} \texttt{tick}.Cl_3 + \texttt{tick}.\mathbf{0}$. A winning play is almost identical to the previous game play, $(Cl_3, \nu Z. \langle \texttt{tick} \rangle Z)(Cl_3, U)$ $(Cl_3, \langle \texttt{tick} \rangle U)(Cl_3, U)$. If player II chooses the other tick transition, $Cl_3 \xrightarrow{\texttt{tick}} \mathbf{0}$ instead then she loses. This illustrates that having a winning strategy means that the player is always able to win, and not that every available choice leads to a win. □

Example 2 Cnt is a simple infinite state system, $Cnt \stackrel{\text{def}}{=} \texttt{up}.(Cnt \mid \texttt{down}.\mathbf{0})$. It has the property that it may do up forever, as the single play of $(Cnt, \nu Z. \langle \texttt{up} \rangle Z)$, which is infinite length, is won by player II: $(Cnt, \nu Z. \langle \texttt{up} \rangle Z)(Cnt, U)(Cnt, \langle \texttt{up} \rangle U)$ $(Cnt \mid \texttt{down}.\mathbf{0}, U) \ldots (Cnt \mid \ldots \mid \texttt{down}.\mathbf{0}, U) \ldots$. Player II wins because the constant U recurs infinitely often. For a similar reason player I wins the only play of $(Cnt, \mu Z. [\texttt{up}] Z)$. □

Example 3 Let $D \xrightarrow{a} D'$ and $D' \xrightarrow{a} D$ and $D' \xrightarrow{b} \mathbf{0}$. Assume that Ψ is the formula $\mu Y. \nu Z. [a]((\langle b \rangle \texttt{tt} \vee Y) \wedge Z)$ and that Ψ_1 is $\nu Z. [a]((\langle b \rangle \texttt{tt} \vee U) \wedge Z)$. D' fails to have the property Ψ, and so player I is able to win any play of (D', Ψ): $(D', \Psi)(D', U)(D', \Psi_1)(D', V)(D', [a]((\langle b \rangle \texttt{tt} \vee U) \wedge V))(D, (\langle b \rangle \texttt{tt} \vee U) \wedge V)$ $(D, \langle b \rangle \texttt{tt} \vee U)$ At $(D, \langle b \rangle \texttt{tt} \vee U)$ player II has a choice. If she chooses the first disjunct she then loses because there is not a b transition from D. So the play proceeds: $(D, \langle b \rangle \texttt{tt} \vee U)(D, U)(D, \Psi_1)(D, W)(D, [a]((\langle b \rangle \texttt{tt} \vee U) \wedge W))$ $(D', (\langle b \rangle \texttt{tt} \vee U) \wedge W)(D', W)(D', [a]((\langle b \rangle \texttt{tt} \vee U) \wedge W))(D, (\langle b \rangle \texttt{tt} \vee U) \wedge W)(D, \langle b \rangle \texttt{tt} \vee U)$. Again player II has a choice at $(D, \langle b \rangle \texttt{tt} \vee U)$, but both options lose: as D has no b transition she can not choose the first disjunct, and the second disjunct gives a repeat configuration. Note here the essential requirement in game playing that a new constant is introduced when a fixed point formula is met: if the constant W had not been introduced, and V reused in the fourth configuration above then a repeat configuration (D', V) would have to occur.

[20] Let $|\Phi|$ be the size of Φ, the number of connectives and atomic formulas in Φ, and let $f(\Phi)$ be the number of occurrences of fixed point operators in Φ or 1 if there are not any, and let $|E|$ be the number of processes in $\mathcal{P}(E)$. A play of a game has maximum length $1 + (|E| * |\Phi|)^{f(\Phi)}$.

Later we shall refine the notion of game so that constants are reintroduced, but then the condition for who wins a repeat configuration is more complex. $\Box$

4.2 Tableaux

When processes are game equivalent a bisimulation relation underpins player II's winning strategy. We now develop a *tableau* proof system for property checking so that a successful proof tree underpins a winning strategy for player II. Here we restrict ourselves to finitely branching processes. The proof system here was first presented in [78]. It is guaranteed to show properties of finite state processes, and is a precursor of a later property checker which can also establish temporal features of infinite state processes.

A proof is built from sequents of the form $E \vdash_{\mathcal{V}} \Phi$ which are analogues of the semantic notion $E \models_{\mathcal{V}} \Phi$ and of the game configuration (E, Φ) relative to $\mathcal{V}$. Usually the index $\mathcal{V}$ is dropped from $\vdash_{\mathcal{V}}$. The proof system is goal directed, similar in style to the rules we presented for transitions. Each proof rule has the form

$$\frac{E \vdash \Phi}{E_1 \vdash \Phi_1 \ldots E_n \vdash \Phi_n}$$

with $n \geq 1$ and possibly with side conditions. Again the premise sequent $E \vdash \Phi$ is the *goal* to be achieved (that E has the temporal property Φ) while the consequents are the subgoals. The rules are presented in figure 14. Again we appeal

$$\wedge \ \frac{E \vdash \Phi \wedge \Psi}{E \vdash \Phi \qquad E \vdash \Psi}$$

$$\vee \ \frac{E \vdash \Phi \vee \Psi}{E \vdash \Phi} \qquad\qquad \vee \ \frac{E \vdash \Phi \vee \Psi}{E \vdash \Psi}$$

$$[K] \ \frac{E \vdash [K]\Phi}{E_1 \vdash \Phi \ \ldots \ E_n \vdash \Phi} \ \{F \mid E \xrightarrow{a} F \text{ and } a \in K\} = \{E_1, \ldots, E_n\}$$

$$\langle K \rangle \ \frac{E \vdash \langle K \rangle \Phi}{F \vdash \Phi} \ E \xrightarrow{a} F \text{ and } a \in K$$

$$\sigma Z. \ \frac{E \vdash \sigma Z.\, \Phi}{E \vdash U} \ U \stackrel{\text{def}}{=} \sigma Z.\, \Phi \text{ and } U \text{ is new}$$

$$U \ \frac{E \vdash U}{E \vdash \Phi\{U/Z\}} \ U \stackrel{\text{def}}{=} \sigma Z.\, \Phi$$

Fig. 14. Tableau proof rules

to auxiliary propositional constants, ranged over by U, which are introduced as fixed point formulas are met.

The rules are backwards sound in the sense that if the consequents of any rule are true then so is the premise goal. This is clear for the rules for the boolean and modal operators. In the case of a fixed point formula this is apparent provided we understand the presence of a constant to be its defined equivalent as in the previous section.

To test whether E has the property Φ relative to $\mathcal{V}$, one tries to achieve the goal $E \vdash_{\mathcal{V}} \Phi$ by building a *successful tableau*, a finite proof tree whose root is labelled with this initial sequent, and where all the leaves are labelled by sequents that are true. Sequents labelling the immediate successors of a node labelled $F \vdash \Psi$ are determined by an application of one of the rules, dependent on the form of Ψ. The rules are backwards sound in the sense that if the consequents of any rule are true then so is the premise goal, and therfore it follows that if the leaves of a finite proof tree are true then so is the root. So we need to present conditions for when a node in a proof tree counts as a leaf.

We assume that the rules above only apply to nodes of a proof tree that are not terminal. A node $\mathbf{n}$ labelled with the sequent $F \vdash \Psi$ is *terminal* if one of the following conditions hold which are very similar to when game playing may terminate:

Successful terminal	**Unsuccessful terminal**
1. $\Psi = \mathtt{tt}$ or $\Psi = Z$ and $F \in \mathcal{V}(Z)$	$1'$. $\Psi = \mathtt{ff}$ or $\Psi = Z$ and $F \notin \mathcal{V}(Z)$
2. $\Psi = [K]\Phi$ and the set $\{F : E \stackrel{a}{\longrightarrow} F \text{ and } a \in K\} = \emptyset$	$2'$. $\Psi = \langle K\rangle\Phi$ and the set $\{F : E \stackrel{a}{\longrightarrow} F \text{ and } a \in K\} = \emptyset$
3. $\Psi = U$ and $U \stackrel{\mathrm{def}}{=} \nu Z.\Phi$ and there is a node above $\mathbf{n}$ labelled $F \vdash \Psi$	$3'$. $\Psi = U$ and $U \stackrel{\mathrm{def}}{=} \mu Z.\Phi$ and there is a node above $\mathbf{n}$ labelled $F \vdash \Psi$

It is clear that nodes labelled with sequents obeying 1 or 2 are successful as then F has Ψ relative to $\mathcal{V}$, and similarly nodes labelled with sequents fulfilling $1'$ and $2'$ are not true. The remaining two conditions are analogues of when a finite length game play ends through repetition. They provide precise circumstances for when a proof tree containing formulas with fixed points should terminate, which occurs when a sequent of the form $F \vdash U$ is repeated. It is at this point that we distinguish in the proof theory between the two kinds of fixed points as they are not differentiated by the rules earlier. We also assume that any node labelled with the sequent $E \vdash \mathtt{tt}$ is a successful leaf, and that any node labelled $E \vdash \mathtt{ff}$ is unsuccessful.

Definition 1 A *successful* tableau for $E \vdash \Phi$ is a finite proof tree whose root is labelled with $E \vdash \Phi$, and all other nodes are labelled with sequents which are the result of an application of one of the rules to the sequent labelled at the node immediately above them, and all of whose leaves are successful (obey one of the conditions 1,2,3 above). □

A path in a finite proof tree for $E_0 \vdash \Phi_0$ is a finite sequence of labelled nodes $E_0 \vdash \Phi_0 \ldots E_n \vdash \Phi_n$ where each one lies directly beneath its predecessor, and where the final node (labelled $E_n \vdash \Phi_n$) is a leaf. Associated with such a path is the game play $(E_0, \Phi_0) \ldots (E_n, \Phi_n)$. In the case of a successful tableau, player

II is the winner of each such play: every path when understood as a game play fulfills the conditions for player II's success. Moreover *all* the possible choices player I can make are present in the proof tree (up to renaming of constants). Thus, a successful tableau for $E \vdash \Phi$ captures player II's winning strategy for the game (E, Φ).

