Modal and Temporal Logics for Processes

Colin Stirling

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

Preface

We examine modal and temporal logics for processes. 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 process behaviour. Various combinations of processes are reviewed.

In section 2 modal 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 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.

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 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 several examples. Finally, concluding comments are contained in section 5.

Acknowledgements: Earlier versions of this paper have been presented 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 Summer School in Logic Methods in Concurrency, Aarhus University, 1993. I would like to thank the organisers and the participants of these summer schools, and of the Banff higher order workshop. I would also like to thank Julian Bradfield for use of his Tex tree constructor for building derivation trees and Carron Kirkwood, Faron Moller, Perdita Stevens 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 the action a, feature prominently, underpinning the behavioural meaning of a process. Structured rules guide their derivation in the sense that the transitions of a compound process are determined by those of its components. Families of transitions can be arranged as labelled graphs, concrete summaries of the behaviour of processes. Here we 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 [40]) is a clock that perpetually ticks, $Cl \overset{\text{def}}{=} \texttt{tick}.Cl$. We adopt the usual convention that names of actions such as `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 $\overset{\text{def}}{=}$, 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 `tick` and in so doing becomes Cl again. This is deducible from the rules for transitions.

First is the axiom for the prefix operator when a is an action and E a process:

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

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 $\texttt{tick}.Cl \xrightarrow{\texttt{tick}} Cl$. Next is the transition rule for $\overset{\text{def}}{=}$, which is presented as a *goal directed* inference rule:

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

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

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

$$\frac{Cl \xrightarrow{\texttt{tick}} Cl}{\texttt{tick}.Cl \xrightarrow{\texttt{tick}} Cl} \; Cl \overset{\text{def}}{=} \texttt{tick}.Cl$$

This *proof* of $Cl \xrightarrow{\texttt{tick}} Cl$ is presented with the desired conclusion as antecedent which follows from the axiom instance beneath it via an application of $\text{R}(\overset{\text{def}}{=})$.

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

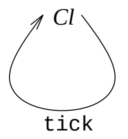


Fig. 1. The transition graph for Cl

directed labelled arcs between them. Each vertex is a process term, and one of them is Cl which can be thought of as the root. All the possible transitions from each vertex, 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

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

Fig. 2. A vending machine

CCS, Calculus of Communicating Systems, [52]. 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[1]:

$$\frac{Ven \xrightarrow{\texttt{2p}} Ven_b}{\dfrac{\texttt{2p}.Ven_b + \texttt{1p}.Ven_l \xrightarrow{\texttt{2p}} Ven_b}{\texttt{2p}.Ven_b \xrightarrow{\texttt{2p}} Ven_b}}$$

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

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

$$\begin{tikzcd}
& Ven \arrow[dl, "\texttt{2p}"'] \arrow[dr, "\texttt{1p}"] & \\
Ven_b \arrow[d, "\texttt{big}"'] & & Ven_l \arrow[d, "\texttt{little}"] \\
\texttt{collect}_b.Ven \arrow[uur, bend left=60, "\texttt{collect}_\texttt{b}"] & & \texttt{collect}_l\,.Ven \arrow[uul, bend right=60, "\texttt{collect}_\texttt{l}"']
\end{tikzcd}$$

Fig. 3. The 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 [59]. 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

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

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 different vertices: such graphs are *infinite state* 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.

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

A special case is when the indexing set I is empty. By the rule $\mathrm{R}(\sum)$ this process has no transitions as the subgoal can never be fulfilled. In CCS this nil process is abbreviated to $\mathbf{0}$ (and to STOP in Hoare's CSP, Communicating Sequential Processes, [40]). Thus `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:

$$\boxed{\text{R(in)} \quad a(x).E \xrightarrow{a(v)} E\{v/x\} \quad v \in D}$$

where $E\{v/x\}$ is the process term which is the result of replacing all free occurrences of x in E with v[2]. 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 output depends on extra machinery for expression evaluation. Assume that $\text{Val}(e)$ is the data value in D (if there is one) to which e evaluates:

$$\boxed{\text{R(out)} \quad \overline{a}(e).E \xrightarrow{\overline{a}(v)} E \quad \text{Val}(e) = v}$$

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

$$Cop \overset{\text{def}}{=} \texttt{in}(x).\overline{\texttt{out}}(x).Cop$$

For any $v \in D$ the transition $Cop \xrightarrow{\text{in}(v)} \overline{\texttt{out}}(v).Cop$ is derived as follows:

$$\frac{Cop \xrightarrow{\text{in}(v)} \overline{\texttt{out}}(v).Cop}{\texttt{in}(x).\overline{\texttt{out}}(x).Cop \xrightarrow{\text{in}(v)} \overline{\texttt{out}}(v).Cop}$$

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 $\text{Val}(v)$ 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 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 $\mathbb{N}$:

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

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, $\text{Val}(even(i)) = \text{true}$ if i is even and is false otherwise. This allows us to use conditionals in the definition of a process, as exemplified by S which sieves numbers:

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

$$S \overset{\text{def}}{=} \texttt{in}(x).\textbf{if } even(x) \textbf{ then } \overline{\texttt{out}}_e(x).S \textbf{ else } \overline{\texttt{out}}_o(x).S$$

Below are the transition rules for this conditional.

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

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

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

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

$T(5)$ performs the transition sequence $T(5) \xrightarrow{\overline{\texttt{out}}(5)} T(8) \xrightarrow{\overline{\texttt{out}}(8)} T(4) \xrightarrow{\overline{\texttt{out}}(4)} T(2)$, 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 expressed as the process $E \mid F$. The crucial transition rule for $\mid$ which conveys communication is:

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

Consider a potential user of the copier *Cop* of the previous section who first writes a file before sending it through the port $\texttt{in}$:

$$User \overset{\text{def}}{=} \texttt{write}(x).User_x \qquad User_x \overset{\text{def}}{=} \overline{\texttt{in}}(x).User$$

As soon as *User* has written the file v it becomes the process $User_v$ which can communicate with *Cop*. Rule R($\mid$com) licenses the following proof[3]:

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

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

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:

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

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

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

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

The behaviour of the users sharing the copier is not affected by order of parallel subcomponents or by placement of brackets. $(Cop \mid User_{v1}) \mid User_{v2}$ and $User_{v1} \mid (Cop \mid User_{v2})$ have the same capabilities as $Cop \mid (User_{v1} \mid User_{v2})$. These three process expressions have isomorphic transition graphs, and therefore in the sequel we omit brackets between multiple concurrent processes[4].

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 $Cnt \overset{\mathrm{def}}{=} \texttt{up}.(Cnt \mid \texttt{down}.\mathbf{0})$. It can perform $\texttt{up}$ and become $Cnt \mid \texttt{down}.\mathbf{0}$ which in turn can also perform $\texttt{up}$ and thereby becomes $Cnt \mid \texttt{down}.\mathbf{0} \mid \texttt{down}.\mathbf{0}$, 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 [52] (and should be distinguished from transition graphs). A flow graph summarizes the

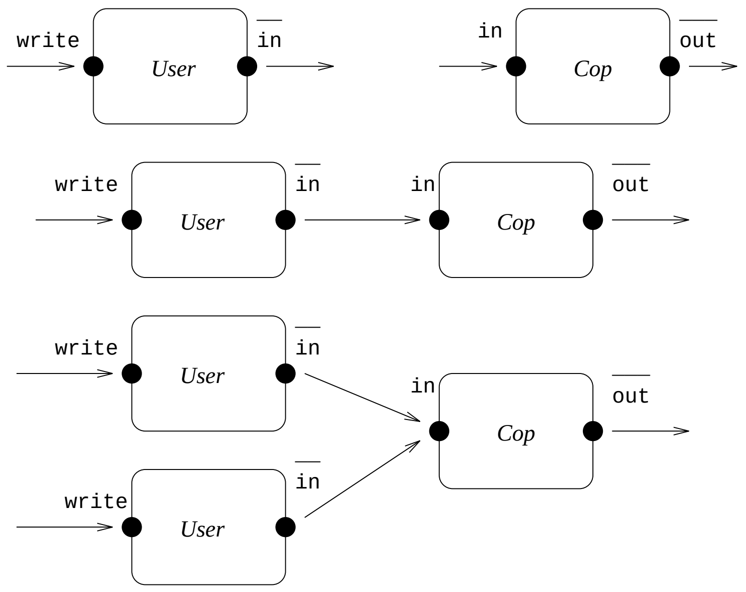


Fig. 5. The flow graphs of $User$, Cop, $Cop \mid User$ and $Cop \mid User \mid User$

potential movement of information flowing into and out of ports, and also exhibits the ports through which a process is in principle willing to communicate. In the case of $User$ the incoming arrow to the port labelled `write` represents input whereas the outgoing arrow from $\overline{\texttt{in}}$ symbolizes output. The flow graph for $Cop \mid User$ has the crucial feature that there is a potential linkage between the output port `in` of $User$ and its input in Cop permitting information to circulate from $User$ to Cop when communication takes place. However this port is still available for other users: both users in $Cop \mid User \mid User$ are able to communicate, at different times, with Cop.

Consider now the situation where a user has private access to a copier. 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 $\{\texttt{in}(v) : v \in D\}$ where D is the space of values that could be accessed through `in`. In the process $(Cop \mid User)\backslash K$ 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.

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

[4] Section 2.4 provides further justification for this.

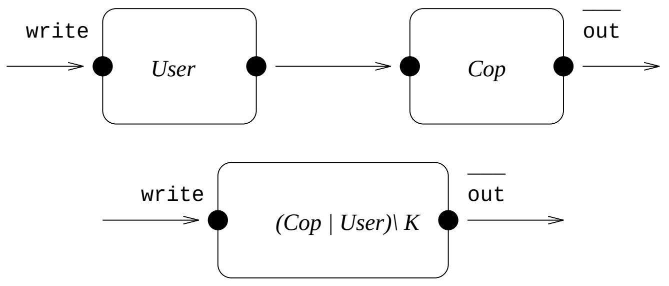


Fig. 6. Flow graphs of $(Cop \mid User)\backslash K$

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

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

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

The second user in $(Cop \mid User)\backslash K \mid User$ 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 $\backslash\{\texttt{in}\}$.

Process descriptions can become quite large, especially when they consist of multiple components in parallel. So we shall employ abbreviations of process expressions using the relation $\equiv$ where $P \equiv F$ means that P 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 [17] consisting of three components. The actions `car` and `train` represent the approach of a car and a train, `up` is the gates opening for the car, $\overline{\texttt{ccross}}$ is the car crossing, `down` closes the gates, `green` is the receipt of a green signal by the train, $\overline{\texttt{tcross}}$ is the train crossing, and `red` automatically sets the light red. Unlike most crossings it keeps the barriers down except when a car actually

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

Fig. 7. A level crossing

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. □

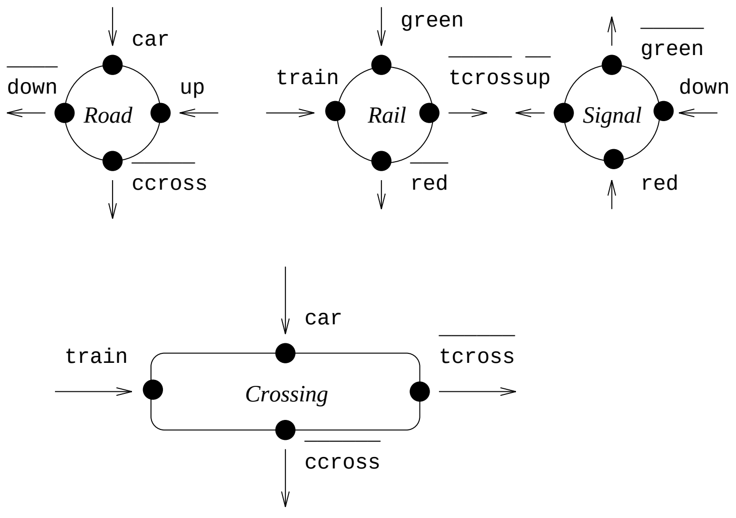
An important arena for the use of process descriptions is modelling protocols [58]. An example is *Protocol* of figure 9 taken from [69] which models a simple communications protocol that allows a message to be lost during transmission. Its flow graph is the same as that of *Cop* earlier, and the size 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 cannot be lost). Having received the acknowledgement, the sender may again receive a message at port `in`.

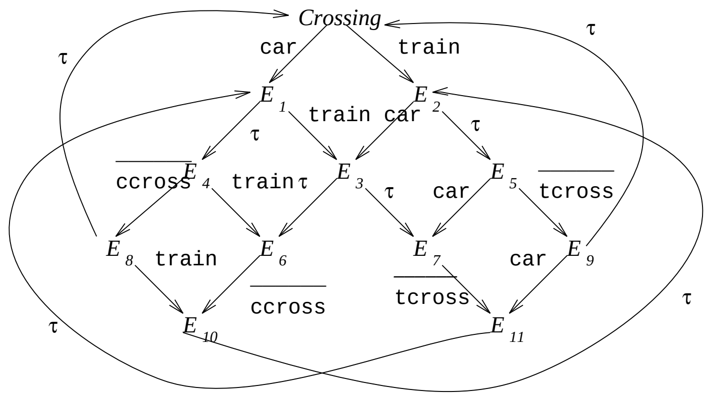
Although the flow graphs for *Protocol* and for *Cop* are the same, their levels of detail are very different. It turns out that these two processes are equivalent in the sense defined in section 2.6. As process descriptions they are very different. *Cop* is close to a specification, as its desired behaviour 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 [17]). 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 of money, B_n a bank holding n pounds, and D the wheel-spinning decision component. □

1.3 Observable 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:





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

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

$$\begin{aligned}
\mathit{Sender} &\stackrel{\text{def}}{=} \mathtt{in}(x).\overline{\mathtt{sm}}(x).\mathit{Send1}(x)\\
\mathit{Send1}(x) &\stackrel{\text{def}}{=} \mathtt{ms}.\overline{\mathtt{sm}}(x).\mathit{Send1}(x) + \mathtt{ok}.\mathit{Sender}\\
\mathit{Medium} &\stackrel{\text{def}}{=} \mathtt{sm}(y).\mathit{Med1}(y)\\
\mathit{Med1}(y) &\stackrel{\text{def}}{=} \overline{\mathtt{mr}}(y).\mathit{Medium} + \tau.\overline{\mathtt{ms}}.\mathit{Medium}\\
\mathit{Receiver} &\stackrel{\text{def}}{=} \mathtt{mr}(x).\overline{\mathtt{out}}(x).\overline{\mathtt{ok}}.\mathit{Receiver}\\[1ex]
\mathit{Protocol} &\equiv (\mathit{Sender} \mid \mathit{Medium} \mid \mathit{Receiver})\backslash\{\mathtt{sm},\mathtt{ms},\mathtt{mr},\mathtt{ok}\}
\end{aligned}$$

Fig. 9. A simple protocol

$$\begin{aligned}
\mathit{IO} &\stackrel{\text{def}}{=} \mathtt{slot}.\overline{\mathtt{bank}}.(\mathtt{lost}.\overline{\mathtt{loss}}.\mathit{IO} + \mathtt{release}(y).\overline{\mathtt{win}}(y).\mathit{IO})\\
B_n &\stackrel{\text{def}}{=} \mathtt{bank}.\overline{\mathtt{max}}(n+1).\mathtt{left}(y).B_y\\
D &\stackrel{\text{def}}{=} \mathtt{max}(z).(\overline{\mathtt{lost}}.\overline{\mathtt{left}}(z).D + \textstyle\sum\{\overline{\mathtt{release}}(y).\overline{\mathtt{left}}(z-y).D \;:\; 1 \le y \le z\})\\[1ex]
\mathit{SM}_n &\equiv (\mathit{IO} \mid B_n \mid D)\backslash\{\mathtt{bank},\mathtt{max},\mathtt{left},\mathtt{release}\}
\end{aligned}$$

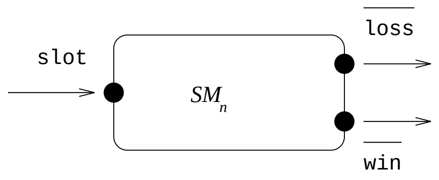


Fig. 10. A slot machine

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

For instance, the crossing of figure 7 performs the cycle $\mathit{Crossing} \xrightarrow{w} \mathit{Crossing}$, when w is $\mathtt{train}\,\tau\,\overline{\mathtt{tcross}}\,\tau$.