Proposition 1 *If $E \vdash_{\mathcal{V}} \Phi$ has a successful tableau then $E \models_{\mathcal{V}} \Phi$.*

Example 1 A complete description of the clock Cl is that it perpetually ticks and can do nothing else. This singular capability is expressed by the following formula, $\nu Z.\,([-\mathtt{tick}]\mathtt{ff} \wedge \langle - \rangle \mathtt{tt}) \wedge [-]Z$. Below is a proof that Cl has this property (where we leave implicit the side conditions on the $[K]$, $\langle K \rangle$, and σZ rules):

$$\dfrac{Cl \vdash \nu Z.\,([-\mathtt{tick}]\mathtt{ff} \wedge \langle - \rangle \mathtt{tt}) \wedge [-]Z}{\dfrac{Cl \vdash U}{\dfrac{Cl \vdash ([-\mathtt{tick}]\mathtt{ff} \wedge \langle - \rangle \mathtt{tt}) \wedge [-]U}{\dfrac{Cl \vdash [-\mathtt{tick}]\mathtt{ff} \wedge \langle - \rangle \mathtt{tt}}{Cl \vdash [-\mathtt{tick}]\mathtt{ff} \quad \dfrac{Cl \vdash \langle - \rangle \mathtt{tt}}{Cl \vdash \mathtt{tt}}} \quad \dfrac{Cl \vdash [-]U}{Cl \vdash U}}}}$$

There are three kinds of successful leaf here. □

Proposition 1 asserts that if $E \vdash \Phi$ *has* a successful tableau then E has the property Φ. This leaves open the possibility that $E \vdash \Phi$ may have many associated tableaux, some of which may be successful and some of which may be unsuccessful. The rules that permit choice are the $\langle K \rangle$ and the $\vee$ rules.

Example 2 Let $Cl_3 \stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl_3 + \mathtt{tick}.\mathbf{0}$. Below are two tableaux for $Cl_3 \vdash \nu Z.\,\langle \mathtt{tick} \rangle Z$ illustrating the choice permitted by the $\langle K \rangle$ rule.

$$\dfrac{Cl_3 \vdash \nu Z.\,\langle \mathtt{tick} \rangle Z}{\dfrac{Cl_3 \vdash U}{\dfrac{Cl_3 \vdash \langle \mathtt{tick} \rangle U}{\dfrac{\mathbf{0} \vdash U}{\mathbf{0} \vdash \langle \mathtt{tick} \rangle U}} \; Cl_3 \stackrel{\mathtt{tick}}{\longrightarrow} \mathbf{0}}} \; U \stackrel{\mathrm{def}}{=} \nu Z.\,\langle \mathtt{tick} \rangle Z$$

$$\dfrac{Cl_3 \vdash \nu Z.\,\langle \mathtt{tick} \rangle Z}{\dfrac{Cl_3 \vdash U}{\dfrac{Cl_3 \vdash \langle \mathtt{tick} \rangle U}{Cl_3 \vdash U} \; Cl_3 \stackrel{\mathtt{tick}}{\longrightarrow} Cl_3}} \; U \stackrel{\mathrm{def}}{=} \nu Z.\,\langle \mathtt{tick} \rangle Z$$

The first of these tableaux is unsuccessful and corresponds to the game play of example 1 of section 4.1 when player II chooses the bad transition $Cl_3 \xrightarrow{\texttt{tick}} \mathbf{0}$. The second is successful. $\square$

The tableau proof method provides a means for laying out a proof of a property. It does not itself offer insight into how to find a successful tableau, for the "correct" choices under the $\langle K \rangle$ and $\vee$ rules may depend on complex behavioural factors of the particular process under consideration, and therefore on matters outwith the proof system itself.

Example 3 The vending machine *Ven*, from section 1.1, has the property that whenever 2p is deposited eventually a big item is collected, which is expressed by the following formula $\Psi = \nu Z. [\texttt{2p}](\mu Y. \langle - \rangle \texttt{tt} \wedge [-\texttt{collect}_b]Y) \wedge [-]Z$. A successful tableau is presented in two stages in figure 15 where Φ abbreviates the subformula $\mu Y. \langle - \rangle \texttt{tt} \wedge [-\texttt{collect}_b]Y$, and $\texttt{c}_b.Ven$ and $\texttt{c}_l.Ven$ abbreviate $\texttt{collect}_b.Ven$ and $\texttt{collect}_l.Ven$. Notice how similar the subtableaux $T1$ and $T2$ are. Later we examine how to amalgamate their proof trees. $\square$

Example 4 Let $D \xrightarrow{a} D'$ and $D' \xrightarrow{a} D$ and $D' \xrightarrow{b} \mathbf{0}$. Assume that Φ is the following formula $\nu Z. \mu Y. [a]((\langle b \rangle \texttt{tt} \wedge Z) \vee Y)$. The successful tableau for $D \vdash \Phi$ is presented in figure 16, where we omit the side conditions on the applicationof rules. Notice the important requirement that a new constant is introduced when a fixed point is met. Both V and W abbreviate the same fixed point formula. $\square$

The tableau proof system presented here is *complete* for finite state processes. This follows from the game characterization of satisfaction.

Proposition 2 *If E is finite-state and $E \models_{\mathcal{V}} \Phi$ then $E \vdash_{\mathcal{V}} \Phi$ has a successful tableau.*

4.3 Refinement of games and tableaux

In this section we refine the definition of game play to provide a more efficient characterization of the satisfaction relation by reintroducing constants. We then consider how this refinement affects the construction of tableaux. We change just the rule for introducing constants for fixed points, and divide it into two cases. Recall that we are defining the next pair in the play $(E_0, \Phi_0) \ldots (E_j, \Phi_j)$:

- if $\Phi_j = \nu Z. \Psi$ ($\mu Z. \Psi$) and player I (player II) has not previously introduced a constant $V \stackrel{\text{def}}{=} \nu Z. \Psi$ ($V \stackrel{\text{def}}{=} \mu Z. \Psi$) then player I (player II) chooses a *new* constant U and sets $U \stackrel{\text{def}}{=} \nu Z. \Psi$ ($U \stackrel{\text{def}}{=} \mu Z. \Psi$): the process E_{j+1} is E_j and Φ_{j+1} is U.
- if $\Phi_j = \nu Z. \Psi$ ($\mu Z. \Psi$) and player I (player II) has previously introduced a constant $V \stackrel{\text{def}}{=} \nu Z. \Psi$ ($V \stackrel{\text{def}}{=} \mu Z. \Psi$) then E_{j+1} is E_j and Φ_{j+1} is V.

All the other moves and who is responsible for them are as in section 4.1, and player II wins a play in the circumstances 1 and 2, and player I in the case of

$$
\dfrac{Ven \vdash \Psi}{\dfrac{Ven \vdash U}{\dfrac{Ven \vdash [2\mathrm{p}]\Phi \wedge [-]U}{\dfrac{Ven \vdash [2\mathrm{p}]\Phi}{\dfrac{Ven_b \vdash \Phi}{\dfrac{Ven_b \vdash V}{\dfrac{Ven_b \vdash \langle - \rangle \mathtt{tt} \wedge [-\mathtt{collect}_b]V}{\dfrac{Ven_b \langle - \rangle \mathtt{tt}}{\mathtt{c}_b.\,Ven \vdash \mathtt{tt}} \quad \dfrac{Ven_b \vdash [-\mathtt{collect}_b]V}{\dfrac{\mathtt{c}_b.\,Ven \vdash V}{\dfrac{\mathtt{c}_b.\,Ven \vdash \langle - \rangle \mathtt{tt} \wedge [-\mathtt{collect}_b]V}{\dfrac{\mathtt{c}_b.\,Ven \vdash \langle - \rangle \mathtt{tt}}{Ven \vdash \mathtt{tt}} \quad \mathtt{c}_b.\,Ven \vdash [-\mathtt{collect}_b]V}}}}} \; V \stackrel{\mathrm{def}}{=} \Phi}} \quad \dfrac{Ven \vdash [-]U}{T1 \quad T2}} \; U \stackrel{\mathrm{def}}{=} \Psi}}
$$

$$
\dfrac{T1}{\dfrac{Ven_b \vdash U}{\dfrac{Ven_b \vdash [2\mathrm{p}]\Phi \wedge [-]U}{Ven_b \vdash [2\mathrm{p}]\Phi \quad \dfrac{Ven_b \vdash [-]U}{\dfrac{\mathtt{c}_b.\,Ven \vdash U}{\dfrac{\mathtt{c}_b.\,Ven \vdash [2\mathrm{p}]\Phi \wedge [-]U}{\mathtt{c}_b.\,Ven \vdash [2\mathrm{p}]\Phi \quad \dfrac{\mathtt{c}_b.\,Ven \vdash [-]U}{Ven \vdash U}}}}}}}
$$

$$
\dfrac{T2}{\dfrac{Ven_l \vdash U}{\dfrac{Ven_l \vdash [2\mathrm{p}]\Phi \wedge [-]U}{Ven_l \vdash [2\mathrm{p}]\Phi \quad \dfrac{Ven_l \vdash [-]U}{\dfrac{\mathtt{c}_l.\,Ven \vdash U}{\dfrac{\mathtt{c}_l.\,Ven \vdash [2\mathrm{p}]\Phi \wedge [-]U}{\mathtt{c}_l.\,Ven \vdash [2\mathrm{p}]\Phi \quad \dfrac{\mathtt{c}_l.\,Ven \vdash [-]U}{Ven \vdash U}}}}}}}
$$

Fig. 15.

$$
\dfrac{D \vdash \Phi}{\dfrac{D \vdash U}{\dfrac{D \vdash \mu Y.\,[a]((\langle b\rangle \mathtt{tt} \wedge U) \vee Y)}{\dfrac{D \vdash V}{\dfrac{D \vdash [a]((\langle b\rangle \mathtt{tt} \wedge U) \vee V)}{\dfrac{D' \vdash (\langle b\rangle \mathtt{tt} \wedge U) \vee V}{\dfrac{D' \vdash \langle b\rangle \mathtt{tt} \wedge U}{\dfrac{D' \vdash \langle b\rangle \mathtt{tt}}{\mathbf{0} \vdash \mathtt{tt}} \quad \dfrac{D' \vdash U}{\dfrac{D' \vdash \mu Y.\,[a]((\langle b\rangle \mathtt{tt} \wedge U) \vee Y)}{\dfrac{D' \vdash W}{\dfrac{D' \vdash [a]((\langle b\rangle \mathtt{tt} \wedge U) \vee W)}{\dfrac{D \vdash (\langle b\rangle \mathtt{tt} \wedge U) \vee W}{\dfrac{D \vdash W}{\dfrac{D \vdash [a]((\langle b\rangle \mathtt{tt} \wedge U) \vee W)}{\dfrac{D' \vdash (\langle b\rangle \mathtt{tt} \wedge U) \vee W}{\dfrac{D' \vdash \langle b\rangle \mathtt{tt} \wedge U}{\dfrac{D' \vdash \langle b\rangle \mathtt{tt}}{\mathbf{0} \vdash \mathtt{tt}} \quad D' \vdash U}}}}}}}}}}}}}}
$$

Fig. 16. A successful tableau for $D \vdash \Phi$

$1'$ and $2'$: for instance, if the configuration reached in a play is $(E, [K]\Phi)$ and E has no K transition then player II wins. The other conditions for winning from section 4.1, 3 and $3'$ when there is a repeat configuration, and 4 and $4'$ for infinite length plays, need to be redefined because constants are reintroduced: an infinite length play may now contain more than one constant that recurs infinitely often.

Definition 1 The constant U is *active* in Φ iff either U occurs in Φ, or some constant V occurs in Φ and $V \stackrel{\text{def}}{=} \sigma Z.\Psi$ and U is active in $\sigma Z.\Psi$.

The constraints on how constants are introduced ensure that being active is well defined. Activity of a constant can be extended to finite or infinite length sequences of formulas, U is *active* throughout $\Phi_0, \ldots, \Phi_n \ldots$ if it is active in each Φ_i.

Lemma 1 i. *If* $(E_0,\Phi_0),\ldots,(E_n,\Phi_n)$ *is an initial part of a game play and* $\Phi_i = \Phi_n$ *for some* $i < n$*, then there is a unique constant* U *which is active throughout* $\Phi_i,\ldots,\Phi_n$ *and which occurs there,* $\Phi_j = U$ *for some* $j : i \leq j \leq n$.

ii. *If* $(E_0,\Phi_0),\ldots,(E_n,\Phi_n)\ldots$ *is an infinite length game play then there is a unique constant* U *which occurs infinitely often and is active throughout* $\Phi_j,\ldots,\Phi_n\ldots$ *for some* $j \geq 0$.

Lemma 1 governs the remaining winning conditions for game playing. A repeat configuration (E,Ψ) when Ψ is *any formula*, and not just a constant, terminates play. Who wins depends on the sequence of formulas between (and including) the identical configurations. There is exactly one constant U which is active in this cycle and which occurs within it: if it abbreviates a maximal fixed point formula then player II wins and otherwise it must abbreviate a least fixed point formula and player I wins. In any infinite length play there is a unique constant which is traversed infinitely often and which is active for all but a finite prefix: if this constant abbreviates a maximal fixed point formula player II wins and otherwise player I wins.

As before a player is said to have a winning strategy for a game if she is able to win any play of it. She is then always able to respond effectively to any move her opponent makes. The following result provides an alternative characterization of the satisfaction relation between processes and formulas.

Theorem 1 $E \models_{\mathcal{V}} \Phi$ *iff player II has a winning strategy for* (E,Φ) *relative to* $\mathcal{V}$.

Assume that E is a finite state process. There are at most $(|E| * |\Phi|)$ different configurations in any game play (up to renaming of constants). This means that any play of (E,Φ) has at most length $1 + (|E| * |\Phi|)$.

Example 1 A case where game playing is shorter is example 3 of section 4.1. Let D, D', Ψ and Ψ_1 be as in that example. We show that player I has a winning strategy for (D',Ψ) where $U \stackrel{\text{def}}{=} \Psi$ and $V \stackrel{\text{def}}{=} \Psi_1$: $(D',\Psi)(D',U)(D',\Psi_1)$ $(D',V)(D',[a]((\langle b\rangle\mathtt{tt} \vee U) \wedge V))(D,(\langle b\rangle\mathtt{tt} \vee U) \wedge V)(D,\langle b\rangle\mathtt{tt} \vee U)$. At this configuration player II has a choice. If she chooses the first disjunct she then loses because there is not a b transition from D. Otherwise the play proceeds: $(D,\langle b\rangle\mathtt{tt} \vee U)(D,U)(D,\Psi_1)(D,V)(D,[a]((\langle b\rangle\mathtt{tt} \vee U) \wedge V))$ $(D',(\langle b\rangle\mathtt{tt} \vee U) \wedge V)(D',V)$. There is now a repeat configuration (D',V). Consider the sequence of formulas, the cycle in the fraph for Ψ, between and including these identical configurations:

$$V, [a]((\langle b\rangle\mathtt{tt} \vee U) \wedge V), (\langle b\rangle\mathtt{tt} \vee U) \wedge V, \langle b\rangle\mathtt{tt} \vee U, U,$$
$$\Psi_1, V, [a]((\langle b\rangle\mathtt{tt} \vee U) \wedge V), (\langle b\rangle\mathtt{tt} \vee U) \wedge V, V$$

The constant V is not active throughout this sequence (since it is not active in the formula Ψ_1). However U is active throughout and also occurs here, and because it abbreviates a least fixed point formula player I is the winner. □

A tableau proof system was introduced in section 4.2 in such a way that a successful tableau underpins a winning strategy for player II (when the initial process is finite state). Given the refinements above to game playing we now redefine this tableau proof system. The only change to the rule is that when a fixed point formula is met again the same constant is reintroduced, new constants are only introduced for fixed point formulas that have not been met before.

To test whether E has the property Ψ one tries to build a successful tableau for $E \vdash_{\mathcal{V}} \Psi$ which is a finite proof tree whose root is labelled with this sequent. The leaves have to be successful as defined in section 4.2, except for the case of termination because of repetition. The new conditions for terminating through repetition are:

$$
\begin{array}{lllll}
\vdots & U \stackrel{\mathrm{def}}{=} \nu Z.\Phi & \qquad & \vdots & U \stackrel{\mathrm{def}}{=} \mu Z.\Phi \\
\vdots & & & \vdots & \\
E \vdash \Phi & \uparrow & & E \vdash \Phi & \uparrow \\
\vdots & U \text{ active} & & \vdots & U \text{ active} \\
F \vdash U & & & F \vdash U & \\
\vdots & \text{throughout} & & \vdots & \text{throughout} \\
E \vdash \Phi & \downarrow & & E \vdash \Phi & \downarrow \\
 & & & & \\
\multicolumn{2}{l}{\textbf{Successful}} & & \multicolumn{2}{l}{\textbf{Unsuccessful}}
\end{array}
$$

Again it is at this point that we distinguish in the proof theory between the two kinds of fixed points as they are not differentiated by the rules earlier. Again a successful tableau for $E \vdash \Phi$ captures player II's winning strategy for the game (E, Φ).

Proposition 1 *If $E \vdash_{\mathcal{V}} \Phi$ has a successful tableau then $E \models_{\mathcal{V}} \Phi$.*

Example 3 We redo the proof of example 4 of section 4.2 in figure 17. As U, a maximal fixed point constant, is active throughout the cycle from $D \vdash V$ to $D \vdash V$ and occurs within it the tableau is successful. $\square$

The tableau proof system presented here is again complete for finite-state processes. This is a consequence of the game characterization of satisfaction. Instead of building tableaux for finite state processes, an alternative approach is to construct the game graph for (E, Φ) whose vertices are pairs (F, Ψ) when this is a possible configuration of a game play from (E, Φ) modulo a canonical means of choosing constants. There is a directed edge between two vertices, $v_1 \longrightarrow v_2$ if a player can make as her next move v_2 from v_1: for instance, $(E, \Phi \vee \Psi) \longrightarrow (E, \Psi)$. the first to the second node. The size of a game graph for (E, Φ) is at most $|E| * |\Phi|$. In fact a game graph is an (amorphous) alternating automaton in the sense of [66, 13]. The and-vertices are the configurations from which player I must proceed and the or-vertices are those from which player II moves. The acceptance condition is given in terms of constants: for each least fixed point constant U if U occurs infinitely often then V occurs infinitely often for some maximal fixed point constant active in U. The root of the graph is the initial game configuration, and non-emptiness of this graph is equivalent

$$
\begin{array}{c}
D \vdash \Phi \\ \hline
D \vdash U \\ \hline
D \vdash \mu Y.\,[a]((\langle b\rangle \mathtt{tt} \wedge U) \vee Y) \\ \hline
D \vdash V \\ \hline
D \vdash [a]((\langle b\rangle \mathtt{tt} \wedge U) \vee V) \\ \hline
D' \vdash (\langle b\rangle \mathtt{tt} \wedge U) \vee V \\ \hline
D' \vdash \langle b\rangle \mathtt{tt} \wedge U \\ \hline
\begin{array}{cc}
\begin{array}[t]{c} D' \vdash \langle b\rangle \mathtt{tt} \\ \hline \mathbf{0} \vdash \mathtt{tt} \end{array} &
\begin{array}[t]{c}
D' \vdash U \\ \hline
D' \vdash \mu Y.\,[a]((\langle b\rangle \mathtt{tt} \wedge U) \vee Y) \\ \hline
D' \vdash V \\ \hline
D' \vdash [a]((\langle b\rangle \mathtt{tt} \wedge U) \vee V) \\ \hline
D \vdash (\langle b\rangle \mathtt{tt} \wedge U) \vee V \\ \hline
D \vdash V
\end{array}
\end{array}
\end{array}
$$

Fig. 17.

to player II having a winning strategy (and there being a successful tableau). Techniques similar to those in [13, 40] for checking non-emptiness of automata, provide efficient algorithms for automatically checking properties of finite state systems.