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 $C \stackrel{\text{def}}{=} \mathtt{in}(x).\overline{\mathtt{out}}(x).\overline{\mathtt{ok}}.C$, and a user U, defined as $\texttt{write}(x).\overline{\texttt{in}}(x).\texttt{ok}.U$, who writes a file before sending it through $\texttt{in}$ and then waits for an acknowledgement. The behaviour of $(C \mid U)\backslash\{\texttt{in}, \texttt{ok}\}$ is very similar to that of $\mathit{Ucop} \overset{\mathrm{def}}{=} \texttt{write}(x).\overline{\texttt{out}}(x).\mathit{Ucop}$. The only difference in their capabilities 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 $(\texttt{train}\,\tau\,\overline{\texttt{tcross}}\,\tau) \upharpoonright \{\overline{\texttt{tcross}}\} = \overline{\texttt{tcross}}$, and $\tau\,\tau \upharpoonright \{\overline{\texttt{tcross}}\} = \varepsilon$. Associated with any trace w is its *observable* trace, the subsequence $w \upharpoonright \mathcal{O}$ when $\mathcal{O}$ 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 $\texttt{train}\,\tau\,\overline{\texttt{tcross}}\,\tau$ is $\texttt{train}\,\overline{\texttt{tcross}}$. 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 $E \overset{w}{\Longrightarrow} F$ to represent this. A transition rule for such traces utilizes R(tr) and $\upharpoonright$:

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

Observable traces can also be built from their observable components. The extended observable transition $\mathit{Crossing} \overset{\texttt{train}\,\overline{\texttt{tcross}}}{\Longrightarrow} \mathit{Crossing}$ is the result of gluing together the two transitions $\mathit{Crossing} \overset{\texttt{train}}{\Longrightarrow} E$ and $E \overset{\overline{\texttt{tcross}}}{\Longrightarrow} \mathit{Crossing}$ when the intermediate state is either E_2 or E_5 of figure 8. Following [52] 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:

$$\boxed{\mathrm{R}(\overset{\varepsilon}{\Longrightarrow}) \quad E \overset{\varepsilon}{\Longrightarrow} E \qquad \frac{E \overset{\varepsilon}{\Longrightarrow} F}{E \overset{\tau}{\longrightarrow} E' \quad E' \overset{\varepsilon}{\Longrightarrow} F}}$$

$$\boxed{\mathrm{R}(\overset{a}{\Longrightarrow}) \quad \frac{E \overset{a}{\Longrightarrow} F}{E \overset{\varepsilon}{\Longrightarrow} E' \quad E' \overset{a}{\longrightarrow} F' \quad F' \overset{\varepsilon}{\Longrightarrow} F}}$$

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 $\overset{\varepsilon}{\Longrightarrow}$ or $\overset{a}{\Longrightarrow}$ 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 notions of observable equivalence, the graphs may be dramatically simplified as their vertices are fused. In this way the observable transitions of a process offer an abstraction from its 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, $B \overset{\text{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).B$. *Cop* is the process B when port `i` is `in` and `o` is `out` and *User* is B when `i` is `write` and `o` is `in`. Relabelling of ports or actions can be made explicit, as in CCS with a renaming combinator.

The crux of renaming is a function mapping actions into actions. To ensure pleasant algebraic properties (see section 2.7) a renaming function f is subject to a few restrictions. First it should respect complements: for any observable a the actions $f(a)$ and $f(\overline{a})$ are co-actions. Second f should also preserve values passing between ports, $f(a(v))$ is an action $b(v)$ with the same value, and for any other value w the action $f(a(w))$ is $b(w)$. Finally it 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 $E[f]$, and is the process E whose actions are relabelled according to f. Following [52] a renaming function f is usually abbreviated to its essential part: when the a_i are distinct observable 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. For instance *Cop* abbreviates the process $B[\mathtt{in}/\mathtt{i}, \mathtt{out}/\mathtt{o}]$, as 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 the derivations of the following pair of 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, when $n > 1$, following [52], 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* and then internalize the contiguous $\overline{\mathtt{o}}$ and `i` ports. Renaming permits linking as the following variants of B show:

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

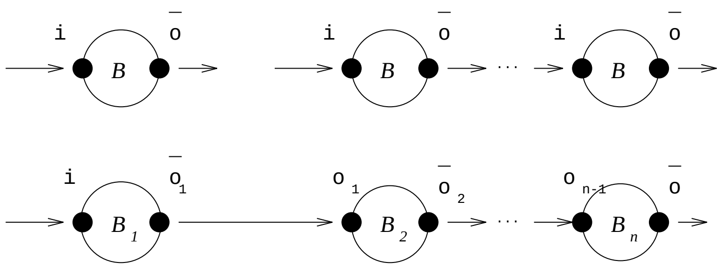


Fig. 11. Flow graph of n instances of B, and $B_1 \mid \ldots \mid B_n$

The flow graph of $B_1 \mid \ldots \mid B_n$ is also depicted in figure 11, and contains the intended links. The n-place buffer is the result of internalizing these links, $(B_1 \mid \ldots \mid B_n)\backslash\{\mathtt{o}_1, \ldots, \mathtt{o}_{n-1}\}$.

A more involved example from [52] 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.

Let Cy' be a cycler of length four, $Cy' \stackrel{\text{def}}{=} a.c.b.d.Cy'$, whose 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 first attempt at building the required scheduler is as a ring of Cy' cyclers.

$$\begin{aligned} Cy_1^{'} &\equiv Cy'[a_1/a, c_1/c, b_1/b, \overline{c}_n/d] \\ Cy_i^{'} &\equiv (d.Cy')[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d] \quad 1 < i \leq n \end{aligned}$$

When n is four the flow graph of the process $Cy_1' \mid Cy_2' \mid Cy_3' \mid Cy_4'$ with initial states marked is depicted in figure 12. Next the c_i actions are internalized. Let $Sched_4'$ abbreviate the process $(Cy_1' \mid Cy_2' \mid Cy_3' \mid Cy_4')\backslash\{c_1, \ldots, c_4\}$. 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 cannot 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 [52] to this problem is to redefine the cycler as follows, $Cy \stackrel{\text{def}}{=} a.c.(b.d.Cy + d.b.Cy)$, and to use the same renaming functions. Let Cy_i for $1 < i \leq n$ be the process $(d.Cy)[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d]$, and let Cy_1 be $Cy[a_1/a, c_1/c, b_1/b, \overline{c}_n/d]$. The required scheduler, $Sched_n$, is the following process, $(Cy_1 \mid \ldots \mid Cy_n)\backslash\{c_1, \ldots, c_n\}$.

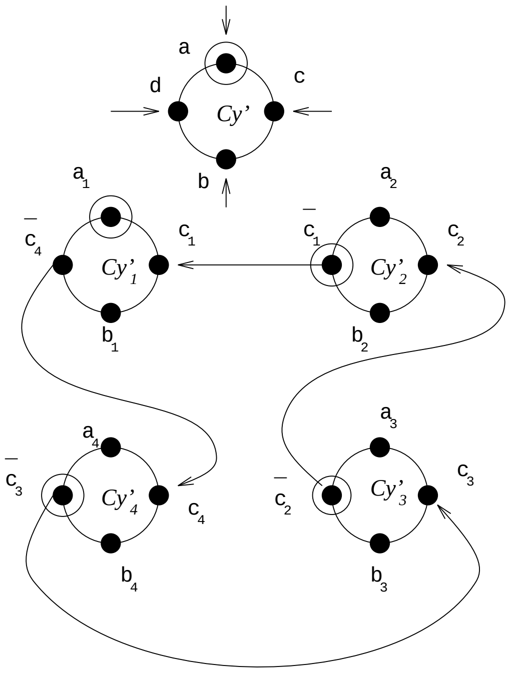


Fig. 12. Flow graph of Cy' and $Cy'_1 \mid Cy'_2 \mid Cy'_3 \mid Cy'_4$

1.5 More combinations of processes

In previous sections we have emphasized the process combinators of CCS. There is a variety of process calculi dedicated to precise modelling of systems. Besides CCS and CSP there is ACP due to Bergstra and Klop [11, 6], Hennessy's EPL [36], MEIJE defined by Austry, Boudol and de Simone [4, 61], Milner's SCCS [51], and Winskel's general process algebra [72]. Although the behavioural meaning of all the operators of these calculi can be presented using inference rules, their conception reflects different concerns, see [7]. ACP is primarily algebraic, highlighting equations[5]. CSP was devised with a distinguished model in mind, the failures model[6], and MEIJE was introduced as a very expressive calculus, initiating general results about families of rules that can be used to define process operators [35]. The general process algebra in [72] has roots in category theory. 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:

[5] See section 2.7.

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

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

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.

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

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. At the other extreme is a pure interleaving operator when the synchronization rule never applies. In between, are 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 in which it is willing to engage. 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

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

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 [55, 54], priorities among actions or among processes

[24, 21], and the description of stochastic behaviour using probabilistic instead of non-deterministic choice [46]. These extensions are useful for modelling hybrid systems which involve a mixture of discrete and continuous, and can be found in chemical plants, and manufacturing.

Processes can also be used to capture foundational models of computation such as Turing machines, counter 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 process is given as a *finite* family of definitions $\{P_i \overset{\mathrm{def}}{=} E_i \ :\ 1 \le i \le n\}$ 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 value passing or infinite sets of indices, their observable behaviour can be described within this restricted process language. Consider, for example, the following finite reformulation [68] of the counter Ct_0, the process *Count*:

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

A two counter machine can simulate a Turing machine, and can be expressed as a process of the form $(Init \mid Fin \mid Count \mid Count')\backslash K$, where *Init* captures initialisation, *Fin* is the finite state control, and the other components are the two counters. The general significance of this is unclear. First there is not a natural formulation of functions within these process calculi, and in particular there is not a simple account of lambda calculus. Second it is not known what the right criteria are for assessing the expressive power of calculi which model concurrency. Should there be a "canonical" process calculus? Is there a concurrent version of the Church-Turing thesis for sequential programs? Some exciting work has been progressing in this area. Two examples are interaction categories [1], and the π-calculus [53].

2 Modalities and Capabilities

Various examples of processes have been presented, from a simple clock to a sophisticated scheduler. In each case a process is 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 $\overset{a}{\Longrightarrow}$, from their thinner counterparts. 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 [38, 39], 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, written $E \models \Phi$. 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:

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

The meanings of modalized formulas appeal to the transition behaviour of a process. 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 $\exists F \in \{E' : E \xrightarrow{\mathtt{tick}} E'\}$. The clock Cl from section 1.1 has this property. 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 $[\texttt{tick}](\langle\texttt{tick}\rangle\texttt{tt} \wedge [\texttt{tock}]\texttt{ff})$[7] expresses the property that after any `tick` action it is possible to perform `tick` again but not possible to perform `tock`.

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

- $\mathit{Ven} \models [2\texttt{p}]([\texttt{little}]\texttt{ff} \wedge \langle\texttt{big}\rangle\texttt{tt})$: after `2p` is deposited the little button cannot be depressed whereas the big one can.
- $\mathit{Ven} \models [\texttt{1p}, \texttt{2p}][\texttt{1p}, \texttt{2p}]\texttt{ff}$: after a coin is entrusted no other coin (`2p` or `1p`) may be deposited.
- $\mathit{Ven} \models [\texttt{1p}, \texttt{2p}][\texttt{big}, \texttt{little}]\langle\texttt{collect}_b, \texttt{collect}_l\rangle\texttt{tt}$: 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 $\mathit{Ven} \models [\texttt{1p}, \texttt{2p}][\texttt{1p}, \texttt{2p}]\texttt{ff}$ iff $\mathit{Ven}_b \models [\texttt{1p}, \texttt{2p}]\texttt{ff}$ and $\mathit{Ven}_l \models [\texttt{1p}, \texttt{2p}]\texttt{ff}$, and clearly both of these hold. Similarly establishing that *Ven* lacks a property, such as $\langle\texttt{1p}\rangle\langle\texttt{1p}, \texttt{big}\rangle\texttt{tt}$, is equally routine. Notice that it is not necessary to *construct* the transition graph of 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 $\langle\overline{\texttt{read}}(5)\rangle\texttt{tt} \wedge [\{\overline{\texttt{read}}(k) \,:\, k \neq 5\}]\texttt{ff}$.

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 $[-]\texttt{ff}$ 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 $\langle-\rangle\texttt{tt} \wedge [-a]\texttt{ff}$. The conjunct $\langle-\rangle\texttt{tt}$ affirms that some action is possible while $[-a]\texttt{ff}$ states that every action except a is impossible. After `2p` is deposited *Ven* must perform `big`, and so $\mathit{Ven} \models [2\texttt{p}](\langle-\rangle\texttt{tt} \wedge [-\texttt{big}]\texttt{ff})$.

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 $\overset{a}{\Longrightarrow}$ permits silent activity before and after a happens: the relation $\overset{a}{\Longrightarrow}$ was defined (see section 1.3) in terms of $\xrightarrow{a}$ and the relation $\overset{\varepsilon}{\Longrightarrow}$ indicating zero or more silent actions.

[7] 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 $\langle\texttt{tick}\rangle\texttt{tt}\wedge[\texttt{tock}]\texttt{ff}$.

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 $\langle\!\langle\ \rangle\!\rangle$:

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

These operators are *not* definable within the modal logic of the previous section. Using them, supplementary modalities $[\![K]\!]$ and $\langle\!\langle K\rangle\!\rangle$ are definable when K is a subset of *observable* actions $\mathcal{O}$.

$$[\![K]\!]\Phi \overset{\text{def}}{=} [\![\ ]\!][K][\![\ ]\!]\Phi \qquad \langle\!\langle K\rangle\!\rangle\Phi \overset{\text{def}}{=} \langle\!\langle\ \rangle\!\rangle\langle K\rangle\langle\!\langle\ \rangle\!\rangle\Phi$$

The derived meanings of these modalities appeal to the observable transition relations $\overset{a}{\Longrightarrow}$ in the same way that their counterparts $[K]$ and $\langle K\rangle$ appeal to $\xrightarrow{a}$. 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 $\langle\!\langle\texttt{tick}\rangle\!\rangle\texttt{tt}$ expresses the observable capability for performing the action $\texttt{tick}$ while $[\![\texttt{tick}]\!]\texttt{ff}$ expresses an inability to tick after any amount of internal activity. Both clocks $Cl' \overset{\text{def}}{=} \texttt{tick}.Cl' + \tau.\mathbf{0}$ and $Cl \overset{\text{def}}{=} \texttt{tick}.Cl$ have the property $\langle\!\langle\texttt{tick}\rangle\!\rangle\texttt{tt}$ but Cl' may at any time silently stop ticking, and therefore it also has the property $\langle\!\langle\texttt{tick}\rangle\!\rangle[\![\texttt{tick}]\!]\texttt{ff}$.

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

Modal formulas can also be used to express notions that are basic to the theory of CSP [40]. A process may perform the observable trace $a_1\ldots a_n$ provided that it has the property $\langle\!\langle a_1\rangle\!\rangle\ldots\langle\!\langle a_n\rangle\!\rangle\texttt{tt}$. The formula $[\![K]\!]\texttt{ff}$ expresses that the observable set of actions K is a *refusal*. The pair $(a_1\ldots a_n, K)$ is an observable *failure* for a process if it has the property $\langle\!\langle a_1\rangle\!\rangle\ldots\langle\!\langle a_n\rangle\!\rangle([\tau]\texttt{ff}\wedge[\![K]\!]\texttt{ff})$: a process 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 $[\![-]\!]\texttt{ff}$ expresses an inability to perform an *observable* action: so the process *Div* has this property when $Div \overset{\text{def}}{=} \tau.Div$.

The modal logic of the previous section permits the expression of immediate necessity or inevitability. The property that a process must perform a next is given by the formula $\langle -\rangle\texttt{tt}\wedge[-a]\texttt{ff}$. However the similarly structured formula $\langle\!\langle -\rangle\!\rangle\texttt{tt}\wedge[\![-a]\!]\texttt{ff}$ 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 $\langle\!\langle -\rangle\!\rangle\texttt{tt}\wedge[\![-\texttt{tick}]\!]\texttt{ff}$: Cl' has this property because it is able to perform an observable transition (so satisfies $\langle\!\langle -\rangle\!\rangle\texttt{tt}$) and is unable to perform any observable action other than $\texttt{tick}$ (and so satisfies $[\![-\texttt{tick}]\!]\texttt{ff}$). But

Cl' may silently break down and therefore be unable to tick. This shortcoming can be surmounted with the strengthened formula $[\![\ ]\!]\,\langle\!\langle -\rangle\!\rangle\mathtt{tt} \wedge [\![-\mathtt{tick}]\!]\,\mathtt{ff}$. Now $Cl' \not\models [\![\ ]\!]\,\langle\!\langle -\rangle\!\rangle\mathtt{tt}$ because of the silent transition $Cl' \overset{\varepsilon}{\Longrightarrow} \mathbf{0}$. But there is still a question mark over this formula as an expression of necessity. Let Cl_1 be a further clock, $Cl_1 \overset{\mathrm{def}}{=} \mathtt{tick}.Cl_1 + \tau.Cl_1$, 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 $\{F \,:\, Cl_1 \overset{\varepsilon}{\Longrightarrow} F\}$ contains the sole element Cl_1 which obeys $\langle\!\langle -\rangle\!\rangle\mathtt{tt}$. But interpreting this as the inevitability that $\mathtt{tick}$ 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 [36], 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:

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

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

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

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 [36], 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. Our discussion has omitted criteria for when two expressions 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 $\mathtt{tick}$ whereas *Ven* can not: 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 descriptions of a possible implementation. Ct_0 is a specification (even the requirement specification) of a counter whereas the finite description *Count* with its very different structure can be seen as a description of a possible implementation. Similarly the buffer *Cop* can be seen as a 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[8].