4.4 Generalized tableaux I

We have presented characterizations of when a process has a temporal property using games. Player II is able to win any play of (E, Φ) just in case E has the property Φ. Using games we also developed tableau proof systems for verifying that processes have properties. However a successful proof is only guaranteed in the case of a property holding of a finite state process. We would like a more general proof technique that allows us to show properties of processes that are infinite state. Moreover, even in the finite state case, a tableau proof may be too unwieldy because of its size.

There is another motivation for examining more general proof systems. Often we are really interested in properties of schematic processes such as the scheduler of section 1.4 which is parametric on the size of its cycle. We would often like to prove a property for all such systems of sufficient size. The extra variable here means that the system is in effect infinite state. Similar comments can also be made in the case of processes that are parametric on their data. To deal with infinite state systems and schematic process descriptions we generalize the tableau proof system of section 4.2, and build proofs from more general sequents which relate sets of processes to properties. This we do over two sections.

We start by generalizing the satisfaction relation with the following stipulation:

$$\boxed{\mathcal{E} \models_{\mathcal{V}} \Phi \ \text{ iff } \ \text{for all } E \in \mathcal{E}.\ E \models_{\mathcal{V}} \Phi}$$

As usual when Φ is a closed formula we drop the valuation index from the satisfaction relation.

Example 1 A straightforward instance is that the family of counters of figure 4, $\{Ct_i \,:\, i \geq 0\}$ has the property $[\mathtt{up}]([\mathtt{round}]\mathtt{ff} \wedge [\mathtt{up}]\langle\mathtt{down}\rangle\langle\mathtt{down}\rangle\mathtt{tt})$. The following proof is more direct than appealing to induction over process indices.

$$\begin{array}{ll}
 & \{Ct_i \,:\, i \geq 0\} \models [\mathtt{up}]([\mathtt{round}]\mathtt{ff} \wedge [\mathtt{up}]\langle\mathtt{down}\rangle\langle\mathtt{down}\rangle\mathtt{tt}) \\
\text{iff} & \{Ct_i \,:\, i \geq 1\} \models [\mathtt{round}]\mathtt{ff} \wedge [\mathtt{up}]\langle\mathtt{down}\rangle\langle\mathtt{down}\rangle\mathtt{tt} \\
\text{iff} & \{Ct_i \,:\, i \geq 1\} \models [\mathtt{round}]\mathtt{ff} \text{ and } \{Ct_i \,:\, i \geq 1\} \models [\mathtt{up}]\langle\mathtt{down}\rangle\langle\mathtt{down}\rangle\mathtt{tt} \\
\text{iff} & \{Ct_i \,:\, i \geq 1\} \models [\mathtt{up}]\langle\mathtt{down}\rangle\langle\mathtt{down}\rangle\mathtt{tt} \\
\text{iff} & \{Ct_i \,:\, i \geq 2\} \models \langle\mathtt{down}\rangle\langle\mathtt{down}\rangle\mathtt{tt} \\
\text{iff} & \{Ct_i \,:\, i \geq 1\} \models \langle\mathtt{down}\rangle\mathtt{tt} \\
\text{iff} & \{Ct_i \,:\, i \geq 0\} \models \mathtt{tt}
\end{array}$$

Here we have used the expected inductive definition of $\mathcal{E} \models \Phi$. □

We now generalize the tableau proof system of section 4.2 by extending proof sequents so that a set of processes may appear on the left hand side, $\mathcal{E} \vdash_{\mathcal{V}} \Phi$: as usual we drop the index $\mathcal{V}$ wherever possible. Each proof rule has one of two forms:

$$\frac{\mathcal{E} \vdash \Phi}{\mathcal{F} \vdash \Phi_1} \qquad \frac{\mathcal{E} \vdash \Phi}{\mathcal{F}_1 \vdash \Phi_1 \quad \mathcal{F}_2 \vdash \Phi_2}$$

possibly with side conditions. As in section 4.2 the premise sequent $\mathcal{E} \vdash \Phi$ is the goal to be achieved (that every process in $\mathcal{E}$ has the property Φ) while the consequents are the subgoals. A little notation is employed in the tableau proof rules, presented in figure 18. When $\mathcal{E}$ is a set of processes we let $K(\mathcal{E})$ be the set of processes which are the result of transitions whose labels are drawn from K, the set $\{F \,:\, \exists E \in \mathcal{E}.\, \exists a \in K.\, E \xrightarrow{a} F\}$. In the rule for the diamond modality we appeal to functions. A function f from $\mathcal{E}$ to $K(\mathcal{E})$ is a *choice* function provided that for each $E \in \mathcal{E}$ there is an $a \in K$ with the feature that $E \xrightarrow{a} f(E)$. When f is such a function we let $f(\mathcal{E}) = \{f(E) \,:\, E \in \mathcal{E}\}$. Here we generalize the proof system of section 4.2 and therefore we introduce new constants as fixed point formulas are met: this makes termination less complex than if we generalized the proof system of section 4.3 where constants are reintroduced. There is one new kind of rule, a structural rule Thin, which allows the set of processes in a goal sequent to be expanded. Clearly the rules are backwards sound.

To test if all the processes in $\mathcal{E}$ have the property Φ, one tries to achieve the goal $\mathcal{E} \vdash_{\mathcal{V}} \Phi$ by building a successful tableau, as before. A successful tableau is again a finite proof tree whose root is labelled with this initial sequent, and where

$$\wedge\ \frac{\mathcal{E} \vdash \Phi \wedge \Psi}{\mathcal{E} \vdash \Phi \quad \mathcal{E} \vdash \Psi} \qquad \vee\ \frac{\mathcal{E} \vdash \Phi \vee \Psi}{\mathcal{E}_1 \vdash \Phi \quad \mathcal{E}_2 \vdash \Psi}\ \mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$$

$$[K]\ \frac{\mathcal{E} \vdash [K]\Phi}{K(\mathcal{E}) \vdash \Phi} \qquad \langle K\rangle\ \frac{\mathcal{E} \vdash \langle K\rangle\Phi}{f(\mathcal{E}) \vdash \Phi}\ f : \mathcal{E} \to K(\mathcal{E}) \text{ is a choice function}$$

$$\sigma Z.\ \frac{\mathcal{E} \vdash \sigma Z.\Phi}{\mathcal{E} \vdash U}\ U \stackrel{\text{def}}{=} \sigma Z.\Phi \text{ and } U \text{ is new}$$

$$U\ \frac{\mathcal{E} \vdash U}{\mathcal{E} \vdash \Phi\{U/Z\}}\ U \stackrel{\text{def}}{=} \sigma Z.\Phi \qquad \text{Thin}\ \frac{\mathcal{E} \vdash \Phi}{\mathcal{F} \vdash \Phi}\ \mathcal{E} \subseteq \mathcal{F}$$

Fig. 18. Tableaux rules

all the leaves are labelled by sequents that are successful. Sequents labelling the immediate successors of a node labelled $\mathcal{F} \vdash \Psi$ are determined by an application of one of the rules, dependent on the structure of Ψ. Again we assume that the proof rules only apply to nodes of a proof tree that are not terminal. A node **n** labelled by the sequent $\mathcal{F} \vdash \Psi$ is terminal if one of the following conditions hold:

Successful terminal	**Unsuccessful terminal**
1. $\Psi = \mathtt{tt}$ or $\Psi = Z$ and $\mathcal{F} \subseteq \mathcal{V}(Z)$	$1'$. $\Psi = \mathtt{ff}$ or $\Psi = Z$ and $\mathcal{F} \not\subseteq \mathcal{V}(Z)$
2. $\mathcal{F} = \emptyset$	$2'$. $\Psi = \langle K\rangle\Phi$ and for some $F \in \mathcal{F}.\ K(\{F\}) = \emptyset$
3. $\Psi = U$ and $U \stackrel{\text{def}}{=} \nu Z.\Phi$ and there is a node above **n** labelled $\mathcal{E} \vdash U$ with $\mathcal{E} \supseteq \mathcal{F}$	$3'$. $\Psi = U$ and $U \stackrel{\text{def}}{=} \mu Z.\Phi$ and there is a node above **n** labelled $\mathcal{E} \vdash U$ with $\mathcal{F} \supseteq \mathcal{E}$

Conditions 3 and $3'$ generalize those for the proof system of section 4.2. A *successful* tableau for $\mathcal{E} \vdash \Phi$ relative to $\mathcal{V}$ is a *finite* proof tree all of whose leaves are successful. It turns out that a successful tableau only contains true leaves. However the system is not complete as it does not allow verification of some properties. We need one further termination condition for least fixed point constants. However this condition is a little complex and so we delay its discussion until the next section. Instead we present various examples that can be proved without it. In the following we often write $E \vdash \Phi$ instead of $\{E\} \vdash \Phi$.