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, $Cl \stackrel{\text{def}}{=} \texttt{tick}.Cl$ and $Cl_2 \stackrel{\text{def}}{=} \texttt{tick}.\texttt{tick}.Cl_2$. Although they have different transition graphs, whatever transitions one of these clocks makes can be matched by the other and the resulting processes also have this property. An alternative basis for suggesting that these two clocks are equivalent starts with the observation that Cl and $\texttt{tick}.Cl$ should count as equivalent expressions since the second is the definition of Cl. A useful principle is that if two expressions are equivalent then replacing one with the other in some other expression should preserve equivalence. For instance, replacing Cl with $\texttt{tick}.Cl$ in an expression E should result in an expression which is equivalent to E. In particular, if E is $\texttt{tick}.Cl$ then $\texttt{tick}.Cl$ and $\texttt{tick}.\texttt{tick}.Cl$ are equivalent. As 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 which is also a congruence offers the potential 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. Distinguishability can be extended to many other features, such as initial necessities, traces, and deadlock potentials. We can therefore imagine choosing some simple properties as *the* basic distinguishable features, and then stipulating that two processes are equivalent if whenever they are placed in a process context the resultant processes have the same basic properties, and thereby by definition the resulting equivalence is a congruence. This approach is sensitive to three important considerations. First is the choice of what counts as a basic distinguishable property and whether it is to do with observable behaviour as determined by the $\stackrel{a}{\Longrightarrow}$ and $\stackrel{\varepsilon}{\Longrightarrow}$ transitions or the single arrow transitions. Second is the family of process operators that are permitted in process expressions. Finally there is the question as to whether the resulting congruence can be characterized independently of its definition, as preservation of basic properties by all process contexts.

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

Interesting work has been done on this topic, mostly however with respect to the behaviour of processes as determined by the single thin transitions $\xrightarrow{a}$. Candidates for basic distinguishable features include traces and completed traces (given respectively by formulas of the form $\langle a_1\rangle \dots \langle a_n\rangle\mathtt{tt}$ and $\langle a_1\rangle \dots \langle a_n\rangle[-]\mathtt{ff}$). Elegant results are contained within [13, 35, 34] which isolate congruences for traces and completed traces. These cover very general families of process operators whose behavioural meaning is governed by the permissible format of their transition rules. The resulting congruences are independently definable as equivalences[9]. Results for observable behaviour include those for the failures model [40] which takes the notion of observable failure as basic. Related results are contained in the testing framework of [36] where processes are tested as to what they may and must do.

Example 1 Consider the following three vending machines:

$$\begin{aligned}
\mathit{Ven}_1 &\stackrel{\mathrm{def}}{=} \mathtt{1p}.\mathtt{1p}.(\mathtt{tea}.\,\mathit{Ven}_1 + \mathtt{coffee}.\,\mathit{Ven}_1)\\
\mathit{Ven}_2 &\stackrel{\mathrm{def}}{=} \mathtt{1p}.(\mathtt{1p}.\mathtt{tea}.\,\mathit{Ven}_2 + \mathtt{1p}.\mathtt{coffee}.\,\mathit{Ven}_2)\\
\mathit{Ven}_3 &\stackrel{\mathrm{def}}{=} \mathtt{1p}.\mathtt{1p}.\mathtt{tea}.\,\mathit{Ven}_3 + \mathtt{1p}.\mathtt{1p}.\mathtt{coffee}.\,\mathit{Ven}_3
\end{aligned}$$

which have the same (observable) traces. Assume a user, $\mathit{Use} \stackrel{\mathrm{def}}{=} \overline{\mathtt{1p}}.\overline{\mathtt{1p}}.\overline{\mathtt{tea}}.\overline{\mathtt{ok}}.\mathbf{0}$, who only wishes to drink a single tea by offering coins and having done so expresses visible satisfaction as the action $\overline{\mathtt{ok}}$. For each of the three vending machines we can build the process $(\mathit{Ven}_i \mid \mathit{Use})\backslash K$ where K is $\{\mathtt{1p}, \mathtt{tea}, \mathtt{coffee}\}$. When $i = 1$ there is just one completed trace $\tau\,\tau\,\tau\,\overline{\mathtt{ok}}$:

$$(\mathit{Ven}_1 \mid \mathit{Use})\backslash K \xrightarrow{\tau\,\tau\,\tau\,\overline{\mathtt{ok}}} (\mathit{Ven}_1 \mid \mathbf{0})\backslash K$$

and so the user must express satisfaction after some silent activity. In the other two cases there is another completed trace, $\tau\,\tau$:

$$(\mathit{Ven}_i \mid \mathit{Use})\backslash K \xrightarrow{\tau\,\tau} (\mathtt{coffee}.\,\mathit{Ven}_i \mid \overline{\mathtt{tea}}.\overline{\mathtt{ok}}.\mathbf{0})\backslash K$$

The resulting process here is deadlocked. So Ven_1 should not be equivalent to either Ven_2 or Ven_3.

With respect to the failures model of CSP [40] and the testing framework of [36] Ven_2 and Ven_3 are equivalent. Finer equivalences distinguish them on the basis that once a coin has been inserted in Ven_3 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.

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

[9] They include failures equivalence (expressed in terms of formulas of the form $\langle a_1\rangle \dots \langle a_n\rangle[K]\mathtt{ff}$), two-thirds bisimulation, two-nested simulation equivalence, and bisimulation equivalence.

The two processes $(Rep(Ven_i \mid Use))\backslash K$, $i = 2$ and $i = 3$, have different completed traces. When $i = 2$ it has the completed trace $\tau\,\tau\,\tau\,\tau\,\overline{\texttt{ok}}$

$$(Rep(Ven_2 \mid Use))\backslash K \xrightarrow{\tau\,\tau\,\tau\,\tau\,\overline{\texttt{ok}}} (Ven_2 \mid \mathbf{0} \mid \texttt{coffee}.Ven_2 \mid \overline{\texttt{tea}}.\overline{\texttt{ok}}.\mathbf{0})\backslash K$$

which $(Rep(Ven_3 \mid Use))\backslash K$ fails to have. □

2.4 Interactive games and bisimulations

Equivalences for CCS processes begin with the simple idea that an observer can repeatedly interact with a process by choosing an available transition from it. Equivalence of processes is then defined in terms of the ability for these observers to match their selections so that they can proceed with further corresponding choices. The crucial difference with the approach of the previous section is that an observer can choose a *particular* transition. Such choices can not be directly simulated in terms of process activity[10]. These equivalences are defined in terms of bisimulation relations which capture precisely what it is for observers to match their selections. However we proceed with an alternative exposition using *games* which offer a powerful image for interaction.

A pair of processes (E_0, F_0) is an interactive game to be played by two participants, players I and II who are the observers who make choices of transitions. A *play* of the game (E_0, F_0) is a finite or infinite length 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 $(E_0, F_0) \ldots (E_j, F_j)$. The next pair (E_{j+1}, F_{j+1}) is determined by one of the following two moves:

- $\langle\ \rangle$: Player I chooses a transition $E_j \xrightarrow{a} E_{j+1}$ from E_j and then player II chooses a transition with the same label from the other process $F_j \xrightarrow{a} F_{j+1}$.
- $[\ ]$: Player I chooses a transition $F_j \xrightarrow{a} F_{j+1}$ from F_j and then player II chooses a transition with the same label from the other process $E_j \xrightarrow{a} E_{j+1}$.

Player II knows which transition player I chose, and more generally both players know all the previous moves. The play then may continue with further moves.

The next move in a game play is therefore very straightforward. The important issue is when a player is said to *win* a play of a game. A game is played until one of the players wins, where the winning circumstances are described in figure 13. If a player is unable to make a move then the other player wins that play of the game. Player II loses when condition $1'$ holds, when no corresponding transition is available in response to a move from player I, which happens in the

[10] For instance if $E \xrightarrow{a} E_1$ and $E \xrightarrow{a} E_2$ then the observer is able to choose either of these, but there is not a "testing" process $\overline{a}.F$ which can guarantee this choice in the context $(\overline{a}.F \mid E)\backslash\{a\}$: the two results $(F \mid E_1)\backslash\{a\}$ and $(F \mid E_2)\backslash\{a\}$ are equally likely after the synchronization on a.

Player II wins	**Player I wins**
1. The play is $(E_0, F_0) \dots (E_n, F_n)$ and there are no available transitions from E_n or from F_n.	1′. The play is $(E_0, F_0) \dots (E_n, F_n)$ and for some a either $E_n \xrightarrow{a} E'$ and not$(\exists F'.\, F_n \xrightarrow{a} F')$ or $F_n \xrightarrow{a} F'$ and not$(\exists E'.\, E_n \xrightarrow{a} E')$.
2. The play is $(E_0, F_0) \dots (E_n, F_n)$ and for some $i < n$ $E_i = E_n$ and $F_i = F_n$.	
3. The play has infinite length.	

Fig. 13. Winning conditions

configuration (E_n, F_n) when one of these processes is able to perform an initial action which the other can not, and so a manifest difference is detectable. Player I loses in the configuration (E_n, F_n) when both these processes have terminated or are deadlocked as described in 1. Player II also wins in the other two circumstances, first when there is a repeat configuration, when (E_n, F_n) has already occurred earlier in the play, as in 2, and second when the play has infinite length. In both these cases player I has been unable to expose a difference between the initial processes. Condition 2 is not necessary, as it is subsumed by 3: however we include it because then an infinite length play is only possible when at least one of the initial processes is infinite state.

Example 1 Each play of the game (Cl, Cl_2), when $Cl \overset{\text{def}}{=} \texttt{tick}.Cl$ and $Cl_2 \overset{\text{def}}{=} \texttt{tick}.\texttt{tick}.Cl_2$, has the form $(Cl, Cl_2)\,(Cl, \texttt{tick}.Cl_2)\,(Cl, Cl_2)$ which ensures that player I loses because of the repeat configuration. It does not matter whether player I chooses the $\langle\ \rangle$ or the $[\ ]$ move, as player II is guaranteed to win. In the case of the game (Cl, Cl_5) when $Cl_5 \overset{\text{def}}{=} \texttt{tick}.Cl_5 + \texttt{tick}.\mathbf{0}$ there are plays that player I wins and plays that player II wins. If player I initially moves $Cl_5 \xrightarrow{\texttt{tick}} \mathbf{0}$ then after her opponent makes the move $Cl \xrightarrow{\texttt{tick}} Cl$, the resulting configuration $(Cl, \mathbf{0})$ obeys condition 1′ of figure 13, and so player I wins. Instead if player I chooses the other $[\ ]$ move $Cl_5 \xrightarrow{\texttt{tick}} Cl_5$ then player II wins immediately with the transition $Cl \xrightarrow{\texttt{tick}} Cl$. However player I has the power to win any play of (Cl, Cl_5) by initially choosing the transition $Cl_5 \xrightarrow{\texttt{tick}} \mathbf{0}$. Similarly player II is able to win any play of (Cl_5, Cl_5) just by copying any move that player I makes in the other process. $\square$

A strategy for a player is a set of rules which tells her how to move depending on what has happened previously in the play. A player uses the strategy π in a play if all her moves in the play obey the rules in π. The strategy π is a *winning strategy* if the player wins every play in which she uses π. In example 1 above, player I's winning strategy for the game (Cl, Cl_5) consists of the single rule: if the current game configuration is (Cl, Cl_5) choose the transition $Cl_5 \xrightarrow{\texttt{tick}} \mathbf{0}$. In the example game (Cl_5, Cl_5), player II's winning strategy contains the two rules: if the current game configuration is (Cl_5, Cl_5) and player I has chosen $Cl_5 \xrightarrow{\texttt{tick}} Cl_5$ then choose $Cl_5 \xrightarrow{\texttt{tick}} Cl_5$, and if it is (Cl_5, Cl_5) and player I has chosen $Cl_5 \xrightarrow{\texttt{tick}} \mathbf{0}$ then choose $Cl_5 \xrightarrow{\texttt{tick}} \mathbf{0}$. It turns out that for any pair of processes one of the players has a winning strategy, and that this strategy is history free in the sense that the rules do not need to appeal to moves that occurred before the current game configuration.

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). □

Example 3 Player II has a winning strategy for the game (B, B') where these processes are, $B \stackrel{\text{def}}{=} \texttt{in}.B \mid \texttt{out}.B$ and $B' \stackrel{\text{def}}{=} \texttt{in}.B' + \texttt{out}.B'$. 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 by player I. □

Example 4 Player II has a winning strategy for $((C \mid U)\backslash\{\texttt{in}, \texttt{ok}\}, Ucop')$ when these processes are:

$$\begin{aligned}
C &\stackrel{\text{def}}{=} \texttt{in}(x).\overline{\texttt{out}}(x).\overline{\texttt{ok}}.C \\
U &\stackrel{\text{def}}{=} \texttt{write}(x).\overline{\texttt{in}}(x).\texttt{ok}.U \\
Ucop' &\stackrel{\text{def}}{=} \texttt{write}(x).\tau.\overline{\texttt{out}}(x).\tau.Ucop'
\end{aligned}$$

A play has to proceed as follows, modulo the choice of data value:

$$\begin{array}{l}
((C \mid U)\backslash\{\texttt{in}, \texttt{ok}\}, Ucop') \\
((C \mid \overline{\texttt{in}}(v).\texttt{ok}.U)\backslash\{\texttt{in}, \texttt{ok}\}, \tau.\overline{\texttt{out}}(v).\tau.Ucop') \\
((\overline{\texttt{out}}(v).\overline{\texttt{ok}}.C \mid \texttt{ok}.U)\backslash\{\texttt{in}, \texttt{ok}\}, \overline{\texttt{out}}(v).\tau.Ucop') \\
((\overline{\texttt{ok}}.C \mid \texttt{ok}.U)\backslash\{\texttt{in}, \texttt{ok}\}, \tau.Ucop') \\
((C \mid U)\backslash\{\texttt{in}, \texttt{ok}\}, Ucop').
\end{array}$$

An important feature is that each play has a finite bounded length even though the processes are infinite state. □

When player II has a winning strategy for the game (E, F) we say that process E is *game equivalent* to process F. In this circumstance player II is able to win any play irrespective of the moves her opponent makes, and so player I is unable to distinguish between E and F.

Player II can always match player I's moves when two processes E and F are game equivalent: by the $\langle\ \rangle$ move if $E \xrightarrow{a} E'$ then there is a corresponding transition $F \xrightarrow{a} F'$ and E' and F' are also game equivalent, and by the $[\ ]$ move if $F \xrightarrow{a} F'$ then there is also a corresponding transition $E \xrightarrow{a} E'$ with E' and F' game equivalent. This is precisely the criterion for being a *bisimulation* relation. Bisimulations were introduced[11] by Park [57] as a small refinement of the equivalence defined by Hennessy and Milner in [38, 39, 50].

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

1. if $E \xrightarrow{a} E'$ then $F \xrightarrow{a} F'$ for some F' such that $(E', F') \in \mathcal{R}$, and
2. if $F \xrightarrow{a} F'$ then $E \xrightarrow{a} E'$ for some E' such that $(E', F') \in \mathcal{R}$.

A binary relation between processes counts as a bisimulation provided that it obeys the two hereditary conditions in this definition. Simple examples of bisimulations are the identity relation and the empty relation.

Two processes E and F are *bisimulation equivalent* (or *bisimilar*), written $E \sim F$, if there is a bisimulation relation $\mathcal{R}$ with $(E, F) \in \mathcal{R}$.

Proposition 1 *The relation $\sim$ between processes is an equivalence relation.*

Example 5 A classical example of two processes that are not bisimilar is $a.(b.\mathbf{0} + c.\mathbf{0})$ and $a.b.\mathbf{0} + a.c.\mathbf{0}$. There cannot be a bisimulation relating this pair because it would have to include either $(b.\mathbf{0} + c.\mathbf{0}, b.\mathbf{0})$ or $(b.\mathbf{0} + c.\mathbf{0}, c.\mathbf{0})$. $\square$

Proposition 2 *If $\{\mathcal{R}_i : i \in I\}$ is a family of bisimulation relations then $\bigcup\{\mathcal{R}_i : i \in I\}$ is a bisimulation relation.*

A corollary of this last Proposition is that the binary relation $\sim$ is the largest bisimulation, as $\sim$ is $\bigcup\{\mathcal{R} : \mathcal{R}$ is a bisimulation$\}$.

Not surprisingly bisimulation and game equivalence coincide.

Proposition 3 *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 (see the discussion in section 1.2).

To show that two processes are bisimilar it is sufficient to exhibit a bisimulation relation which contains them. This offers a very straightforward proof technique for bisimilarity.

Example 6 The two processes Cnt and Ct'_0 are bisimilar where

$$\begin{aligned} Cnt &\overset{\text{def}}{=} \mathtt{up}.(Cnt \mid \mathtt{down}.\mathbf{0}) \\ Ct'_0 &\overset{\text{def}}{=} \mathtt{up}.Ct'_1 \\ Ct'_{i+1} &\overset{\text{def}}{=} \mathtt{up}.Ct'_{i+2} + \mathtt{down}.Ct'_i \end{aligned}$$

A bisimulation containing the pair (Cnt, Ct'_0) has to be infinite because these processes are infinite state. Let $\mathcal{C}_i$ be the following families of processes for $i \geq 0$ (when brackets are dropped between parallel components):

[11] They also occur in a slightly different form in the theory of modal logic as zig-zag relations, see [12].

$$\begin{aligned}\mathcal{C}_0 &= \{Cnt \mid \mathbf{0}^j \ : \ j \geq 0\}\\ \mathcal{C}_{i+1} &= \{E \mid \mathbf{0}^j \mid \mathtt{down}.\mathbf{0} \mid \mathbf{0}^k \ : \ E \in \mathcal{C}_i \text{ and } j \geq 0 \text{ and } k \geq 0\}\end{aligned}$$

where $F \mid \mathbf{0}^0 = F$ and $F \mid \mathbf{0}^{i+1} = F \mid \mathbf{0}^i \mid \mathbf{0}$. The relation $\mathcal{R} = \{(E, Ct'_i) \ : \ i \geq 0 \text{ and } E \in \mathcal{C}_i\}$ is a bisimulation which contains (Cnt, Ct'_0). □

Bisimulation equivalence is also a congruence with respect to all the process combinators introduced in previous sections (including the operator *Rep*).