Example 1 In example 3 of section 4.2 we noted how similar the two subtableaux $T1$ and $T2$ are. These can be amalgamated as in Figure 19 (where the same abbreviations are used). The terminals have changed slightly. □

Example 2 It is not possible to show that Cnt has the property $\nu Z.\langle \mathtt{up}\rangle Z$ using the proof rules of sections ??, when $Cnt \stackrel{\text{def}}{=} \mathtt{up}.(Cnt \,|\, \mathtt{down}.\mathbf{0})$. There is a

$$
\begin{array}{c}
\{Ven_b, Ven_l\} \vdash U \\ \hline
\{Ven_b, Ven_l\} \vdash [2\mathrm{p}]\Phi \wedge [-]U \\ \hline
\begin{array}{cc}
\begin{array}{c}
\{Ven_b, Ven_l\} \vdash [2\mathrm{p}]\Phi \\ \hline
\emptyset \vdash \Phi
\end{array}
&
\begin{array}{c}
\{Ven_b, Ven_l\} \vdash [-]U \\ \hline
\{\mathtt{c}_b.Ven, \mathtt{c}_l.Ven\} \vdash U \\ \hline
\{\mathtt{c}_b.Ven, \mathtt{c}_l.Ven\} \vdash [2\mathrm{p}]\Phi \wedge [-]U \\ \hline
\begin{array}{cc}
\begin{array}{c}
\{\mathtt{c}_b.Ven, \mathtt{c}_l.Ven\} \vdash [2\mathrm{p}]\Phi \\ \hline
\emptyset \vdash \Phi
\end{array}
&
\begin{array}{c}
\{\mathtt{c}_b.Ven, \mathtt{c}_l.Ven\} \vdash [-]U \\ \hline
Ven \vdash U
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
$$

Fig. 19. Amalgamated tableaux

very simple proof within this more general proof system. Let Cnt_0 be Cnt and let Cnt_{i+1} be $Cnt_i \,|\, \mathtt{down}.\mathbf{0}$ for any $i \geq 0$.

$$
\begin{array}{c}
Cnt \vdash \nu Z.\, \langle \mathtt{up} \rangle Z \\ \hline
\{Cnt_i \,:\, i \geq 0\} \vdash \nu Z.\, \langle \mathtt{up} \rangle Z \\ \hline
\{Cnt_i \,:\, i \geq 0\} \vdash U \\ \hline
\{Cnt_i \,:\, i \geq 0\} \vdash \langle \mathtt{up} \rangle U \\ \hline
\{Cnt_i \,:\, i \geq 1\} \vdash U
\end{array}
\quad U \overset{\mathrm{def}}{=} \nu Z.\, \langle \mathtt{up} \rangle Z
$$

Notice here the essential use of the Thin rule, and the simple condition for termination. The choice function which we have left implicit maps each Cnt_i to Cnt_{i+1}. $\square$

Example 3 Recall the level crossing of figure 7. Its safety property is expressed as $\nu Z.([\overline{\mathtt{tcross}}]\mathtt{ff} \vee [\overline{\mathtt{ccross}}]\mathtt{ff}) \wedge [-]Z$. Let Φ be this formula. We employ the abbreviations in figure 8, and let $\mathcal{E}$ be the full set $\{E_0, \ldots E_{11}\}$. Below is a successful tableau showing that the crossing has this property. The reader is invited to prove this property using the tableau proof system of section 4.2, and to compare its size with the proof here.

$$\dfrac{\mathit{Crossing} \vdash \Phi}{\dfrac{\dfrac{\mathcal{E} \vdash \Phi}{\mathcal{E} \vdash U}\ U \stackrel{\mathrm{def}}{=} \Phi}{\dfrac{\mathcal{E} \vdash ([\overline{\texttt{tcross}}]\texttt{ff} \lor [\overline{\texttt{ccross}}]\texttt{ff}) \land [-]U}{\dfrac{\mathcal{E} \vdash [\overline{\texttt{tcross}}]\texttt{ff} \lor [\overline{\texttt{ccross}}]\texttt{ff}}{\dfrac{\mathcal{E} - \{E_5, E_7\} \vdash [\overline{\texttt{tcross}}]\texttt{ff}}{\emptyset \vdash \texttt{ff}} \quad \dfrac{\mathcal{E} - \{E_4, E_6\} \vdash [\overline{\texttt{ccross}}]\texttt{ff}}{\emptyset \vdash \texttt{ff}}} \quad \dfrac{\mathcal{E} \vdash [-]U}{\mathcal{E} \vdash U}}}}$$

Again notice the essential use of the Thin rule at the first step. □

Example 4 The slot machine in figure 10 is infinite state. The safety property, that the machine never pays out more than it has in its bank as described in example 2 of section 3.7, has the following tableau where the falsity of Q_0 indicates that the slot machine owes money:

$$\dfrac{\{SM_n \,:\, n \geq 0\} \vdash \nu Z.Q_0 \land [-]Z}{\dfrac{\mathcal{E} \vdash \nu Z.Q_0 \land [-]Z}{\dfrac{\mathcal{E} \vdash U}{\dfrac{\mathcal{E} \vdash Q_0 \land [-]U}{\mathcal{E} \vdash Q_0 \quad \dfrac{\mathcal{E} \vdash [-]U}{\mathcal{E} \vdash U}}}}}$$

Here the Thin rule is used to enlarge the set of slot machines to the set $\mathcal{E}$ of derivatives of $\{SM_n \,:\, n \geq 0\}$. □

4.5 Generalized tableau II: well foundedness

The proof system of the previous section is not complete. A simple example for which there is not a proof is the cell $C \stackrel{\mathrm{def}}{=} \texttt{in}(x).B_x$ from section 3.5 when $x : \mathbb{N}$ and $B_{n+1} \stackrel{\mathrm{def}}{=} \texttt{down}.B_n$. This cell has the property of eventual termination, $\mu Z.\,[-]Z$. An attempt to show that C has this property using the tableau proof rules is:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{C \vdash \mu Z.[-]Z}{C \vdash U}}{C \vdash [-]U}}{(\mathbf{1})\;\; \{B_i \,:\, i \geq 0\} \vdash U}}{\{B_i \,:\, i \geq 0\} \vdash [-]U}}{\{B_i \,:\, i \geq 0\} \vdash U}
$$

The final node is terminal because of the node labelled $(\mathbf{1})$ above it, and it is unsuccessful because U is a least fixed point constant. The sequent at node $(\mathbf{1})$ is provable if we permit induction on top of the proof system by showing that each B_i has the property U. However we would like to avoid explicit induction principles.

The property checker as it stands therefore fails to guarantee that there is a successful tableau for temporal properties of infinite state processes. We now rectify this by including one more condition for termination. The additional circumstance for being a leaf node of a proof tree concerns least fixed point constants. A node $\mathbf{n}$ labelled by the sequent $\mathcal{F} \vdash U$ when $U \stackrel{\rm def}{=} \mu Z.\Phi$ is a terminal if there is a node above $\mathbf{n}$ labelled $\mathcal{E} \vdash U$ and $\mathcal{F} \subseteq \mathcal{E}$. This almost repeat circumstance is similar to the terminal condition 3 of the previous section except here it is with respect to a least fixed point, and $3'$ except the condition is $\mathcal{F} \subseteq \mathcal{E}$. Some nodes obeying this new condition are successful while the rest are not. But the definition of success here (taken from [20]) is intricate, and requires some notation.

A leaf which obeys condition 3 of being a terminal from the previous section or the new terminal condition above is called a *σ-terminal*, where σ may be instantiated by a ν or μ depending on the kind of constant involved. Suppose node $\mathbf{n}'$ is an immediate successor of $\mathbf{n}$, and $\mathbf{n}$ is labelled by $\mathcal{E} \vdash \Phi$ and and $\mathbf{n}'$ is labelled $\mathcal{E}' \vdash \Phi'$. We say that $E' \in \mathcal{E}'$ at $\mathbf{n}'$ is a *dependant* of $E \in \mathcal{E}$ at $\mathbf{n}$ if

- the rule applied to $\mathbf{n}$ is $\wedge$, $\vee$, U, σZ, or Thin, and $E = E'$, or
- the rule is $[K]$ and $E \xrightarrow{a} E'$ for some $a \in K$, or
- the rule is $\langle K \rangle$, and $E' = f(E)$ where f is the choice function.

For instance, each B_i at node $(\mathbf{1})$ is a dependant of C at the node directly above it in the example earlier. Assume that the *companion* of a σ-terminal is the most recent node above it which makes it a terminal. (There may be more than one node above a σ-terminal which makes it a leaf, hence we take the lowest one.) Next we define the notion of a trail.

Definition 1 A *trail* to E_k at a μ-terminal $\mathbf{n}_k$ from E_1 at its companion $\mathbf{n}_1$ is a sequence of pairs of nodes and processes $(\mathbf{n}_1, E_1), \ldots, (\mathbf{n}_k, E_k)$ such that for all i with $1 \leq i < k$ either E_{i+1} at $\mathbf{n}_{i+1}$ is a dependant of E_i at $\mathbf{n}_i$, or $\mathbf{n}_i$

is the immediate predecessor of a σ-terminal node $\mathbf{n}'$ (where $\mathbf{n}' \neq \mathbf{n}_k$) whose companion is $\mathbf{n}_j$ for some $j \geq 1$ and $j \leq i$, and $\mathbf{n}_{i+1} = \mathbf{n}_j$ and E_{i+1} at $\mathbf{n}'$ is a dependant of E_i at $\mathbf{n}_i$.

A simple trail occurs when it is carved out from the companion to the leaf. A more involved circumstance is when there is another companion on this path and the trail then proceeds via (the node above) its leaf before continuing from its companion. This can be iterated when there are further companion nodes.