Proposition 4 *If $E \sim F$ then for any process G, set of actions K, action a, and renaming function f,*

1. $a.E \sim a.F$	2. $E + G \sim F + G$	3. $E \mid G \sim F \mid G$
4. $E[f] \sim F[f]$	5. $E\backslash K \sim F\backslash K$	6. $E\backslash\backslash K \sim F\backslash\backslash K$
7. $E\|_K G \sim F\|_K G$	8. $rep(E) \sim rep(F)$.	

Bisimulation equivalence is a very fine equivalence between processes, reflecting the fact that in the presence of concurrency a more intensional description of process behaviour is needed than for instance its set of traces. For full CCS the question whether two processes are bisimilar is undecidable. As was noted in section 1.5 Turing machines can be "coded" in CCS. Let TM_n be this coding of the n-th Turing machine when all observable actions are hidden (using the operator $\backslash\backslash$ which can be defined in CCS). The undecidable Turing machine halting problem is equivalent to whether or not $TM_n \sim Div$ where $Div \overset{\text{def}}{=} \tau.Div$. However an interesting question is for which subclasses of processes it is decidable. Clearly this is the case for finite state processes, as there are only finitely many candidates for being a bisimulation. Surprisingly it is also decidable for families of infinite state processes including context-free processes and basic parallel processes [23, 22]. By exploiting the observation in example 4, when the length of a game play is boundedly finite, one can also show decidability of bisimilarity for various classes of value passing processes whose data may be drawn from an infinite value space.

2.5 Modal properties and equivalences

Another approach to understanding equivalences between processes is in terms of families of modal properties. Each finite or infinite set of modal formulas Γ induces an equivalence relation $\equiv_\Gamma$ on processes[12]:

$$E \equiv_\Gamma F \ \text{ iff } \ \forall \Phi \in \Gamma.\ E \models \Phi \ \text{ iff } \ F \models \Phi$$

Example 1 When Γ consists of all formulas of the form $\langle a_1\rangle \ldots \langle a_n\rangle\mathtt{tt}$ the relation $\equiv_\Gamma$ is traces equivalence. If Γ contains the formulas $\langle a_1\rangle \ldots \langle a_n\rangle[-]\mathtt{ff}$ then the induced equivalence is completed traces, and when generalized to formulas

[12] Similarly we can also define the preorder induced by Γ: $E \sqsubseteq_\Gamma F$ iff $\forall \Phi \in \Gamma$ if $E \models \Phi$ then $F \models \Phi$.

$\langle a_1 \rangle \dots \langle a_n \rangle [K]\mathtt{ff}$ it is failures equivalence. Observable equivalences include observable traces and observable failures equivalence. $\Box$

There is an intimate relationship between bisimulation equivalence and having the same modal properties. Let Γ be the family of all modal formulas built from the boolean connectives and the modal operators $\{[K], \langle K \rangle, [\![-]\!], \langle\!\langle - \rangle\!\rangle, [\![\downarrow]\!]\}$. Bisimulation equivalence preserves modal properties.

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

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 modal formulas, and Proposition 2 holds for the subset of modal formulas when the modalities are restricted to be either $[a]$ or $\langle a \rangle$, $a \in \mathcal{A}$. Under this restriction, these two results are known as the modal *characterization* of bisimulation equivalence, due to Hennessy and Milner [38, 39]. This characterization not only reinforces the naturalness of bisimulation equivalence but also suggests that modal logic is well-suited for describing properties of processes.

Example 2 Consider the family of clocks $Cl^{i+1} \overset{\text{def}}{=} \mathtt{tick}.Cl^i$ for $i \geq 0$ and the clock $Cl \overset{\text{def}}{=} \mathtt{tick}.Cl$. Let E be the process $\sum\{Cl^i \,:\, i \geq 0\}$, and let F be $E + Cl$. The processes E and F are not bisimilar because the transition $F \xrightarrow{\mathtt{tick}} Cl$ would have to be matched by $E \xrightarrow{\mathtt{tick}} Cl^j$ for some $j \geq 0$, and clearly $Cl \not\sim Cl^j$. On the other hand E and F satisfy the same modal properties. $\Box$

There is an unrestricted characterization result for infinitary modal logic, M_∞, given as follows where I ranges over arbitrary finite and infinite indexing families:

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

The satisfaction relation between processes and $\bigwedge$ and $\bigvee$ formulas is defined as expected

$$\begin{array}{l} E \models \bigwedge\{\Phi_i \,:\, i \in I\} \text{ iff } E \models \Phi_j \text{ for every } j \in I \\ E \models \bigvee\{\Phi_i \,:\, i \in I\} \text{ iff } E \models \Phi_j \text{ for some } j \in I \end{array}$$

The atomic formula $\mathtt{tt}$ is defined as $\bigwedge\{\Phi_i \,:\, i \in \emptyset\}$ and $\mathtt{ff}$ as $\bigvee\{\Phi_i \,:\, i \in \emptyset\}$.

Proposition 3 *$E \sim F$ iff $E \equiv_{M_\infty} F$.*

A variety of the proposed equivalences between image finite processes in the linear and branching time spectrum, as summarized in [32], can be presented in terms of games and in terms of $\equiv_\Gamma$ when Γ is an appropriate subset of modal formulas.

2.6 Observable bisimulations

In the previous two sections we have seen that there is a close relationship between winning strategies, bisimulation relations, and modal properties. Not one of this trio abstracts from the silent action τ, as each appeals to the family of transition relations $\{\xrightarrow{a} : a \in \mathcal{A}\}$. By consistently replacing this set with the family of *observable* transitions as defined in section 1.3, these notions uniformly abstract from τ. We can define *observable* modal logic whose modalities are restricted to the set $\{[\![K]\!], [\![\;]\!], \langle\!\langle K\rangle\!\rangle, \langle\!\langle\;\rangle\!\rangle\}$. We can also define *observable* games whose moves appeal to the thicker transition relations $\overset{a}{\Longrightarrow}$, $a \in \mathcal{O} \cup \{\varepsilon\}$, and *observable* bisimulation relations.

A play of the observable game (E_0, F_0) is again a finite or infinite length sequence of pairs $(E_0, F_0) \ldots (E_i, F_i) \ldots$ played by the two observers players I and II. The next pair after an initial part of the play $(E_0, F_0) \ldots (E_j, F_j)$ is determined by one of the moves where $a \in \mathcal{O} \cup \{\varepsilon\}$:

- $\langle\!\langle\;\rangle\!\rangle$: Player I chooses a transition $E_j \overset{a}{\Longrightarrow} E_{j+1}$ from E_j and then player II chooses a transition with the same label $F_j \overset{a}{\Longrightarrow} F_{j+1}$ from F_j.
- $[\![\;]\!]$: Player I chooses a transition $F_j \overset{a}{\Longrightarrow} F_{j+1}$ from F_j and then player II chooses a transition with the same label $E_j \overset{a}{\Longrightarrow} E_{j+1}$ from E_j.

A game is played until one of the players wins, where the winning circumstances are depicted in figure 14. The only difference with section 2.4 is that player I is

Player II wins	**Player I wins**
1. The play is $(E_0, F_0) \ldots (E_n, F_n)$ and for some $i < n$, $E_i = E_n$ and $F_i = F_n$.	1′. The play is $(E_0, F_0) \ldots (E_n, F_n)$ and for some a either $E_n \overset{a}{\Longrightarrow} E'$ and not$(\exists F'.\, F_n \overset{a}{\Longrightarrow} F')$ or $F_n \overset{a}{\Longrightarrow} F'$ and not$(\exists E'.\, E_n \overset{a}{\Longrightarrow} E')$.
2. The play has infinite length.	

Fig. 14. Winning conditions

always able to make a move (because of the empty transition).

Again a strategy for a player is a set of rules which tells her how to move, and it is winning if the player wins every play in which she uses it. E and F are *observably game equivalent* if player II has a winning strategy for (E, F).

Underpinning observable game equivalence is the existence of an observable bisimulation relation whose definition is as in section 2.4 except with respect to observable transitions $\overset{a}{\Longrightarrow}$.

Definition 1 A binary relation $\mathcal{R}$ between processes is an *observable bisimulation* just in case whenever $(E, F) \in \mathcal{R}$ and $a \in \mathcal{O} \cup \{\varepsilon\}$,

1. if $E \overset{a}{\Longrightarrow} E'$ then $F \overset{a}{\Longrightarrow} F'$ for some F' such that $(E', F') \in \mathcal{R}$, and
2. if $F \overset{a}{\Longrightarrow} F'$ then $E \overset{a}{\Longrightarrow} E'$ for some E' such that $(E', F') \in \mathcal{R}$.

E and F are observably bisimilar, written $E \approx F$, if there is an observable bisimulation relation[13] $\mathcal{R}$ with $(E, F) \in \mathcal{R}$. The relation $\approx$ has many properties in common with $\sim$. It is an equivalence relation, and the union of a family of observable bisimulations is also an observable bisimulation, and so $\approx$ is itself an observable bisimulation. Observable bisimulation and observable game equivalence also coincide.

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

A direct proof that two processes are observably bisimilar consists in exhibiting an observable bisimulation relation containing them.

Example 1 We show that *Protocol* $\approx$ *Cop* by exhibiting an observable bisimulation which contains them. (Again we drop brackets when processes contain multiple parallel components as $|$ is associative and commutative with respect to this equivalence.) The following relation

$$\begin{array}{l}
\{(\mathit{Protocol}, \mathit{Cop})\} \cup \\
\{((\mathit{Send1}(m) \mid \mathit{Medium} \mid \overline{\mathtt{ok}}.\mathit{Receiver})\backslash J,\ \mathit{Cop}) \ :\ m \in D\} \cup \\
\{((\overline{\mathtt{sm}}(m).\mathit{Send1}(m) \mid \mathit{Medium} \mid \mathit{Receiver})\backslash J,\ \overline{\mathtt{out}}(m).\mathit{Cop}) \ :\ m \in D\} \cup \\
\{((\mathit{Send1}(m) \mid \mathit{Med1}(m) \mid \mathit{Receiver})\backslash J,\ \overline{\mathtt{out}}(m).\mathit{Cop}) \ :\ m \in D\} \cup \\
\{((\mathit{Send1}(m) \mid \mathit{Medium} \mid \overline{\mathtt{out}}(m).\overline{\mathtt{ok}}.\mathit{Receiver})\backslash J,\ \overline{\mathtt{out}}(m).\mathit{Cop}) \ :\ m \in D\} \cup \\
\{((\mathit{Send1}(m) \mid \overline{\mathtt{ms}}.\mathit{Medium} \mid \mathit{Receiver})\backslash J,\ \overline{\mathtt{out}}(m).\mathit{Cop}) \ :\ m \in D\}
\end{array}$$

is an observable bisimulation. □

There is also an intimate relationship between observable bisimulation equivalence and observable modal logic, Γ.

Proposition 2 *If $E \approx F$ then $E \equiv_\Gamma F$.*

Notice that this result is *not* true if we include the modalities $[K]$ and $\langle K \rangle$, or the divergence sensitive modality $[\![\downarrow]\!]$. The converse of Proposition 1 holds for observably image finite processes. A process E is immediately image finite if for each $a \in \mathcal{O} \cup \{\varepsilon\}$ the set $\{F \ :\ E \overset{\varepsilon}{\Longrightarrow} F\}$ is finite, and E is observably image finite if each member of the set $\{F \ :\ \exists w \in (\mathcal{O} \cup \{\varepsilon\})^*\}$ is immediately image finite.

Proposition 3 *If E and F are observably image finite and $E \equiv_\Gamma F$ then $E \approx F$.*

[13] We have slightly departed from standard terminology, where $\approx$ is called *weak* bisimilarity, and Definition 1 above defines what is usually called a weak bisimulation relation.

So far there is a very smooth transition from previous results concerning the transitions $\xrightarrow{a}$ to observable transitions $\overset{a}{\Longrightarrow}$. This does break down at an important point. Observable bisimilarity is not a congruence for the $+$ operator because of the initial preemptive power of τ. The two processes E and $\tau.E$ are observably bisimilar but for many instances of F the processes $E+F$ and $\tau.E+F$ are not. In CCS [39, 52] an important equivalence is the largest subset of $\approx$ which is also a congruence. As observable bisimilarity is a congruence for all the other operators[14] of CCS, this congruence, denoted by $\approx^c$, can also be described using transitions. For it is only that initial preemptive τ transition that causes failure.

Definition 2 $E \approx^c F$ iff

1. $E \approx F$
2. if $E \xrightarrow{\tau} E'$ then $F \xrightarrow{\tau} F_1 \overset{\varepsilon}{\Longrightarrow} F'$ and $E' \approx F'$ for some F_1 and F', and
3. if $F \xrightarrow{\tau} F'$ then $E \xrightarrow{\tau} E_1 \overset{\varepsilon}{\Longrightarrow} E'$ and $E' \approx F'$ for some E_1 and E'.

When E and F are stable (as is the case with *Protocol* and *Cop* of example 1) $E \approx F$ implies $E \approx^c F$.

There is a finer observable bisimulation equivalence called branching bisimulation equivalence which also has a logical characterization [27]: we consider it briefly in section 3.2. Observable bisimilarity and its congruence are not sensitive to divergence. So they do not preserve the strong necessity properties discussed at the end of section 2.2. However it is possible to define equivalences that take divergence into account [36, 40, 71].

2.7 Equivalence checking

A direct proof that two processes are (observably) bisimilar is to exhibit the appropriate bisimulation relation which contains them. Example 7 of section 2.4 and example 1 of section 2.6 exemplify this proof technique. In the case that processes are finite state this can be done automatically. There is a variety of tools which include this capability including the Edinburgh Concurrency Workbench [25] which exploits efficient algorithms for checking bisimilarity between finite state processes, as developed in [41].

Alternatively equivalence proofs can utilize conditional equational reasoning. There is an assortment of algebraic, and semi-algebraic, theories of processes depending on the equivalence and the process combinators: for details see [36, 40, 6, 52]. It is essential that the equivalence is a congruence. To give a flavour of equational reasoning we present a proof in the equational theory for CCS that a simplified slot machine without data values is equivalent to a streamlined process description. The congruence is $\approx^c$ which was defined in the previous section.

The following are important CCS laws which are used in the proof:

[14] More generally only case 2 of Proposition 4 of section 2.4 fails for $\approx$.

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

The last four are clear from the behavioural meanings of the operators. The first two are τ-laws and show that we are dealing with an observable equivalence. There is also appeal to a rule schema, called an *expansion law* by Milner [52], relating concurrency and choice:

$$\begin{array}{l}
\text{if } x_i = \sum\{a_{ij}.x_{ij} \; : \; 1 \leq j \leq n_i\} \;\; \text{for } i : 1 \leq i \leq m \text{ then } x_1 \mid \ldots \mid x_m \;\; = \\
\sum\{a_{ij}.(x_1 \mid \ldots \mid x_{i-1} \mid x_{ij} \mid x_{i+1} \mid \ldots \mid x_m) \; : \; 1 \leq i \leq m \text{ and } 1 \leq j \leq n_i\} \; + \\
\sum\{\tau.(x_1 \mid \ldots \mid x_{k-1} \mid x_{kl} \mid x_{k+1} \mid \ldots \mid x_{i-1} \mid x_{ij} \mid x_{i+1} \mid \ldots \mid x_m) \; : \\
\qquad\qquad\qquad 1 \leq k < i \leq m \text{ and } a_{kl} = \overline{a}_{ij}\}
\end{array}$$

Proof rules for recursion are also needed. In the case that E does not contain any occurrence of $|$, we say that P is *guarded* in E if all its occurrences in E are within the scope of a prefix a. operator when a is an observable action. This condition guarantees that the equation $P = E$, when the only process constant E contains is P, has a unique solution up to $\approx^c$: that is, if $F \approx^c E\{F/P\}$ and $G \approx^c E\{G/P\}$ then $F \approx^c G$. This justifies the following two conditional rules for recursion:

- if $P \overset{\text{def}}{=} E$ then $P = E$.
- if $P = E$ and P is guarded in E, and $Q = F$ and Q is guarded in F, and $E\{Q/P\} = F$, then $P = F$.