Each companion node $\mathbf{n}$ of a μ-terminal induces a pre-order $\sqsupset_{\mathbf{n}}$ where $E \sqsupset_{\mathbf{n}} F$ if there is a trail to F at $\mathbf{n}$ from E at its companion.

Definition 2 A μ-terminal node $\mathbf{m}$ is *successful* if the ordering $\sqsubset_{\mathbf{n}}$ induced by its companion $\mathbf{n}$ is well-founded: there is no infinite descending chain $E_0 \sqsupset_{\mathbf{n}} E_1 \sqsupset_{\mathbf{n}} \cdots$.

Finally a tableau is *successful* if it is finite and all its leaves obey one of the conditions for being a successful terminal (either 1,2, or 3 from the previous section, or that of being a successful μ-terminal).

The tableau technique is both *sound* and *complete* for arbitrary (infinite) transition systems. Again the result is proved using the game characterization of satisfaction.

Theorem 1 *$\mathcal{E} \vdash_{\mathcal{V}} \Phi$ has a successful tableau iff $\mathcal{E} \models_{\mathcal{V}} \Phi$.*

This tableau proof system was first presented in [17, 20] where direct proofs of the result are given. Of course it can be highly undecidable whether or not a tableau is successful. For there can not be an effective proof system for checking temporal properties of arbitrary infinite state processes. One can code up the behaviour of a Turing machines as a process, and in such a case the formula $\mu Z.[-]Z$ expresses the halting capability. However there are some interesting infinite state cases of when property checking is decidable [41]. Anyway tableaux techniques provide a suitable basis for interactive computer aided verification as illustrated in [18], in contrast with automatic techniques [28, 38, 5, 31, 2, 40, 57].

Example 1 The tableau at the start of this section is now successful. The companion of the leaf is the node labelled (**1**). The ordering $\sqsubset_{\mathbf{1}}$ is well founded as $B_{i+1} \sqsupset_{\mathbf{n}} B_i$ for any $i \geq 0$: therefore there can not be an infinite sequence $B_{j1} \sqsupset_{\mathbf{n}} B_{j2} \ldots$. Suppose we change the definition of B_{i+1} as follows: $B_{i+1} \stackrel{\rm def}{=} \mathtt{down}.B_i + \mathtt{round}.B_{i+2}$. Each B_{i+1} has the extra capability for performing $\mathtt{round}$. An attempt to prove that C eventually terminates yields the same tableau. However it is now unsuccessful as the relation $\sqsubset_{\mathbf{1}}$ is not well founded as $B_{i+1} \sqsupset_{\mathbf{n}} B_{i+1}$. □

Example 2 The liveness property of the crossing subject to the assumption of fairness of the signal, as we saw in example 3 of section 3.7, is expressed by the following open formula $\nu Y.[\mathtt{car}]\Psi_1 \wedge [-]Y$, where Ψ_1 is

$$\mu X.\nu Y_1.(Q \vee [-\overline{\mathtt{ccross}}](\nu Y_2.(R \vee X) \wedge [-\overline{\mathtt{ccross}}]Y_2)) \wedge [-\overline{\mathtt{ccross}}]Y_1$$

and where Q holds when the crossing is in any state where *Rail* can by itself perform `green` and R holds in any state where *Road* can by itself perform `up`. We employ the abbreviations in figure 8, and we let $\mathcal{E}$ be the full set $\{E_0, \ldots, E_{11}\}$. The states that Q holds at are E_2, E_3, E_6, and E_{10} and R holds at E_1, E_3, E_7, and E_{11}. A proof that the crossing has this property is given in stages in figure 20. In this tableau there is one μ-terminal, labelled **(1)** whose companion is (**c**). The relation $\sqsubset_{\mathbf{c}}$ is well founded as we only have: $E_1 \sqsupset_{\mathbf{c}} E_4$, $E_4 \sqsupset_{\mathbf{c}} E_6$, and $E_3 \sqsupset_{\mathbf{c}} E_6$. Therefore the tableau is successful. □

Example 3 The verification that the slot machine has the weak liveness property that a million pounds can be won infinitely often is given by the following successful tableau:

$$\dfrac{\{SM_n \,:\, n \geq 0\} \vdash \nu Y.\mu Z.\langle\overline{\texttt{win}}(10^6)\rangle Y \vee \langle - \rangle Z}{\dfrac{\mathcal{E} \vdash \nu Y.\mu Z.\langle\overline{\texttt{win}}(10^6)\rangle Y \vee \langle - \rangle Z}{\dfrac{\mathcal{E} \vdash U}{\dfrac{\mathcal{E} \vdash \mu Z.\langle\overline{\texttt{win}}(10^6)\rangle U \vee \langle - \rangle Z}{\dfrac{(1)\ \mathcal{E} \vdash V}{\dfrac{(2)\ \mathcal{E} \vdash \langle\overline{\texttt{win}}(10^6)\rangle U \vee \langle - \rangle V}{\dfrac{\mathcal{E}_1 \vdash \langle\overline{\texttt{win}}(10^6)\rangle U}{\mathcal{E}_1' \vdash U} \quad \dfrac{(3)\ \mathcal{E}_2 \vdash \langle - \rangle V}{(4)\ \mathcal{E}_2' \vdash V}}}}}}}$$

$\mathcal{E}$ is the set of all derivatives. The vital rules in this tableau are the disjunction at node (1), where $\mathcal{E}_1$ is exactly those processes capable of performing a $\overline{\texttt{win}}(10^6)$ action, and $\mathcal{E}_2$ is the remainder; and the $\langle K \rangle$ rule at node (3), where f is defined to ensure that $\mathcal{E}_1$ is eventually reached: for process with less than 10^6 in the bank, f chooses events leading towards loss, so as to increase the amount in the bank; and for processes with more than 10^6, f chooses to $\overline{\texttt{release}}(10^6)$. The formal proof requires partitioning $\mathcal{E}_2$ into several classes, each parametrized by an integer n, and showing that while $n < 10^6$, n is strictly increasing over a cycle through the classes; then when $n = 10^6$, f selects a successor that is not in $\mathcal{E}_2$, and so a chain from E_0 through nodes (1), (2), (3), (4) terminates, and therefore $\sqsubset_{(1)}$ is well-founded. □

Example 4 Consider the following family of processes for $i \geq 1$ from section 1.1:

$$T(i) \stackrel{\text{def}}{=} \textbf{if } even(i) \textbf{ then } \overline{\texttt{out}}(i).T(i/2) \textbf{ else } \overline{\texttt{out}}(i).T((3i+1)/2)$$

If $T(n)$ for all $n \geq 1$ stabilizes into the cycle $T(2) \stackrel{\overline{\texttt{out}}(2)}{\longrightarrow} T(1) \stackrel{\overline{\texttt{out}}(1)}{\longrightarrow} T(2)$ then the following tableau is successful, and otherwise it is not. But which of these holds is not known!

$$\dfrac{\{Crossing\} \vdash \nu Y.[\mathtt{car}]\Psi_1 \wedge [-]Y}{\dfrac{\mathcal{E} \vdash \nu Y.[\mathtt{car}]\Psi_1 \wedge [-]Y}{\dfrac{\mathcal{E} \vdash U}{\dfrac{\mathcal{E} \vdash [\mathtt{car}]\Psi_1 \wedge [-]U}{\dfrac{\mathcal{E} \vdash [\mathtt{car}]\Psi_1}{\dfrac{\{E_1, E_3, E_7, E_{11}\} \vdash \Psi_1}{T1}} \quad \dfrac{\mathcal{E} \vdash [-]U}{\mathcal{E} \vdash U}}}}}$$

$$\mathcal{E}_1 = \{E_1, E_3, E_4, E_6, E_7, E_{11}\}$$

$$\dfrac{T1}{\dfrac{\mathcal{E}_1 \vdash \Psi_1}{\dfrac{(\mathbf{c})\ \mathcal{E}_1 \vdash V}{\dfrac{\mathcal{E}_1 \vdash \nu Y_1.(Q \vee [-\overline{\mathtt{ccross}}](\nu Y_2.(R \vee V) \wedge [-\overline{\mathtt{ccross}}]Y_2)) \wedge [-\overline{\mathtt{ccross}}]Y_1}{\dfrac{\mathcal{E}_1 \vdash U_1}{\dfrac{\mathcal{E}_1 \vdash Q \vee [-\overline{\mathtt{ccross}}](\nu Y_2.(R \vee V) \wedge [-\overline{\mathtt{ccross}}]Y_2) \wedge [-\overline{\mathtt{ccross}}]U_1}{\dfrac{\mathcal{E}_1 \vdash Q \vee [-\overline{\mathtt{ccross}}](\nu Y_2.(R \vee V) \wedge [-\overline{\mathtt{ccross}}]Y_2)}{\{E_3, E_6\} \vdash Q \quad T2} \quad \dfrac{\mathcal{E}_1 \vdash [-\overline{\mathtt{ccross}}]U_1}{\mathcal{E}_1 \vdash U_1}}}}}}}$$

$$\dfrac{T2}{\dfrac{\{E_1, E_4, E_7, E_{11}\} \vdash [-\overline{\mathtt{ccross}}](\nu Y_2.(R \vee V) \wedge [-\overline{\mathtt{ccross}}]Y_2)}{\dfrac{\{E_1, E_3, E_4, E_6, E_{11}\} \vdash \nu Y_2.(R \vee V) \wedge [-\overline{\mathtt{ccross}}]Y_2}{\dfrac{\{E_1, E_3, E_4, E_6, E_7, E_{11}\} \vdash \nu Y_2.(R \vee V) \wedge [-\overline{\mathtt{ccross}}]Y_2}{\dfrac{\{E_1, E_3, E_4, E_6, E_7, E_{11}\} \vdash U_2}{\dfrac{\{E_1, E_3, E_4, E_6, E_7, E_{11}\} \vdash (R \vee V) \wedge [-\overline{\mathtt{ccross}}]U_2}{\dfrac{\{E_1, E_3, E_4, E_6, E_7, E_{11}\} \vdash R \vee V}{\{E_1, E_3, E_7, E_{11}\} \vdash R \quad (\mathbf{1})\ \{E_4, E_6\} \vdash V} \quad \dfrac{\{E_1, E_3, E_4, E_6, E_7, E_{11}\} \vdash [-\overline{\mathtt{ccross}}]U_2}{\{E_1, E_3, E_4, E_6, E_7, E_{11}\} \vdash U_2}}}}}}}$$