The slot machine SM without data values, and its succinct description SM' appear in figure 15. We prove that $SM = SM'$. The idea of the proof is to first simplify SM by showing that it is equal to an expression which does not contain the parallel operator. The proof proceeds on SM using the expansion law and the laws earlier for $\backslash K$, $\mathbf{0}$ and τ (and the first recursion rule):

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

Let $SM_1 \equiv (IO_1 \mid B \mid D)\backslash K$. By similar reasoning to the above we obtain:

$$SM_1 = \tau.\tau.(IO_2 \mid \texttt{left}.B \mid D_1)\backslash K$$

Assume $SM_2 \equiv (IO_2 \mid \texttt{left}.B \mid D_1)\backslash K$. Similarly, $SM_2 = \tau.SM_3 + \tau.SM_4$:

$$\begin{aligned}
&IO \overset{\text{def}}{=} \texttt{slot}.IO_1 \qquad IO_1 \overset{\text{def}}{=} \overline{\texttt{bank}}.IO_2\\
&IO_2 \overset{\text{def}}{=} \texttt{lost}.\overline{\texttt{loss}}.IO + \overline{\texttt{release}}.\overline{\texttt{win}}.IO\\
&B \overset{\text{def}}{=} \texttt{bank}.B_1 \qquad B_1 \overset{\text{def}}{=} \overline{\texttt{max}}.\texttt{left}.B\\
&D \overset{\text{def}}{=} \texttt{max}.D_1 \qquad D_1 \overset{\text{def}}{=} \overline{\texttt{lost}}.\overline{\texttt{left}}.D + \overline{\texttt{release}}.\overline{\texttt{left}}.D\\
&SM \equiv (IO \mid B \mid D)\backslash K \;\text{ where } K = \{\texttt{bank}, \texttt{max}, \texttt{left}, \texttt{release}\}\\
&SM' \overset{\text{def}}{=} \texttt{slot}.(\tau.\overline{\texttt{loss}}.SM' + \tau.\overline{\texttt{win}}.SM')
\end{aligned}$$

Fig. 15. A simplified slot machine

$$\begin{aligned}
SM_3 &\equiv (\overline{\texttt{loss}}.IO \mid \texttt{left}.B \mid \overline{\texttt{left}}.D)\backslash K\\
SM_4 &\equiv (\overline{\texttt{win}}.IO \mid \texttt{left}.B \mid \overline{\texttt{left}}.D)\backslash K
\end{aligned}$$

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

$$\begin{aligned}
SM_3 &= (\overline{\texttt{loss}}.IO \mid \texttt{left}.B \mid \overline{\texttt{left}}.D)\backslash K\\
&= \overline{\texttt{loss}}(IO \mid \texttt{left}.B \mid \overline{\texttt{left}}.D)\backslash K + \tau.\overline{(\overline{\texttt{loss}}.IO \mid B \mid D)}\\
&= \overline{\texttt{loss}}.(\tau.SM + \texttt{slot}.(IO_1 \mid \texttt{left}.B \mid \overline{\texttt{left}}.D)\backslash K) + \tau.\overline{\texttt{loss}}.SM\\
&= \overline{\texttt{loss}}.(\tau.SM + \texttt{slot}.\tau.(IO_1 \mid B \mid D)\backslash K) + \tau.\overline{\texttt{loss}}.SM\\
&= \overline{\texttt{loss}}.(\tau.SM + \texttt{slot}.(IO_1 \mid B \mid D)\backslash K) + \tau.\overline{\texttt{loss}}.SM\\
&= \overline{\texttt{loss}}.(\tau.SM + SM) + \tau.\overline{\texttt{loss}}.SM\\
&= \overline{\texttt{loss}}.\tau.SM + \tau.\overline{\texttt{loss}}.SM\\
&= \tau.\overline{\texttt{loss}}.SM
\end{aligned}$$

By similar reasoning $SM_4 = \tau.\overline{\texttt{win}}.SM$. Backtracking and substituting equals for equals, and then applying τ laws:

$$\begin{aligned}
SM &= \texttt{slot}.SM_1\\
&= \texttt{slot}.\tau.\tau.SM_2\\
&= \texttt{slot}.\tau.\tau.(\tau.SM_3 + \tau.SM_4)\\
&= \texttt{slot}.\tau.\tau.(\tau.\tau.\overline{\texttt{loss}}.SM + \tau.\tau.\overline{\texttt{win}}.SM)\\
&= \texttt{slot}.(\tau.\overline{\texttt{loss}}.SM + \tau.\overline{\texttt{win}}.SM)
\end{aligned}$$

We have now shown that $SM = E$ where E does not contain the parallel operator, and where SM is guarded in E. The expression E is very close to the definition of SM'. Clearly $E\{SM'/SM\} = \texttt{slot}.(\tau.\overline{\texttt{loss}}.SM' + \tau.\overline{\texttt{win}}.SM')$, and so by the second recursion rule $SM = SM'$ which completes the proof.

3 Temporal Properties

Modal logic as introduced in sections 2.1 and 2.2 is able to express local capabilities and necessities of processes such as that `tick` is a possible next (observable) action or that it must happen next. However it cannot express enduring capabilities (such as that `tick` is always possible) or long term inevitabilities (such as that `tick` must eventually happen). These features, especially in the guise of safety or liveness properties, have been found to be very useful when analysing the behaviour of concurrent systems. Another abstraction from behaviour is a run of a process which is a finite or infinite length sequence of transitions. Runs provide a basis for understanding longer term capabilities. Logics where properties are primarily ascribed to runs of systems are called temporal logics. An alternative foundation for temporal logic is to view these enduring features as extremal solutions to recursive modal equations.

3.1 Modal properties revisited

A property separates a set of processes into two disjoint subsets, those with the property and those without it. 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 $Cl_1 \overset{\text{def}}{=} \texttt{tick}.\texttt{tock}.Cl_1$. We let $\| \Phi \|^{\mathcal{E}}$ be the subset of the family of processes $\mathcal{E}$ having the modal property Φ, the set $\{E \in \mathcal{E} : E \models \Phi\}$. Consequently any modal formula Φ partitions $\mathcal{E}$ into $\| \Phi \|^{\mathcal{E}}$ 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 below. First the boolean connectives:

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

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

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

The operator $\| \# \|^{\mathcal{E}}$ is the semantic analogue of $\#$ in the same way that $\cap$ is the interpretation of $\wedge$. In the cases of $[K]$ and $\langle K\rangle$ these transformers are:

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

Of course we would like the direct inductive definition of $\| \Phi \|^{\mathcal{E}}$ to coincide with its definition as $\{E \in \mathcal{E} : E \models \Phi\}$. But this is only guaranteed when $\mathcal{E}$

is a *transition closed* set, which obeys the condition if $E \in \mathcal{E}$ 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 $\overset{a}{\Longrightarrow}$ and $\overset{\varepsilon}{\Longrightarrow}$. In the following result it is assumed that $\|\Phi\|^{\mathcal{P}}$ is determined by its inductive definition.

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

The following features 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 $Cl \overset{\text{def}}{=} \texttt{tick}.Cl$ and $Cl^{i+1} \overset{\text{def}}{=} \texttt{tick}.Cl^i$ for every $i \geq 0$. What distinguishes Cl from the rest of $\mathcal{P}$ is its long term capability for ticking endlessly. Each Cl^i ticks exactly 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[15] of the modal logics in sections 2.1 and 2.2.

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 cannot capture more global or long term features of processes. We can contrast the local 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, which throws light on this contrast between immediate and long term, is that of a *run* of a process E_0 which is a finite or infinite length sequence of transitions of the form $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots$. When a run has finite length its final process is then unable to perform a transition. So a run from E_0 can be viewed as a computation from E_0, a maximal performance of actions.

Game or bisimulation equivalence, as defined in section 2.4, "preserves" runs as stated by the next Proposition.

Proposition 1 *Suppose that $E_0 \sim F_0$,*

[15] 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.

1. *if* $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} E_n$ *is a finite length run from* E_0 *then there is a run* $F_0 \xrightarrow{a_1} F_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} F_n$ *from* F_0 *such that* $E_i \sim F_i$ *for all* $i : 0 \leq i \leq n$, *and*
2. *if* $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots$ *is an infinite length run from* E_0 *then there is an infinite length run* $F_0 \xrightarrow{a_1} F_1 \xrightarrow{a_2} \ldots$ *from* F_0 *such that* $E_i \sim F_i$ *for all* i.

Because $E_0 \sim F_0$ implies $F_0 \sim E_0$, each run from F_0 also has to be matched with a run from E_0. A simple consequence is that the clock Cl is not bisimilar to any clock Cl^i because Cl has a run which cannot be emulated by Cl^i.

Clearly observable bisimulation equivalence, $\approx$, from section 2.6 does not preserve runs in the sense of Proposition 1. A simple example is that the infinite run $Div \xrightarrow{\tau} Div \xrightarrow{\tau} \ldots$ has no correlate from $\tau.\mathbf{0}$ although $Div \approx \tau.\mathbf{0}$ when $Div \stackrel{\rm def}{=} \tau.Div$. We may try and weaken the matching requirement. For any run from E_0 there is a corresponding run from F_0 such that there is a finite or infinite partition across these runs containing equivalent processes. However this induces a finer equivalence than observable bisimulation, called branching bisimulation [33]. It should also be compared with "stuttering" equivalence as proposed within [19].

Observable bisimilarity does preserve *observable* runs whose transitions are given by the thicker arrows $\stackrel{a}{\Longrightarrow}$ and $\stackrel{\varepsilon}{\Longrightarrow}$. But there is a drawback because of $\stackrel{\varepsilon}{\Longrightarrow}$ transitions. The process $Cl \stackrel{\rm def}{=} \texttt{tick}.Cl$ has the observable inactive run $Cl \stackrel{\varepsilon}{\Longrightarrow} Cl \stackrel{\varepsilon}{\Longrightarrow} \ldots$ which means that it fails to have the observable property that `tick` must eventually happen.

Many significant properties of systems can be understood as features of all their runs. Especially important is a classification of properties into *safety* and *liveness*, originally due to Lamport [43]. A safety property states that *nothing bad ever happens* whereas a liveness property expresses that *something good does eventually happen*. A process has a safety property just in case no run from it contains the bad feature, and it has a liveness property when each of its runs 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 each 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 perform both $\overline{\texttt{tcross}}$ and $\overline{\texttt{ccross}}$ as next actions, and so the bad feature is $\langle\overline{\texttt{tcross}}\rangle\texttt{tt} \land \langle\overline{\texttt{ccross}}\rangle\texttt{tt}$. □

Liveness and safety properties of a process concern all its runs. 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 eventually happen, that it eventually happens in some run.

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

There is considerable middle ground between all and some runs. Often we are only interested in liveness and safety in the case of a subset of runs 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 $\overline{\texttt{ccross}}$. $\square$

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 retransmitted. However we may make the assumption that the medium must eventually pass to the receiver a repeatedly retransmitted message, and that therefore these deficient runs are thereby precluded.

Other properties are important such as cyclic properties. The clock Cl_1 earlier performs `tick` immediately followed by `tock` cyclically starting with `tick`.

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_j) may be interspersed before and after each a_i. $\square$

Modal logic expresses properties of processes as their behaviour unfolds through transitions. Temporal logic, on the other hand, ascribes properties to processes by expressing features of some or all of their runs. (For surveys of temporal logic see [31, 49, 64].) In fact there is not a clear demarcation because modal operators can also be viewed as temporal operators, which express "next":

$$E_0 \models [K]\Phi \text{ iff for all } E_0 \text{ runs } E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots, \text{ if } a_1 \in K \text{ then } E_1 \models \Phi.$$
$$E_0 \models \langle K\rangle\Phi \text{ iff for some } E_0 \text{ run } E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots, \; a_1 \in K \text{ and } E_1 \models \Phi.$$

In this work we do not base temporal logic upon the notion of a run. It is of course a very useful abstraction. A run is simply a subset of a transition closed set. Although the properties described in the examples above are not expressible in modal logic of section 2 we shall find appropriate closure conditions on sets of processes which define them, by appealing to *inductive* definitions built out of modal logic. The idea is, for instance, that a long term capability is just a particular closure of an immediate capability.

3.3 Modal equations and fixed points

Definitional equality, $\overset{\text{def}}{=}$, 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 [44]. For instance the modal equation $Z \overset{\text{def}}{=} \langle \texttt{tick} \rangle \texttt{tt}$ stipulates that Z expresses the same property, an ability to perform `tick`, as $\langle \texttt{tick} \rangle \texttt{tt}$. On any transition closed set $\mathcal{P}$ this characteristic is given by the set $\| \langle \texttt{tick} \rangle \texttt{tt} \|^{\mathcal{P}}$.

A more intricate modal equation is $Z \overset{\text{def}}{=} \langle \texttt{tick} \rangle Z$ where both occurrences of Z select the same trait. Unlike the use of $\overset{\text{def}}{=}$ 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 $\mathcal{E} = \| \langle \texttt{tick} \rangle \|^{\mathcal{P}} \mathcal{E}$. Any such solution $\mathcal{E}$ is a *fixed point* of the function $f = \lambda X \subseteq \mathcal{P}.\ \| \langle \texttt{tick} \rangle \|^{\mathcal{P}} X$, 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 $\mathcal{E}$:

$$\begin{array}{ll} \text{PRE} & \text{if } F \in \mathcal{E} \text{ and } E \in \mathcal{P} \text{ and } E \xrightarrow{\texttt{tick}} F \text{ then } E \in \mathcal{E} \\ \text{POST} & \text{if } E \in \mathcal{E} \text{ then } E \xrightarrow{\texttt{tick}} F \text{ for some } F \in \mathcal{E} \end{array}$$

One solution is the empty set as it trivially fulfills both conditions. When $\mathcal{P}$ is $\{Cl\}$, the other subset $\{Cl\}$ also obeys both conditions because of the transition $Cl \xrightarrow{\texttt{tick}} Cl$. In this instance both candidate solutions are successful fixed points, and they can be ordered by subset, $\emptyset \subseteq \{Cl\}$. In the case that $\mathcal{P}$ is generated by the more sonorous clock $Cl_1 \overset{\text{def}}{=} \texttt{tick.tock}.Cl_1$ 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 $Z \overset{\text{def}}{=} \langle \texttt{tick} \rangle Z$ 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 post-fixed points, of all those subsets fulfilling POST. The result applies to arbitrary monotonic functions from subsets of $\mathcal{P}$ to subsets of $\mathcal{P}$. The set $2^{\mathcal{P}}$ is the set of all subsets of $\mathcal{P}$, and the function $g : 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ is monotonic with respect to $\subseteq$ if $\mathcal{E} \subseteq \mathcal{F}$ implies $g(\mathcal{E}) \subseteq g(\mathcal{F})$.

Proposition 1 *If $g : 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ 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 $\bigcup \{\mathcal{E} \subseteq \mathcal{P} : \mathcal{E} \subseteq g(\mathcal{E})\}$.*

Proposition 1 applies to any modal equation $Z \overset{\text{def}}{=} \Psi$ when Ψ is built from modal operators, boolean connectives, the constants `tt` and `ff`, and Z: this follows directly from Proposition 2 of section 3.1, which shows that all these operators induce semantically monotonic functions. Relinquishing the equational format we let $\mu Z.\Psi$ express the property given by the least solution of $Z \stackrel{\rm def}{=} \Psi$, and we let $\nu Z.\Psi$ express the property determined by its largest solution.

For the equation earlier, the least solution $\mu Z.\langle \texttt{tick}\rangle Z$ 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 modal logic. 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.

A more composite equation schema is $Z \stackrel{\rm def}{=} \Phi \vee \langle K\rangle Z$ where Φ does not contain 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 finite or infinite length run of the form $E_0 \xrightarrow{a_0} \ldots \xrightarrow{a_{n-1}} E_n \xrightarrow{a_n} \ldots$ with $E_n \models \Phi$ for some n and where each action a_j, $j < n$, belongs to K: that is, E_0 is able to perform K actions *until* Φ holds. The maximal solution, $\nu Z.\Phi \vee \langle K\rangle Z$, 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 it 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 $Z \stackrel{\rm def}{=} \Phi \wedge \langle K\rangle Z$. 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 $\nu Z.\Phi \wedge (\langle K\rangle Z \vee [K]\texttt{ff})$, 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 $\nu Z.\Phi \wedge (\langle -\rangle Z \vee [-]\texttt{ff})$ 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 $Z \overset{\text{def}}{=} \neg Z$. It fails the monotonicity requirement of Proposition 1. However, if we restrict the form of an equation $Z \overset{\text{def}}{=} \Phi$ 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 (assuming that $Z^c = Z$):

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

It follows that $\| \Phi^c \|^{\mathcal{P}} = \mathcal{P} - \| \Phi \|^{\mathcal{P}}$.

If a property is an extremal solution to the equation $Z \overset{\text{def}}{=} \Phi$ then its complement is the dual solution to $Z \overset{\text{def}}{=} \Phi^c$. Consider convergence which holds of a process when it cannot perform silent actions for ever. The formula $\nu Z.\, \langle \tau \rangle Z$ expresses its complement, divergence, and so convergence is captured by $\mu Z.\, [\tau] Z$. More generally $\mu Z.\, [-] Z$ expresses that all behaviour is finite, that there are no infinite length runs.

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 expressed as $\nu Z.\Phi \wedge [\tau]Z$. 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 the formula $\nu Z.\Phi^c \wedge (\langle K \rangle Z \vee [K]\mathtt{ff})$ 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 replaced by $-$). An example is that the slot machine must eventually output (winnings or an indication of loss). A better description is that whenever a coin is input the slot machine must eventually output, a property expressed using both fixed point operators. However 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 logic, modal mu-calculus, 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 `tt` and `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 $\nu Z.\, \langle \mathtt{tick} \rangle Z \wedge \langle \mathtt{tock} \rangle Z$.

Assume a fixed transition closed set of processes $\mathcal{P}$. We wish to inductively define when the process $E \in \mathcal{P}$ has a temporal property. Semantic clauses for when a process satisfies a fixed point formula $\sigma Z.\Phi$ are needed. However such clauses depend on interpreting subformulas of Φ with possible free occurrences of variables with respect to subfamilies of $\mathcal{P}$. The satisfaction relation, $\models$, is therefore defined indirectly in terms of $\| \Phi \|^{\mathcal{P}}$, 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 $\| \Psi \|^{\mathcal{P}}_{\mathcal{V}}$, 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 $\| [K] \|^{\mathcal{P}}$ and $\| \langle K \rangle \|^{\mathcal{P}}$ 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 $\lambda \mathcal{E} \subseteq \mathcal{P}.\ \| \Phi \|^{\mathcal{P}}_{\mathcal{V}[\mathcal{E}/Z]}$ 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* $\Phi\{\sigma Z.\Phi/Z\}$.

Formulas of the logic without free variables are closed under complement: this follows from the observations in the previous section. In particular $(\nu Z.\,\Phi)^c$ is $\mu Z.\,\Phi^c$ and $(\mu Z.\,\Phi)^c$ is $\nu Z.\,\Phi^c$: for instance $(\nu Z.\,\mu Y.\,\nu X.\,[a](((\langle b\rangle X \wedge Z) \vee [K]Y))^c$ is the formula $\mu Z.\,\nu Y.\,\mu X.\,\langle a\rangle(([b]X \vee Z) \wedge \langle K\rangle Y)$. This is not true for open formulas containing free variables. For example the formula Z does not have an explicit complement. However as we employ valuations we are free to introduce the understanding that a free variable Y has the meaning of the complement of a different free variable Z.

Modal mu-calculus was originally proposed by Kozen [42] (and also see Pratt [60]) but not for its use here[16]. Its roots lie with more general program logics employing extremal fixed points, developed by Park, De Bakker and De Roever, especially when formulated as relational calculi [8, 9, 56]. 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 [44]. Previously Clarke and Emerson used extremal fixed points on top of a temporal logic for expressing properties of concurrent systems [28].

In the case of a closed formula Φ (one without free variables), 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 $\|\,\Phi\,\|^{\mathcal{P}}$ 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 $E \in \|\,\Phi\,\|^{\mathcal{P}}$. 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 are the most difficult to understand. Assume that D and D' are the two processes $D \overset{\text{def}}{=} a.D'$ and $D' \overset{\text{def}}{=} b.\mathbf{0} + a.D$, and that $\mathcal{P}$ is $\{D, D', \mathbf{0}\}$. Let Φ and Ψ be the following similar formulas:

$$\begin{aligned}\Phi &= \nu Z.\,\mu Y.\,[a](((\langle b\rangle\texttt{tt} \wedge Z) \vee Y)\\ \Psi &= \mu Y.\,\nu Z.\,[a](((\langle b\rangle\texttt{tt} \vee Y) \wedge Z)\end{aligned}$$

The formula Φ expresses that b is possible infinitely often throughout any infinite length run consisting wholly of a actions, and so all the processes in $\mathcal{P}$ have this property. The set $\bigcup\{\mathcal{E} \subseteq \mathcal{P} \,:\, \mathcal{E} \subseteq \|\,\mu Y.\,[a](((\langle b\rangle\texttt{tt} \wedge Z) \vee Y)\,\|_{\mathcal{V}[\mathcal{E}/Z]}\}$ is $\mathcal{P}$. To show this we establish that $\mathcal{P} \subseteq \|\,\mu Y.\,[a](((\langle b\rangle\texttt{tt} \wedge Z) \vee Y)\,\|_{\mathcal{V}[\mathcal{P}/Z]}$. This depends on proving that

$$\mathcal{P} \;=\; \bigcap\{\mathcal{F} \subseteq \mathcal{P} \,:\, \|\,[a](((\langle b\rangle\texttt{tt} \wedge Z) \vee Y)\,\|_{(\mathcal{V}[\mathcal{P}/Z])[\mathcal{F}/Y]} \subseteq \mathcal{F}\}$$

Both $\mathbf{0}$ and D belong to $\|\,[a](((\langle b\rangle\texttt{tt} \wedge Z) \vee Y)\,\|_{(\mathcal{V}[\mathcal{P}/Z])[\mathcal{F}/Y]}$ because $\mathbf{0}$ is unable to perform a, and $D \xrightarrow{a} D'$ and $D' \xrightarrow{b} \mathbf{0}$. Therefore $\mathcal{F}$ must also contain D' as $D' \xrightarrow{a} D$.

[16] The modalities here slightly extend those of Kozen's logic as sets of labels may appear within them instead of single labels, and on the other hand Kozen has explicit negation. Kozen calls the logic "propositional mu-calculus" which would be more appropriate for boolean logic with fixed points.

In contrast Ψ expresses that the action b is almost always possible throughout any infinite length run consisting only of a actions. This means that $\| \Psi \|$ is the singleton set $\{\mathbf{0}\}$ because $\mathbf{0}$ has no infinite length runs. First $\mathbf{0}$ belongs to the intersection $\bigcap\{\mathcal{F} \subseteq \mathcal{P} : \| \nu Z. [a](((\langle b \rangle \mathtt{tt} \vee Y) \wedge Z) \|_{\mathcal{V}[\mathcal{F}/Y]} \subseteq \mathcal{F}\}$. So we show that $\{\mathbf{0}\} = \bigcup\{\mathcal{E} \subseteq \mathcal{P} : \mathcal{E} \subseteq \| [a](((\langle b \rangle \mathtt{tt} \vee Y) \wedge Z) \|_{(\mathcal{V}[\{\mathbf{0}\}/Y])[\mathcal{E}/Z]}\}$. Note that $D' \notin \| [a](((\langle b \rangle \mathtt{tt} \vee Y) \wedge Z) \|_{(\mathcal{V}[\{\mathbf{0}\}/Y])[\mathcal{E}/Z]}$ as D does not have the property Y under this valuation. So D' can not belong to any $\mathcal{E}$ which is a subset of $\| [a](((\langle b \rangle \mathtt{tt} \vee Y) \wedge Z) \|_{(\mathcal{V}[\{\mathbf{0}\}/Y])[\mathcal{E}/Z]}$. This means that D is also excluded as its presence in $\mathcal{E}$ would require D' to have the property Z under this valuation. $\square$

In section 2.2 we introduced other 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 fixed points these modalities are definable as follows (where we assume that Z is not free in Φ):

$$\boxed{\begin{aligned} \langle\langle\ \rangle\rangle\Phi &\overset{\text{def}}{=} \mu Z. \Phi \vee \langle\tau\rangle Z \\ [\![\ ]\!]\Phi &\overset{\text{def}}{=} \nu Z. \Phi \wedge [\tau] Z \\ [\![\downarrow]\!]\Phi &\overset{\text{def}}{=} \mu Z. \Phi \wedge [\tau] Z \end{aligned}}$$

Therefore the 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 $\nu Z. [K](\nu Y. \Phi \wedge [\tau]Y) \wedge [\tau]Z$. Observable modal mu-calculus is the sublogic when the modalities are restricted to the subset $\{[\![\ ]\!], \langle\langle\ \rangle\rangle, [\![K]\!], \langle\langle K \rangle\rangle\}$, when $\tau \notin K$. This fixed point logic is suited for expressing observable properties of processes.

An important feature of modal mu-calculus is that it has the *finite model property*: if a closed formula holds of some process then there is a finite state process satisfying it. A proof of this can be found in [67].

3.5 Approximants

At first sight there is a chasm between the meaning of an extremal fixed point and techniques (other than exhaustive analysis) for actually finding the set it defines. There is however a more mechanical method, an iterative technique, due to Tarski and others, for discovering least and greatest 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 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 the set $g(\nu^i g)$. 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 a possibly decreasing sequence of sets, $\nu^0 g \supseteq \nu^1 g \supseteq \ldots \supseteq \nu^i g \supseteq \ldots$. The required set νg is a *subset* of each member of this sequence. 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 $\nu^n g$. 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, the case $i = n$, 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 $\lambda \mathcal{E} \subseteq \mathcal{P}.\ \|\langle \mathtt{tick} \rangle Z\|_{\mathcal{V}[\mathcal{E}/Z]}$.

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

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 $n \geq 2$. Consequently νg is the singleton set $\{Cl\}$. □

When $\mathcal{P}$ is not a finite set of processes, we can still guarantee that νg is reachable iteratively by invoking ordinals as indices. Ordinals can be ordered as follows:

$$0, 1, \ldots, \omega, \omega + 1, \ldots, \omega + \omega, \omega + \omega + 1, \ldots$$

Here ω is the initial *limit* ordinal (one without an immediate predecessor) while $\omega + 1$ is its successor[17]. Assume that α, β and λ range over ordinals. 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[18], 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 $C \overset{\text{def}}{=} \mathtt{in}(x).B_x$ with $x : \mathbb{N}$, and $B_{n+1} \overset{\text{def}}{=} \mathtt{down}.B_n$ 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.)

$$\begin{aligned}
\nu^0 g &= \mathcal{P} &&= \{C, B_i \ : \ i \geq 0\} \\
\nu^1 g &= \|\langle - \rangle Z\|_{\mathcal{V}[\nu^0 g/Z]} &&= \{C, B_i \ : \ i \geq 1\} \\
&\vdots \\
\nu^{j+1} g &= \|\langle - \rangle Z\|_{\mathcal{V}[\nu^j g/Z]} &&= \{C, B_i \ : \ i \geq j+1\}
\end{aligned}$$

The set $\nu^\omega g$, defined as $\bigcap\{\nu^i g \ : \ i < \omega\}$, is the singleton set $\{C\}$ as each B_j fails to belong to it. The next iterate is the fixed point, $\nu^{\omega+1} g$ is $\|\langle - \rangle Z\|_{\mathcal{V}[\nu^\omega g/Z]}$ which is $\emptyset$. Here stabilization occurs at $\nu^{\omega+1} g$ with $\emptyset$ as the fixed point νg. □

[17] 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 $\mathbb{N}$.

[18] The proof is similar to that described for the finite state case earlier.

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 the possibly increasing sequence of sets $\mu^0 g \subseteq \mu^1 g \subseteq \ldots \subseteq \mu^i g \subseteq \ldots$ and μg is a *superset* of each of the sets $\mu^i g$. 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 sets $\mu^i g$ starting with $\mu^0 g$ until it is the same as its successor. 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 $\mathcal{P}$ is as in example 1.

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

Stabilization occurs at $\mu^2 g$ which is the required fixed point. Notice that if we consider νg instead then we obtain the following different set.

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

This stabilizes at the initial point. $\square$

If $\mathcal{P}$ is not a finite set then again 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. Therefore 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 a superset of each member of this sequence, and also occurs within it, and the first such time is not when the ordinal is a limit.

Example 4 Consider the following clock Cl'

$$\begin{aligned}
Cl' &\stackrel{\mathrm{def}}{=} \textstyle\sum\{Cl^i \ : \ i \geq 0\} \\
Cl^{i+1} &\stackrel{\mathrm{def}}{=} \mathtt{tick}.Cl^i
\end{aligned}$$

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 $\lambda \mathcal{E} \subseteq \mathcal{P}.\ \| [\mathtt{tick}]Z \|_{\mathcal{V}[\mathcal{E}/Z]}$.

$$\begin{aligned}
\mu^0 g &= \emptyset \\
\mu^1 g &= \| [\mathtt{tick}]Z \|_{\mathcal{V}[\mu^0 g/Z]} = \{Cl^j \ : \ j < 1\} \\
&\vdots \\
\mu^{i+1} g &= \| [\mathtt{tick}]Z \|_{\mathcal{V}[\mu^i g/Z]} = \{Cl^j \ : \ j < i+1\}
\end{aligned}$$

So the initial limit point $\mu^\omega g$ is $\bigcup\{\mu^i g \,:\, i < \omega\}$ which is $\{Cl^j \,:\, j \geq 0\}$. At the next iteration the required fixed point is reached as $\mu^{\omega+1}g$ is $\|\,[\mathtt{tick}]Z\,\|_{\mathcal{V}[\mu^\omega g/Z]}$ which is $\mathcal{P}$, and moreover $\mu^\alpha g = \mathcal{P}$ for all $\alpha \geq \omega + 1$. $\square$

The sets $\sigma^\alpha g$ are *approximants* for σg in that they converge towards it. Each $\nu^\alpha g$ approximates νg from above, whereas each $\mu^\alpha g$ 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 $\|\,\nu Z.\Phi\,\|_{\mathcal{V}}$ 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 M_∞. 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 is that we can now give a more direct definition of $E \models_{\mathcal{V}} \Phi$ when Φ is a fixed point formula:

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

Example 5 In the previous section we contrasted the definitions of $[\![\ ]\!]\Phi$ and $[\![\downarrow]\!]\Phi$ in modal mu-calculus. Let $\nu Z.\Psi$ be the formula $\nu Z.\Phi \wedge [\tau]Z$ (expressing $[\![\ ]\!]\Phi$) and let $\mu Z.\Psi$ be $\mu Z.\Phi \wedge [\tau]Z$ (expressing $[\![\downarrow]\!]\Phi$)[19]. Consider the different approximants these formulas generate:

$$\begin{array}{ll}
\nu Z^0.\Psi = \mathtt{tt} & \mu Z^0.\Psi = \mathtt{ff} \\
\nu Z^1.\Psi = \Phi \wedge [\tau]\mathtt{tt} \ = \ \Phi & \mu Z^1.\Psi = \Phi \wedge [\tau]\mathtt{ff} \\
\nu Z^2.\Psi = \Phi \wedge [\tau]\Phi & \mu Z^2.\Psi = \Phi \wedge [\tau](\Phi \wedge [\tau]\mathtt{ff}) \\
\vdots \qquad \vdots & \vdots \qquad \vdots
\end{array}$$

$$\begin{array}{l}
\nu Z^i.\Psi = \Phi \wedge [\tau](\Phi \wedge [\tau](\Phi \wedge \ldots [\tau]\Phi \ldots)) \\
\mu Z^i.\Psi = \Phi \wedge [\tau](\Phi \wedge [\tau](\Phi \wedge \ldots [\tau](\Phi \wedge [\tau]\mathtt{ff}) \ldots)) \\
\vdots \qquad \vdots
\end{array}$$

The approximant $\mu Z^i.\Psi$ carries the extra demand that there can not be a sequence of silent actions of length i. Hence $[\![\downarrow]\!]\Phi$ requires all immediate τ behaviour to eventually peter out. $\square$

[19] It is assumed that Z is not free in Φ.

3.6 Embedded approximants

Approximants provide an iterative technique for discovering fixed point sets. All the examples in the previous section involved a single fixed point. In this section we examine the technique in the presence of multiple fixed points, and comment on entanglement of approximants.

Example 1 The vending machine *Ven* has the property $\nu Z. [\mathsf{2p}, \mathtt{1p}]\Psi \wedge [-]Z$, when Ψ is $\mu Y. \langle - \rangle \mathtt{tt} \wedge [-\{\mathtt{collect}_b, \mathtt{collect}_l\}]Y$. Let $\mathcal{P}$ be the transition closed set $\{ Ven, Ven_b, Ven_l, \mathtt{collect}_b. Ven, \mathtt{collect}_l. Ven\}$. First using approximants we evaluate the embedded fixed point Ψ, and we abbreviate its ith approximant to μY^i:

$$\begin{aligned}
\mu Y^0 &= \emptyset \\
\mu Y^1 &= \| \langle - \rangle \mathtt{tt} \wedge [-\{\mathtt{collect}_b, \mathtt{collect}_l\}]Y \|_{\mathcal{V}[\mu Y^0/Y]} \\
&= \{\mathtt{collect}_b. Ven, \mathtt{collect}_l. Ven\} \\
\mu Y^2 &= \| \langle - \rangle \mathtt{tt} \wedge [-\{\mathtt{collect}_b, \mathtt{collect}_l\}]Y \|_{\mathcal{V}[\mu Y^1/Y]} \\
&= \{ Ven_b, Ven_l, \mathtt{collect}_b. Ven, \mathtt{collect}_l. Ven\} \\
\mu Y^3 &= \| \langle - \rangle \mathtt{tt} \wedge [-\{\mathtt{collect}_b, \mathtt{collect}_l\}]Y \|_{\mathcal{V}[\mu Y^2/Y]} \\
&= \mathcal{P}
\end{aligned}$$

Next the outermost fixed point is evaluated, given that the meaning of Ψ is $\mathcal{P}$. We abbreviate its ith approximant to νZ^i.

$$\begin{aligned}
\nu Z^0 &= \mathcal{P} \\
\nu Z^1 &= \| [\mathsf{2p}, \mathtt{1p}]\Psi \wedge [-]Z \|_{\mathcal{V}[\nu Z^0/Z]} \\
&= \mathcal{P}
\end{aligned}$$

Here the embedded fixed point can be evaluated independently of the outermost fixed point. □

Example 1 illustrates how the iterative technique works for formulas with multiple fixed points that are independent of each other. In abstract terms, the formula of example 1 has the form $\nu Z. \Phi(Z, \mu Y. \Psi(Y))$ where the notation makes explicit which variables can be be free in subformulas: here Z does not occur free within the subformula $\mu Y. \Psi(Y)$ but may occur within $\Phi(Z, \mu Y. \Psi(Y))$. Consequently when evaluating the outermost fixed point we have that:

$$\begin{aligned}
\nu Z^0 &= \mathcal{P} \\
\nu Z^1 &= \| \Phi(Z, \mu Y. \Psi(Y)) \|_{\mathcal{V}[\nu Z^0/Z]} \\
&\vdots \\
\nu Z^{i+1} &= \| \Phi(Z, \mu Y. \Psi(Y)) \|_{\mathcal{V}[\nu Z^i/Z]} \\
&\vdots
\end{aligned}$$

Throughout these approximants the meaning of the subformula $\mu Y. \Psi(Y)$ is invariant because it does not contain Z free: consequently $\| \mu Y. \Psi(Y) \|_{\mathcal{V}[\nu Z^\alpha/Z]}$ is the same set as $\| \mu Y. \Psi(Y) \|_{\mathcal{V}[\nu Z^\beta/Z]}$ for any ordinals α and β.