Fig. 20.

$$
\dfrac{\dfrac{\dfrac{\{T(i)\,:\,i \geq 1\} \vdash \mu Y.\, \langle \overline{\texttt{out}}(2) \rangle \texttt{tt} \vee [-]Y}{\{T(i)\,:\,i \geq 1\} \vdash U}}{\{T(i)\,:\,i \geq 1\} \vdash \langle \overline{\texttt{out}}(2) \rangle \texttt{tt} \vee [-]U}}{\dfrac{T(2) \vdash \langle \overline{\texttt{out}}(2) \rangle \texttt{tt}}{T(1) \vdash \texttt{tt}} \quad \dfrac{\{T(i)\,:\,i \geq 1 \wedge i \neq 2\} \vdash [-]U}{\{T(i)\,:\,i > 1\} \vdash U}}
$$

The problem is that we do not know if the relation induced by the companion of this leaf is well-founded. □

4.6 Exposing structure

An important topic is to what extent verification can be guided by the theory of processes. Game playing and the tableau proof rules are directed by the logical structure of the property. A simple case of where the theory of processes may impinge is the following proof rule that can be added to the tableaux proof systems of sections 4.2 and 4.3 when Φ is a closed formula.

$$
\dfrac{E \vdash \Phi}{F \sim E \quad F \vdash \Phi}
$$

This is justified because as we saw in section 3.6 bisimulation equivalence preserves temporal properties. Moreover if Φ belongs to weak modal mu-calculus then we only need the subgoal $F \approx E$. Use of this rule could appeal to the theory of bisimulation, and techniques for minimizing process descriptions. Alternatively it could appeal to the equational theory of processes.

Process behaviour is chronicled through transitions. But processes also have structure, defined as they are from combinators. To what extent can process properties be defined without appealing to transitional behaviour, but instead to this algebraic structure? The ascription of boolean combinations of properties to processes does not directly depend on their behaviour: for instance, E satisfies $\Phi \vee \Psi$ provided it satisfies one of the disjuncts. Consequently it is the modal (and fixed point) operators that we need to concern ourselves with, how algebraic structure relates to them. Some simple cases are captured in the following lemma.

Lemma 1 i. *if* $a \notin K$ *then* $a.E \models [K]\Phi$ *and* $a.E \not\models \langle K \rangle \Phi$, ii. *if* $a \in K$ *then* $a.E \models [K]\Phi$ *iff* $E \models \Phi$, iii. *if* $a \in K$ *then* $a.E \models \langle K \rangle \Phi$ *iff* $E \models \Phi$, iv. $\sum\{E_i : i \in I\} \models [K]\Phi$ *iff for all* $j \in I$. $E_j \models [K]\Phi$, v. $\sum\{E_i : i \in I\} \models \langle K \rangle \Phi$ *iff for some* $j \in I$. $E_j \models \langle K \rangle \Phi$, vi. *if* $P \stackrel{\text{def}}{=} E$ *and* $E \models \Phi$ *then* $P \models \Phi$.

Example 1 Using Lemma 1 and the semantic clauses for boolean combinations of properties we can now show that the vending machine *Ven* of figure 2 has the property $[\texttt{2p}](\langle - \rangle \texttt{tt} \wedge [-\texttt{big}]\texttt{ff})$ without refering to transitions. From Lemma 1 vi. this is established if $\texttt{2p}.\mathit{Ven}_b + \texttt{1p}.\mathit{Ven}_l \models [\texttt{2p}](\langle - \rangle \texttt{tt} \wedge [-\texttt{big}]\texttt{ff})$, which reduces by case iv. to demonstrating $\texttt{2p}.\,Ven_b \models [\texttt{2p}](\langle - \rangle \texttt{tt} \wedge [-\texttt{big}]\texttt{ff})$ and $\texttt{1p}.\,Ven_l \models [\texttt{2p}](\langle - \rangle \texttt{tt} \wedge [-\texttt{big}]\texttt{ff})$. The second of these follows from Lemma 1 i. as $\texttt{1p} \notin \{\texttt{2p}\}$. The first by case ii. reduces to showing $Ven_b \models \langle - \rangle \texttt{tt} \wedge [-\texttt{big}]\texttt{ff}$. By vi. this is true if $\texttt{big.collect}_b.\,Ven \models \langle - \rangle \texttt{tt} \wedge [-\texttt{big}]\texttt{ff}$. That is, if the following pair hold $\texttt{big.collect}_b.\,Ven \models \langle - \rangle \texttt{tt}$ and $\texttt{big.collect}_b.\,Ven \models [-\texttt{big}]\texttt{ff}$. The second of these holds by Proposition 1 i. and the first is established using ii. as $\texttt{collect}_b.\,Ven \models \texttt{tt}$. □

Restriction $\backslash J$ is dealt with by defining its effect on formulas Φ as $\Phi\backslash J$ inductively as follows where $J^{\#}$ is the set $J \cup \overline{J}$:

$$\begin{array}{llll}
(\Phi \wedge \Psi)\backslash J &= \Phi\backslash J \wedge \Psi\backslash J & (\Phi \vee \Psi)\backslash J &= \Phi\backslash J \vee \Psi\backslash J \\
([K]\Phi)\backslash J &= [K - J^{\#}](\Phi\backslash J) & (\langle K \rangle\Phi)\backslash J &= \langle K - J^{\#}\rangle(\Phi\backslash J) \\
\nu Z.\,\Phi\backslash J &= \nu Z.\,(\Phi\backslash J) & \mu Z.\,\Phi\backslash J &= \mu Z.\,(\Phi\backslash J)
\end{array}$$

and where $\Phi\backslash J = \Phi$ when Φ is Z, $\texttt{tt}$, or $\texttt{ff}$. The operator $\backslash J$ on formulas is an *inverse* of its application to processes, $E\backslash J \models \Phi$ iff $E \models \Phi\backslash J$. Similar inverse operators on formulas can be defined for renaming $[f]$ and hiding $\backslash\backslash J$. This forms the basis for a notion of abstraction which can be used in some cases to simplify verification, see [22].

Using these results we can extend the tableau proof rules (and game playing) by adding rules such as:

$$\frac{E + F \vdash [K]\Phi}{E \vdash [K]\Phi \quad F \vdash \Phi} \qquad \frac{E\backslash\backslash J \vdash \Phi}{E \vdash \Phi\backslash\backslash J}$$

The harder question is how to deal with parallel, some proposals can be found in [75, 55, 3, 4].

References

1. Abramsky, S., (1994). Interaction categories, this workshop.
2. Andersen, H., (1992). Model checking and boolean graphs. *Lect. Notes in Comp. Science*, **582**.
3. Andersen, H., and Winskel, G. (1991). Compositional checking of satisfaction. *Lect. Notes in Comp. Science*, **575**.
4. Andersen, H., Stirling, C., and Winskel, G. (1994) A compositional proof system for the modal mu-calculus. *Procs 9th IEEE Symp. on Logic in Comput. Sci.*, 144-153.
5. Arnold, A., and Crubille, P. (1988). A linear algorithm to solve fixed-point equations on transition systems. *Inf. Process. Letters*, 29, 57-66.
6. Arnold, A., and Niwinski, D. (1992). Fixed point characterization of weak monadic logic definable sets of trees. In *Tree Automata and Languages*, ed. M. Nivat and A. Podelski, Elsevier, 159-188.
7. Austry, D., and Boudol, G. (1984). Algebra de processus et synchronisation. *Theoret. Comp. Sci.* **30**, 90-131.
8. Baeten, J., Bergstra, J., and Klop, J. (1987). Decidability of bisimulation equivalence for processes generating context-free languages. *Lect. Notes in Comp. Science*, **259**, 94-113.