Example 2 Assume that D and D' are as in example 1 of section 3.4, where $D \xrightarrow{a} D'$, $D' \xrightarrow{a} D$, and $D' \xrightarrow{b} \mathbf{0}$, and $\mathcal{P}$ is $\{D, D', \mathbf{0}\}$. Also recall the pair of formulas Φ and Ψ.

$$\begin{aligned}\Phi &= \nu Z.\, \mu Y.\, [a]((\langle b\rangle \mathtt{tt} \wedge Z) \vee Y)\\ \Psi &= \mu Y.\, \nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\end{aligned}$$

Consider now evaluating Φ using approximants.

$$\begin{aligned}
&\nu Z^0 = \mathcal{P}\\
&\nu Z^1 = \|\, \mu Y.\, [a]((\langle b\rangle \mathtt{tt} \wedge Z) \vee Y)\,\|_{\mathcal{V}[\nu Z^0/Z]}\\
&\qquad \mu Y^{00} = \emptyset\\
&\qquad \mu Y^{01} = \|\, [a]((\langle b\rangle \mathtt{tt} \wedge Z) \vee Y)\,\|_{(\mathcal{V}[\nu Z^0/Z])[\mu Y^{00}/Y]}\\
&\qquad\quad = \{\mathbf{0}, D\}\\
&\qquad \mu Y^{02} = \|\, [a]((\langle b\rangle \mathtt{tt} \wedge Z) \vee Y)\,\|_{(\mathcal{V}[\nu Z^0/Z])[\mu Y^{01}/Y]}\\
&\qquad\quad = \mathcal{P}\\
&\text{So } \nu Z^1 = \mathcal{P}
\end{aligned}$$

Here calculating νZ^1 depends on calculating the innermost fixed point when Z is contained within it, and therefore when it initially receives the value νZ^0 for Z. Hence the notation: μY^{ji} represents the ith approximant of the subformula prefaced with μY when any free occurrence of Z is understood as the approximant νZ^j. The case of Ψ illustrates this further.

$$\begin{aligned}
&\mu Y^0 = \emptyset\\
&\mu Y^1 = \|\, \nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{\mathcal{V}[\mu Y^0/Y]}\\
&\qquad \nu Z^{00} = \mathcal{P}\\
&\qquad \nu Z^{01} = \|\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{(\mathcal{V}[\mu Y^0/Y])[\nu Z^{00}/Z]}\\
&\qquad\quad = \{\mathbf{0}, D\}\\
&\qquad \nu Z^{02} = \|\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{(\mathcal{V}[\mu Y^0/Y])[\nu Z^{01}/Z]}\\
&\qquad\quad = \{\mathbf{0}\}\\
&\qquad \nu Z^{03} = \|\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{(\mathcal{V}[\mu Y^0/Y])[\nu Z^{02}/Z]}\\
&\qquad\quad = \{\mathbf{0}\}\\
&\text{So } \mu Y^1 = \{\mathbf{0}\}\\
&\mu Y^2 = \|\, \nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{\mathcal{V}[\mu Y^1/Y]}\\
&\qquad \nu Z^{10} = \mathcal{P}\\
&\qquad \nu Z^{11} = \|\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{(\mathcal{V}[\mu Y^1/Y])[\nu Z^{10}/Z]}\\
&\qquad\quad = \{\mathbf{0}, D\}\\
&\qquad \nu Z^{12} = \|\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{(\mathcal{V}[\mu Y^1/Y])[\nu Z^{11}/Z]}\\
&\qquad\quad = \{\mathbf{0}\}\\
&\qquad \nu Z^{13} = \|\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)\,\|_{(\mathcal{V}[\mu Y^1/Y])[\nu Z^{12}/Z]}\\
&\qquad\quad = \{\mathbf{0}\}\\
&\text{So } \mu Y^2 = \{\mathbf{0}\}
\end{aligned}$$

Here we need to evaluate the innermost fixed point with respect to more than one outermost approximant. $\square$

Example 2 illustrates dependency of fixed points. The first formula has the shape $\nu Z.\Phi(Z,\mu Y.\Psi(Z,Y))$ where Z is free in the innermost fixed point. Evaluation of such a formula using approximants takes the form:

$$\begin{array}{ll}
\nu Z^0 & = \mathcal{P} \\
\nu Z^1 & = \| \Phi(Z,\mu Y.\Psi(Z,Y)) \|_{\mathcal{V}[\nu Z^0/Z]} \\
 & \qquad \mu Y^{00} \; = \; \emptyset \\
 & \qquad \mu Y^{01} \; = \| \Psi(Z,Y) \|_{(\mathcal{V}[\nu Z^0/Z])[\mu Y^{00}/Y]} \\
 & \qquad \vdots \\
\vdots & \\
\nu Z^{i+1} & = \| \Phi(Z,\mu Y.\Psi(Z,Y)) \|_{\mathcal{V}[\nu Z^i/Z]} \\
 & \qquad \mu Y^{i0} \; = \; \emptyset \\
 & \qquad \mu Y^{i1} \; = \| \Psi(Z,Y) \|_{(\mathcal{V}[\nu Z^i/Z])[\mu Y^{i0}/Y]} \\
 & \qquad \vdots
\end{array}$$

The meaning of the subformula $\mu Y.\Psi(Z,Y)$ may vary according to the interpretation of Z.

Approximants for $\nu Z.\Phi$ and $\mu Z.\Phi$ start from the sets $\nu Z^0 = \mathcal{P}$ and $\mu Z^0 = \emptyset$. In principle, there will be fewer calculations if the initial approximants are closer to the required fixed points. A set $\nu Z^0 = \mathcal{E}$ where $\mathcal{P} \supseteq \mathcal{E} \supseteq \| \nu Z.\Phi \|$ could be a better initial point than $\mathcal{P}$. This observation can be utilized when evaluating an embedded fixed point formula whose shape is $\nu Z.\Phi(Z,\nu Y.\Psi(Z,Y))$.

$$\begin{array}{ll}
\nu Z^0 & = \mathcal{P} \\
\nu Z^1 & = \| \Phi(Z,\nu Y.\Psi(Z,Y)) \|_{\mathcal{V}[\nu Z^0/Z]} \\
 & \qquad \nu Y^{00} \; = \; \mathcal{P} \\
 & \qquad \nu Y^{01} \; = \| \Psi(Z,Y) \|_{(\mathcal{V}[\nu Z^0/Z])[\nu Y^{00}/Y]} \\
 & \qquad \vdots \\
\nu Z^2 & = \| \Phi(Z,\nu Y.\Psi(Z,Y)) \|_{\mathcal{V}[\nu Z^1/Z]}
\end{array}$$

To evaluate νZ^2 one needs to calculate $\| \nu Y.\Psi(Z,Y) \|_{\mathcal{V}[\nu Z^1/Z]}$. However by monotonicity we know that

$$\| \nu Y.\Psi(Z,Y) \|_{\mathcal{V}[\nu Z^1/Z]} \subseteq \| \nu Y.\Psi(Z,Y) \|_{\mathcal{V}[\nu Z^0/Z]} \subseteq \mathcal{P}$$

Therefore we can use $\| \nu Y.\Psi(Z,Y) \|_{\mathcal{V}[\nu Z^0/Z]}$ as the initial approximant νY^{10}, and so on:

$$
\begin{array}{ll}
\nu Z^0 & = \mathcal{P} \\
\nu Z^1 & = \| \Phi(Z, \nu Y. \Psi(Z,Y)) \|_{\mathcal{V}[\nu Z^0/Z]} \\
& \quad \nu Y^{00} \ = \ \emptyset \\
& \quad \nu Y^{01} \ = \| \Psi(Z,Y) \|_{(\mathcal{V}[\nu Z^0/Z])[\nu Y^{00}/Y]} \\
& \quad \vdots \\
\nu Z^2 & = \| \Phi(Z, \nu Y. \Psi(Z,Y)) \|_{\mathcal{V}[\nu Z^1/Z]} \\
& \quad \nu Y^{10} \ = \| \nu Y. \Psi(Z,Y) \|_{\mathcal{V}[\nu Z^0/Z]} \\
& \quad \nu Y^{11} \ = \| \Psi(Z,Y) \|_{(\mathcal{V}[\nu Z^0/Z])[\nu Y^{10}/Y]} \\
& \quad \vdots \\
\nu Z^{i+1} & = \| \Phi(Z, \nu Y. \Psi(Z,Y)) \|_{\mathcal{V}[\nu Z^i/Z]} \\
& \quad \nu Y^{i+10} \ = \| \nu Y. \Psi(Z,Y) \|_{\mathcal{V}[\nu Z^i/Z]} \\
& \quad \nu Y^{i+11} \ = \| \Psi(Z,Y) \|_{(\mathcal{V}[\nu Z^0/Z])[\nu Y^{i+10}/Y]} \\
& \quad \vdots \\
\vdots &
\end{array}
$$

This observation can be extended to formulas with multiple embedded maximal fixed points.

The situation is dual for least fixed points. Choosing the set $\mathcal{E}$ for μZ^0 when $\emptyset \subseteq \mathcal{E} \subseteq \| \mu Z. \Phi \|$ could be a better starting point than $\emptyset$. In the case of a formula whose shape is $\mu Z. \Phi(Z, \mu Y. \Psi(Z,Y))$, by monotonicity we know that

$$\emptyset \subseteq \| \mu Y. \Psi(Z,Y) \|_{\mathcal{V}[\mu Z^i/Z]} \subseteq \| \mu Y. \Psi(Z,Y) \|_{\mathcal{V}[\mu Z^{i+1}/Z]}$$

and so the set $\| \mu Y. \Psi(Z,Y) \|_{\mathcal{V}[\mu Z^i/Z]}$ can be used as the initial approximant μY^{i+10}.

This insight is of no help for evaluating alternating fixed points, such as $\nu Z. \Phi(Z, \mu Y. \Psi(Z,Y))$. We cannot use the set $\| \mu Y. \Psi(Z,Y) \|_{\mathcal{V}[\nu Z^0/Z]}$ as the initial approximant for $\| \mu Y. \Psi(Z,Y) \|_{\mathcal{V}[\nu Z^1/Z]}$ because the ordering is now reversed: $\nu Z^0 \supseteq \nu Z^1$, and so $\| \mu Y. \Psi(Z,Y) \|_{\mathcal{V}[\nu Z^0/Z]} \supseteq \| \mu Y. \Psi(Z,Y) \|_{\mathcal{V}[\nu Z^1/Z]}$. We do not know how to approximant a least fixed point from above or a greatest fixed point from below. The amount of alternation of fixed points in a formula has become a crucial measure when developing algorithms for checking properties of finite state systems, see [30, 47].

3.7 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 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 fixed points are expressible using infinitary conjunction and disjunction, and that infinitary modal logic M_∞ characterizes bisimulation equivalence exactly. Recall the example of the two clocks in section 2.5 that showed that image finiteness is essential in the case of modal logic. The two clocks have the same modal properties but they are not bisimilar. The presence of fixed points allows us to distinguish them because one of the clocks 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 continuing need for image finiteness (or a weakened version of it). Let $\{Q_i \,:\, i \in I\}$ be the set of all finite state processes whose actions belong to $\{a, b\}$, and assuming $n \in \mathbb{N}$ let $P(n) \stackrel{\mathrm{def}}{=} a^n.b.P(n+1)$, $R \stackrel{\mathrm{def}}{=} \sum\{a.Q_i \,:\; i \in I\}$, and $P \stackrel{\mathrm{def}}{=} P(1) + R$ (where $a^0.E$ is E and $a^{n+1}.E$ is $a^n.a.E$). The behaviour of $P(1)$ is:

$$P(1) \xrightarrow{ab} P(2) \xrightarrow{aab} P(3) \xrightarrow{aaab} \ldots \xrightarrow{a^n b} P(n+1) \xrightarrow{a^{n+1}b} \ldots$$

Consequently P and R are not bisimilar as this would require there to be a finite state process Q_j that is bisimilar to $b.P(2)$, which is not possible (via the pumping lemma for regular languages). On the other hand, P and R have the same closed modal mu-calculus properties.

It also turns out that two bisimilar processes have the same modal mu-calculus properties provided that they are expressed using closed formulas. Let Γ be this set of closed formulas, and let $\equiv_\Gamma$ be as in section 2.5.

Proposition 1 *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, fairness, and cyclic properties when expressed using closed formulas. There is an indirect proof of this Proposition via M_∞ as the set Γ is a sublogic of it, and bisimulation equivalence preserves properties expressible within it. However we shall show how it can be proved directly, as 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 $F \in \mathcal{E}$. Proposition 1 is equivalent to the claim that for any closed Φ and set $\mathcal{P}$ the subset $\|\Phi\|$ is bisimulation closed.

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 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 $\mathcal{V}^u(Z) = (\mathcal{V}(Z))^u$. Proposition 1 is a corollary of the following result where Φ is an arbitrary formula of modal mu-calculus which therefore may contain free variables.

Proposition 2 *If $\mathcal{V}$ is bisimulation closed then $\| \Phi \|_{\mathcal{V}}$ is bisimulation closed.*

The proof of Proposition 2 is by simultaneous induction on the structure of Φ with the following three propositions:

1. If $\mathcal{V}$ is bisimulation closed then $\| \Phi \|_{\mathcal{V}}$ is bisimulation closed.
2. If $\| \Phi \|_{\mathcal{V}} \subseteq \mathcal{E}$ then $\| \Phi \|_{\mathcal{V}^d} \subseteq \mathcal{E}^d$.
3. If $\mathcal{E} \subseteq \| \Phi \|_{\mathcal{V}}$ then $\mathcal{E}^u \subseteq \| \Phi \|_{\mathcal{V}^u}$.

This result tells us more than that bisimilar processes have the same properties when expressed using closed formulas. They also have the same properties when expressed by open formulas provided that the meanings of the free variables are bisimulation closed. The proof of this result also establishes that closed formulas of observable modal mu-calculus (built using the modal operators $[\![K]\!]$, $[\![\ ]\!]$, $\langle\!\langle K \rangle\!\rangle$, and $\langle\!\langle\ \rangle\!\rangle$) are preserved by observable bisimulation equivalence.

3.8 Expressing properties

Modal mu-calculus is a very powerful temporal logic which permits expression of a very rich class of properties. In this section we examine how to express a range of properties that pick out important features of processes.

Informally a safety property states that some bad feature is always precluded. Safety can either be ascribed to *states*, that bad states can never be reached, or to *actions*, that bad actions never happen. In the former case if the formula Φ^c captures those bad states then $\nu Z.\, \Phi \wedge [-]Z$ expresses safety.

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: these bad states can be captured by the formula $([\texttt{tcross}]\texttt{ff} \vee [\texttt{ccross}]\texttt{ff})^c$. Therefore the required safety property is $\nu Z.([\texttt{tcross}]\texttt{ff} \vee [\texttt{ccross}]\texttt{ff}) \wedge [-]Z$. $\square$

It is useful to allow the full freedom of the property notation by employing open formulas with free variables and appropriate valuations which capture their intended meaning. For instance in the case of safety assume that $\mathcal{E}$ is the family of bad states, and so the formula $\nu Z.\, Q \wedge [-]Z$ expresses safety relative to the valuation $\mathcal{V}$ which assigns $\mathcal{P} - \mathcal{E}$ to Q. The variable Q has a definite intended meaning as given by $\mathcal{V}$.

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 the open formula $\nu Z.Q \wedge [-]Z$ with the free variable Q relative to the valuation $\mathcal{V}$ which assigns the set $\mathcal{P} - \{SM_j : j < 0\}$ to Q. Here a bad state is a slot machine with a negative amount of money. $\square$

The idea is that free variables should only express immediate properties of processes, as in example 2 (and not temporal features). When $\mathcal{V}$ is bisimulation closed with respect to the free variables of Φ, we say that the property expressed by Φ relative to $\mathcal{V}$ is *extensional*: by Proposition 2 of the previous section this implies that $\| \Phi \|_{\mathcal{V}}$ is also bisimulation closed 2 of the previous section. The safety formula in example 2 is extensional. If Q expresses a feature such as "has at least three parallel components" then it is intensional.

Safety can also be ascribed to actions, that no action in K ever happens, which is expressed by the formula $\nu Z. [K]\mathtt{ff} \wedge [-]Z$. However there is really no distinction between safety in terms of bad states and in terms of bad actions. For the action case is just equivalent to saying that a bad state obeying $([K]\mathtt{ff})^c$, that is $\langle K \rangle \mathtt{tt}$, can not be reached.