9. Baeten, J, and Weijland, W. (1990). *Process Algebra.* Cambridge Univ. Press.
10. Baeten, J., Bergstra, J., Hoare, C., Milner, R., Parrow, J., and de Simone, R., (1992). The variety of process algebra.
11. De Bakker, J. (1980). *Mathematical Theory of Program Correctness*, Prentice-Hall.
12. De Bakker, J., and De Roever, W. (1973). A calculus for recursive program schemes. In *Automata, Languages and Programming*, ed. Nivat, M. 167-196, North-Holland.
13. Bernholtz, O., Vardi, M. and Wolper, P. (1994). An automata-theoretic approach to branching-time model checking. *Procs. CAV.*
14. Bergstra, J. and Klop, J. (1989). Process theory based on bisimulation semantics. *Lect. Notes in Comp. Science*, **354**, 50-122.
15. van Benthem, J. (1984). Correspondence theory. In *Handbook of Philosophical Logic*, Vol. II, ed. Gabbay, D. and Guenthner, F., 167-248, Reidel.
16. Boudol, G. (1985). Notes on algebraic calculi of processes, in *Logics and Models of Concurrent Systems*, Springer.
17. Bradfield, J. (1992). *Verifying Temporal Properties of Systems.* Birkhauser.
18. Bradfield, J. (1992). A proof assistant for symbolic model checking. *Tech. Report* ECS-LFCS-92-199, Dept. of Comp. Science, Edinburgh University.
19. Bradfield, J. and Stirling, C. (1990). Verifying temporal properties of processes. *Lect. Notes in Comput. Science*, **458**, 115-125.
20. Bradfield, J. and Stirling, C. (1992). Local model checking for infinite state spaces. *Theoret. Comput. Science*, **96**, 157-174.
21. Brebner, G. (1991). A CCS-based investigation of deadlock in a multi-process electronic mail system. *Tech. Report* CSR-17-91, Dept. of comput. Science, University of Edinburgh.
22. Bruns, G. (1993). A practical technique for process abstraction. *Lect. Notes in Comput. Science*, **715**, 37-49.
23. Bruns, G., and Anderson, S. (1991). The formalization and analysis of a communications protocol. *Tech. Report* ECS-LFCS-91-137, Dept. of comput. Science, University of Edinburgh.
24. Camilleri, J., and Winskel, G. (1991). CCS with priority. *Procs. LICS*, 246-255.
25. Caucal, D. (1990) Graphes canoniques de graphes algebriques. *Theoretical Informatics and Applications*, 24, 339-352.
26. Christensen, S., Hirshfeld, Y., and Moller, F. (1993) Bisimulation is decidable for all basic parallel processes. To appear in *Procs* CONCUR 93, Springer Verlag.
27. Christensen, S., Hüttel, H., and Stirling, C. (1992). Bisimulation equivalence is decidable for all context-free processes. *Lect. Notes in Comput. Science*, **630**, 138-147.
28. Clarke, E., Emerson, E., and Sistla, A. (1983). Automatic verification of finite state concurrent systems using temporal logic specifications: a practical approach. *Proc.10th ACM Symp. on Principles of Programming Languages*, 117-126.
29. Cleaveland, R. (1990). Tableau-based model checking in the propositional mu-calculus. *Acta Informat.*, **27**, 725-747.
30. Cleaveland, R., and Hennessy, M. (1988). Priorities in process algebra. *Procs. LICS*, 193-202.
31. Cleaveland, R., Dreimuller, M., and Steffen, B. (1992). Faster model checking for the modal mu-calculus. To appear in *Lect. Notes in Comp. Science.*
32. Cleaveland, R, Parrow, J, and Steffen, B. (1989). The concurrency workbench. In *Proc. IFIP.*

33. Dam, M. (1992). CTL* and ECTL* as fragments of the modal μ-calculus. *Lect. Notes in Comput. Sci.* **581**, 145-164, Springer.
34. De Nicola, R. (1987). Extensional equivalences for transition systems. *Acta Informatica*, **24**, 211-237.
35. De Nicola, R. and Vaandrager, V. (1990). Three logics for branching bisimulation. *Proc. 5th IEEE Symp. on Logic in Comput. Sci.*, 118-129.
36. Emerson, E, and Clarke, E. (1980). Characterizing correctness properties of parallel programs using fixpoints. *Lect. Notes in Comput. Sci.* **85**, 169-181.
37. Emerson, E., and Jutla, C. (1991). Tree automata, mu-calculus and determinacy. In *Procs. Found. of Comput. Science.*
38. Emerson, E, and Lei, C. (1986). Efficient model checking in fragments of the propositional mu-calculus. In *Proc. 1st IEEE Symp. on Logic in Comput. Sci.*, 267-278.
39. Emerson, E, and Srinivasan, J. (1989). Branching time temporal logic. *Lect. Notes in Comput. Sci.* **354**, Springer.
40. Emerson, E., Jutla, C., and Sistla, A. (1993). On model checking for fragments of μ-calculus. *Procs. CAV 93, Lect. Notes in Comput. Sci.*, 385-396.
41. Esparza, J. (1994). On the decidability of model checking for several μ-calculi and Petri nets, *Lect. Notes in Comput. Sci.* **787**, 115-129.
42. Fitting, M. (1983). *Proof Methods for Modal and Intuitionistic Logics.* Reidel.
43. van Glabbeek, J. F., and Weijland, W.P. (1989) Branching time and abstraction in bisimulation semantics. *Inf. Processing*, 89, 613-618.
44. Groote, J. and Vaandrager, F. (1989). Structured operational semantics and bisimulation as a congruence. *Lect. Notes in Comput. Science*, **372**, 423-438.
45. Hennessy, M. (1988). *An Algebraic Theory of Processes.* MIT Press.
46. Hennessy, M. and Milner, R. (1980). On observing nondeterminism and concurrency. *Lect. Notes in Comp. Sci.* **85**, 295-309.
47. Hennessy, M. and Milner, R. (1985). Algebraic laws for nondeterminism and concurrency. *J. Ass. of Comput. Mach.* **32**, 137-162.
48. Hoare, C. (1985). *Communicating Sequential Processes.* Prentice Hall.
49. Hüttel, H. (1990). SnS can be modally characterized. *Theoret. Comput. Sci.*, **74**, 239-248.
50. Kaivola, R. (1994). On modal mu-calculus and Büchi tree automata. *Tech. Report*, Edinburgh University.
51. Kannellakis, P. and Smolka, S. (1990). CCS expressions, finite state processes, and three problems of equivalence. *Inf. and Comp.*, **86**, 43-68.
52. Kozen, D. (1983). Results on the propositional mu-calculus. *Theoret. Comput. Sci* **27**, 333-354.
53. Lamport, L. (1983) Specifying concurrent program modules. *ACM TOPLASS*, 5, 190-222.
54. Larsen, K. (1990). Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoret. Comput. Science*, **72**, 265-288.
55. Larsen, K. (1990). Ideal specification formalism. *Lect. Notes in Comput. Science* **458**, 33-56.
56. Larsen, K. and Skou. (1989). Bisimulation through probabilistic testing. In *Principles of Programming Languages.*
57. Long, D., Browne, A., Clarke, E., Jha, S., and Marrero, W. (1994) An improved algorithm for the evaluation of fixpoint expressions. *Procs. CAV.*
58. Manna, Z. and Pnueli, A. (1989). The anchored version of the temporal framework. *Lect. Notes in Comput. Science*, **354**, 201-284.

59. Manna, Z, and Pnueli, A. (1991). *The Temporal Logic of Reactive and Concurrent Systems.* Springer.
60. Milner, R. (1983). Calculi for synchrony and asynchrony. *Theoret. Comput. Science*, **25**, 267-310.
61. Milner, R. (1989). *Communication and Concurrency.* Prentice Hall.
62. Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, Parts I and II, *Inf. and Comp.*, 100, 1-77.
63. Moller, F. (1991). The Specification of an asynchronous router. Manuscript, University of Edinburgh.
64. Moller, F. and Tofts, C. (1990). A temporal calculus of communicating processes.*Lect. Notes in Comput. Sci.*, **458**, 401-415.
65. Morley, M., (1991). Modelling british rail's interlocking logic: geographic data correctness. *Tech. Report* ECS-LFCS-91-186, Dept. Comput. Science, Edinburgh University.
66. Muller, D., Saoudi, A. and Schupp, P. (1986). Alternating automata, the weak monadic theory of the tree and its complexity. *Lect. Notes in Comput. Sci.*, **225**, 275-283.
67. Nicollin, X. and Sifakis, J. (1992). An overview and synthesis on timed process algebras. *Lect. Notes in Comput. Sci.*, **575**, 376-398.
68. Park, D. (1969). Fixpoint induction and proofs of program properties. Machine Intelligence **5**, 59-78,Edinburgh University Press
69. Park, D. (1981). Concurrency and automata on infinite sequences. *Lect. Notes in Comput. Sci.*, **154**, 561-572.
70. Parrow, J. (1987). Verifying a CSMA/CD-Protocol with CCS. In *Procs. 7th IFIP Symp. on Prot. Spec., Testing, and Verific.*, North-Holland.
71. Plotkin, G. (1981). A structural approach to operational semantics. *Tech Report*, Aarhus University.
72. Pratt, V. (1982). A decidable mu-calculus, *22nd IEEE Symp. on Found. of Comp. Sci.*, 421-427.
73. Rabin, M. (1969). Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, **141**, 1-35.
74. Simone, R. de (1985). Higher-level synchronizing devices in Meije-SCCS. *Theoret. Comput. Science* **37**, 245-267.
75. Stirling, C. (1987). Modal logics for communicating systems, *Theoret. Comput. Sci.* **49**, 311-347.
76. Stirling, C. (1991). An introduction to modal and temporal logics for CCS. *Lect. Notes in Comput. Sci.* **491** 2-20.
77. Stirling, C. (1992) Modal and temporal logics. In *Handbook of Logic in Computer Science* Vol. 2, ed. Abramsky, S, Gabbay, D, and Maibaum, T. pp477-563, Oxford University Press.
78. Stirling, C. and Walker, D. (1991). Local model checking in the modal mu-calculus. *Theoret. Comput. Sci.*, **89**, 161-177.
79. Streett, R. and Emerson, E. (1989). An automata theoretic decision procedure for the propositional mu-calculus. *Inf. and Comp.*, **81**, 249-264.
80. Taubner, D. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets. Lect. Notes in Comput. Sci* **369**, Springer.
81. Walker, D. (1987). Introduction to a calculus of communicating systems. Tech. Report ECS-LFCS-87-22, Dept. of Computer Science, Edinburgh University.
82. Walker, D. (1989). Automated analysis of mutual exclusion algorithms using CCS. *Formal Aspects Comput.* **1** 273-292.

83. Walker, D. (1990). Bisimulations and divergence. *Inf. and Comput.* **85**, 202-241.
84. Winskel, G. (1988). A category of labelled Petri Nets and compositional proof system. *Procs 3rd IEEE Symp. on Logic in Comput. Sci.*, 142-153.
85. Winskel, G. (1989). A note on model checking the modal ν-calculus. *Lect. Notes in Comput. Science* **372**, 761-772.
86. Wolper, P. (1983). Temporal logic can be more expressive. *Info. and Control*, 56, 72-99.