A liveness property states that some good feature is eventually fulfilled. Again it can either be ascribed to states, that a good state is eventually reached, or to actions, that a good action eventually happens. If Φ captures the good states then $\mu Z. \Phi \vee (\langle - \rangle \mathtt{tt} \wedge [-]Z)$ expresses liveness with respect to state. Note the presence of $\langle - \rangle \mathtt{tt}$ to ensure that Φ does become true. In contrast that eventually some action in K happens is expressed by the formula $\mu Z. \langle - \rangle \mathtt{tt} \wedge [-K]Z$ which states that any performance of actions other than K is well-founded, and not because the process terminates.

There is not a reformulation of liveness with respect to actions in terms of liveness with respect to state, as a formula of the form $\mu Z. \Phi \vee (\langle - \rangle \mathtt{tt} \wedge [-]Z)$ where Φ does not contain fixed points. For instance it is not expressed by either of the following pair:

$$\mu Z. \langle K \rangle \mathtt{tt} \vee (\langle - \rangle \mathtt{tt} \wedge [-]Z)$$
$$\mu Z. (\langle - \rangle \mathtt{tt} \wedge [-K]\mathtt{ff}) \vee (\langle - \rangle \mathtt{tt} \wedge [-]Z)$$

The first is too weak as it merely states that eventually some action in K is possible without any guarantee that it happens. The second is too strong as it states that eventually only K actions are possible (and therefore must then happen).

Weak liveness and safety properties may also be ascribed to states or actions. However recall that weak liveness is the complement of safety and weak safety is the complement of liveness. That Φ is eventually true in some run is given by $(\nu Z. \Phi^c \wedge [-]Z)^c$ which is the formula $\mu Z. \Phi \vee \langle - \rangle Z$. And that some action in K happens in some run is expressed by $(\nu Z. [K]\mathtt{ff} \wedge [-]Z)^c$ which is the formula $\mu Z. \langle K \rangle \mathtt{tt} \vee \langle - \rangle Z$. Weak state safety, that Φ is true throughout some run, is expressed by $(\mu Z. \Phi^c \vee (\langle - \rangle \mathtt{tt} \wedge [-]Z))^c$, which is the formula $\nu Z. \Phi \wedge ([-]\mathtt{ff} \vee \langle - \rangle Z)$ and that there is a run where no action in K occurs is $(\mu Z. \langle - \rangle \mathtt{tt} \wedge$

$[-K]Z)^c$ which is $\nu Z.\, [-]\mathtt{ff} \vee \langle -K\rangle Z$. So in the case of weak safety there is the distinction between state and action.

Liveness and safety may relate to subsets of runs. For instance they may be triggered by particular actions or states. A simple case is that if action a ever happens then eventually b happens, so any run with an a action must contain a later b action. This is expressed by the formula $\nu Z.\, [a](\mu Y.\, \langle -\rangle \mathtt{tt} \wedge [-b]Y) \wedge [-]Z$. A safety example is that whenever Ψ becomes true, Φ^c is then always precluded, expressed by $\nu Z.\, (\Psi^c \vee (\Psi \wedge \nu Y.\, \Phi \wedge [-]Y)) \wedge [-]Z$.

More complex is the expression of liveness properties under fairness. An example is the property that in any run if the actions b and c happen infinitely often then so does a which is expressed as follows:

$$\nu Z.\, (\mu X.\, [b](\nu Y.\, [c](\nu Y_1.\, X \wedge [-a]Y_1) \wedge [-a]Y) \wedge [-]Z)$$

Here there is an essential fixed point dependency, as the occurrence of X is free within the fixed point subformula prefaced with νY.

Example 3 The desirable liveness property for the crossing of figure 7 is that whenever a car approaches the crossing eventually it crosses is captured by the formula

$$\nu Z.\, [\mathtt{car}](\mu Y.\, \langle -\rangle \mathtt{tt} \wedge [-\overline{\mathtt{ccross}}]Y) \wedge [-]Z$$

However this only holds if we assume that the signal is fair. Let Q and R be variables and $\mathcal{V}$ a valuation such that Q is true when the crossing is in any state where *Rail* has the form $\mathtt{green}.\overline{\mathtt{tcross}}.\overline{\mathtt{red}}.\mathit{Rail}$ (the states E_2, E_3, E_6 and E_{10} of figure 8) and R holds when it is in any state where *Road* has the form $\mathtt{up}.\overline{\mathtt{ccross}}.\overline{\mathtt{down}}.\mathit{Road}$ (the states E_1, E_3, E_7 and E_{11}). The liveness property is: for any run if Q^c is true infinitely often and R^c is also true infinitely often then whenever a car approaches eventually it crosses. This is expressed by the following open formula relative to $\mathcal{V}$

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

The property expressed here is extensional. In this case we can view Q and R as probes in the sense of [70]. □

Another class of properties is until properties. These are of the form Φ remains true until Ψ becomes true, or in terms of actions K actions happen until a J action occurs (or a mixture of state and action). Again they can be viewed as holding of all runs, or some runs, or of a particular family of runs which obey a condition. The formula $\mu Y.\, \Psi \vee (\Phi \wedge \langle -\rangle \mathtt{tt} \wedge [-]Y)$ expresses that Φ holds until Ψ in every run. Note here the requirement that Ψ does eventually become true. This commitment can be removed by changing fixed points. The property that in every run Φ remains true *unless* Ψ holds does not imply that Ψ does become true, and so is expressed as $\nu Y.\, \Psi \vee (\Phi \wedge [-]Y)$.

Sometimes we are only interested in part of the behaviour of a process. There are many ways to understand what part of a behaviour means. A simple case is when attention is restricted to a subset of the actions that a process can perform. Liveness, safety and until properties can therefore be relativized in this way. An example is the property that Φ is eventually true in any run consisting of K actions.

Cyclic properties can also be described in the logic. A simple example is that each even action is `tock`: if $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots$ is a finite or infinite length run then each action a_{2i} is `tock`. This is expressed as $\nu Z.\, [-]([-\texttt{tock}]\texttt{ff} \wedge [-]Z)$. The clock $Cl_1 \overset{\text{def}}{=} \texttt{tick.tock.}Cl_1$ has this property. It also has the tighter cyclic property that every run involves the repeated cycling of `tick` and `tock` actions, expressed as $\nu Z.\, [-\texttt{tick}]\texttt{ff} \wedge [\texttt{tick}]([-\texttt{tock}]\texttt{ff} \wedge [-]Z)$[20]. These properties can also be weakened to some family of runs. Cyclic properties that allow other actions to intervene within a cycle can also be expressed.

Example 4 Recall the scheduler from section 1.4 which timetables a sequence of tasks, and which must ensure that a task cannot be restarted until its previous performance has finished. Suppose that initiation of one of the tasks is given by the action a and its termination by b. The scheduler therefore has to guarantee the cyclic behaviour $a\,b$ when other actions may occur before and after each occurrence of a and each occurrence of b. This property can be defined inductively:

$$\begin{aligned} cycle(ab) &= [b]\texttt{ff} \wedge [a]\,cycle(ba) \wedge [-a]\,cycle(ab) \\ cycle(ba) &= [a]\texttt{ff} \wedge [b]\,cycle(ab) \wedge [-b]\,cycle(ba) \end{aligned}$$

Here we have left open the possibility that runs have finite length: appropriate occurrences of $\langle - \rangle\texttt{tt}$ preclude it. An important issue is whether these recursive definitions are to be interpreted with least or greatest fixed points, or even with a mixture of them. This depends upon whether intervening actions are allowed to go on forever without the next a or b happening. If we prohibit this, the property is expressed as follows:

$$\mu Y.\, [b]\texttt{ff} \wedge [a](\mu Z.\, [a]\texttt{ff} \wedge [b]Y \wedge [-b]Z) \wedge [-a]Y$$

The number of actions in the cycle can be extended. □

Another class of properties is given by counting. An instance is that in each run there are exactly two a actions, given by:

$$\mu X.\, [a](\mu Y.\, [a](\nu Z.\, [a]\texttt{ff} \wedge [-]Z) \wedge \langle - \rangle\texttt{tt} \wedge [-a]Y) \wedge \langle - \rangle\texttt{tt} \wedge [-a]X$$

Another example is that in each run a can only happen finitely often, $\mu X.\, \nu Y.\, [a]X \wedge [-a]Y$.

However there are also many counting properties that are not expressible in the logic. A notable case is the following property of a buffer (which is a consequence of [62]): the number of `out` actions never exceeds the number of `in` actions.

[20] This formula leaves open the possibility that a run has finite length. To preclude it one adds $\langle - \rangle\texttt{tt}$ at the outer and inner level.

4 Verifying Temporal Properties

A very rich temporal logic has been introduced which is able to describe useful liveness, safety, cyclic and other properties of processes. The next step is to provide techniques for verification, for showing when processes have, or fail to 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, and a proof of a modal property thereby reduces to proofs of subproperties, as stipulated by the inductive definition of satisfaction. Therefore the transition graph of a process is not needed when proving modal properties.

However in the case of modal mu-calculus the satisfaction relation between processes and formulas is defined indirectly. The primary semantics of a formula Φ is defined in terms of every process in a transition closed set which has the property. One method for determining whether E has Φ is to first present a transition closed set of processes containing E, second to calculate $\|\Phi\|_{\mathcal{V}}$ with respect to this set, and then finally to check whether E belongs to it. When E has a small transition graph this is a reasonable technique. 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, safety or cyclic property may involve considerable redundancy if the intention is to show that a *particular* process has it.

An alternative approach to showing that processes satisfy formulas is to appeal to their approximants as described in sections 3.5 and 3.6. A more direct definition of satisfaction is then available. However proofs will now require the use of induction over ordinals, and some care must be taken with limit ordinals. In the presence of embedded fixed points this will require the use of embedded induction. Moreover, we will then lose that simple idea that a proof of a property reduces to proofs of subproperties.

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 whether temporal properties hold. Towards this end we first provide a different characterization of the satisfaction relation between a process and a formula in terms of games. It turns out that a process has a property just in case player II has a winning strategy for the game associated with this pair. Underpinning player II's successful strategy is the notion of a successful tableau. We therefore also present tableau proof systems for property checking, which were originally developed with David Walker [66] and Julian Bradfield [17].

4.1 Games and constants

In this section we present an alternative characterization of the satisfaction relation between a process and a formula relative to a valuation in terms of game playing. A property checking game is a pair, a process and a formula, (E, Φ) relative to a valuation $\mathcal{V}$. As with equivalence games there are two players, I and

II. Player I attempts to show that E *fails* to have the property Φ relative to $\mathcal{V}$ whereas player II wishes to establish that E does have Φ. Unlike equivalence games, players here do not necessarily move in turn[21].

A play of the *property checking game* (E_0, Φ_0) relative to $\mathcal{V}$ is a finite or infinite length 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 pair (E_{j+1}, Φ_{j+1}) is determined by one of the moves of figure 16, according the main operator of Φ_j Note the duality between the rules for $\wedge$ and $\vee$,

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

Fig. 16. Rules for the next move in a game play

$[K]$ and $\langle K \rangle$, $\nu Z.\Psi$ and $\mu Z.\Psi$ as they complement each other. Each time the current game configuration is $(E, \sigma Z.\Phi)$ a new constant U is introduced as an abbreviation for $\sigma Z.\Phi$, and at the next step this fixed point is, in effect, unfolded once as the formula becomes $\Phi\{U/Z\}$[22]. The point of constants is to provide a mechanism for understanding when embedded fixed points recur.

The rules for a next move 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

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

[22] The decision to make player I responsible for introducing and unfolding constants for maximal fixed point formulas and player II responsible for least fixed points is somewhat arbitrary, as these moves never provide real choice for either player. An alternative exposition is to appeal to a third participant, a referee who makes these moves.

$E_{j+1} \not\models_{\mathcal{V}} \Phi_{j+1}$ then $E_j \not\models_{\mathcal{V}} \Phi_j$. In contrast, if player II makes this move and $E_{j+1} \models_{\mathcal{V}} \Phi_{j+1}$ then $E_j \models_{\mathcal{V}} \Phi_j$. This is clear for the rules which govern boolean and modal operators. In the case of a fixed point formula this follows 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 $U_1 \overset{\text{def}}{=} \Psi_1 \ldots U_n \overset{\text{def}}{=} \Psi_n$ 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, that $E \models_{\mathcal{V}} \sigma Z.\Phi$ iff $E \models_{\mathcal{V}} \Phi\{\sigma Z.\Phi/Z\}$, justifies the backwards soundness of the moves determined by constants.

A player *wins* a play of a game in the circumstances depicted in figure 17. If the configuration $(E, \mathtt{tt})$ or (E, Z) when $E \in \mathcal{V}(Z)$ is reached in a play

Player II wins	**Player I wins**
1. The play is $(E_0,\Phi_0)\ldots(E_n,\Phi_n)$ and either $\Phi_n = \mathtt{tt}$ or $\Phi_n = Z$ and $E \in \mathcal{V}(Z)$.	1′. The play is $(E_0,\Phi_0)\ldots(E_n,\Phi_n)$ and either $\Phi_n = \mathtt{ff}$ or $\Phi_n = Z$ and $E \notin \mathcal{V}(Z)$.
2. The play is $(E_0,\Phi_0)\ldots(E_n,\Phi_n)$ and $\Phi_n = [K]\Psi$ and the set $\{F : E \xrightarrow{a} F \text{ and } a \in K\} = \emptyset$.	2′. The play is $(E_0,\Phi_0)\ldots(E_n,\Phi_n)$ and $\Phi_n = \langle K\rangle\Psi$ and the set $\{F : E \xrightarrow{a} F \text{ and } a \in K\} = \emptyset$.
3. The play is $(E_0,\Phi_0)\ldots(E_n,\Phi_n)$ and $\Phi_n = U$ and $U \overset{\text{def}}{=} \nu Z.\Phi$ and $E_i = E_n$ and $\Phi_i = \Phi_n$ for $i < n$.	3′. The play is $(E_0,\Phi_0)\ldots(E_n,\Phi_n)$ and $\Phi_n = U$ and $U \overset{\text{def}}{=} \mu Z.\Phi$ and $E_i = E_n$ and $\Phi_i = \Phi_n$ for $i < n$.
4. The play $(E_0,\Phi_0)\ldots(E_i,\Phi_i)\ldots$ has infinite length and there is a constant $U \overset{\text{def}}{=} \nu Z.\Phi$ such that for infinitely many j, $\Phi_j = U$.	4′. The play $(E_0,\Phi_0)\ldots(E_i,\Phi_i)\ldots$ has infinite length and there is a constant $U \overset{\text{def}}{=} \mu Z.\Phi$ such that for infinitely many j, $\Phi_j = U$.

Fig. 17. Conditions for winning a game play

then player I cannot refute that $E \models_{\mathcal{V}} \mathtt{tt}$ or $E \models_{\mathcal{V}} Z$, and therefore player II wins. Instead if the configuration reached is $(E, \langle K\rangle\Phi)$ and there are no available transitions from E then player II is unable to establish that $E \models_{\mathcal{V}} \langle K\rangle\Phi$. Similar comments apply to the dual conditions 1′ and 2. The other circumstances when a player is said to win a play concern repetition. If the configuration (E, U) is repeated in a play when U abbreviates a maximal fixed point formula then player II wins. Dually if U abbreviates a least fixed point it is player I that wins[23]. More generally as a play can have infinite length this repeat condition for winning is generalized. Player I wins an infinite length play if there is a least fixed point constant U which is traversed infinitely often, and player II wins if there is a greatest fixed point constant U which occurs infinitely often, and only one of these can happen.

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 *which for infinitely many* j, $\Phi_j = U$.

This lemma shows the role of constants (as they provide an exact account of when the same fixed point subformula is repeated).

As with equivalence games, a strategy for a player is a family of rules which tells her how to move depending on what has happened earlier in the play. A player uses the strategy π in a play if all her moves in the play obey the rules in π. The strategy π is winning if the player wins every play in which she uses π. Every game (E, Φ) relative to $\mathcal{V}$ is determined, that is either player I has a winning strategy or player II has a winning strategy, and this strategy is history free in that the rules do not need to appeal to moves that occurred earlier in the play. So a strategy for player I tells her how to move in the game configurations $(E, \Phi_1 \wedge \Phi_2)$, $(E, [K]\Phi)$, $(E, \nu Z.\Phi)$ and (E, U) when $U \overset{\text{def}}{=} \nu Z.\Phi$, and a strategy for player II is similar, as it decides the next configuration when the current one is $(E, \Phi_1 \vee \Phi_2)$, $(E, \langle K \rangle \Phi)$, $(E, \mu Z.\Phi)$ and (E, U) when $U \overset{\text{def}}{=} \mu Z.\Phi$.

Player II has a winning strategy for (E, Φ) relative to $\mathcal{V}$ just in case E has the property Φ relative to $\mathcal{V}$.

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

Theorem 1 yields an alternative account of the satisfaction relation between processes and formulas. Game playing does not require explicit calculation of fixed points, nor does it depend on induction over approximant indices. Moreover it does not require the construction of the transition graph of a process. Game playing also maintains the principle that a proof of a property reduces to subproofs of subproperties, provided that we view the unfolding of a fixed point formula as a subformula. There is another feature which could be exploited, the possibility of more sophisticated game playing where moves may also be guided by the algebraic structure of a process expression.

As an infinite length game play must traverse a particular constant infinitely often, it follows that when E is finite state a play of (E, Φ) has finite length. There is also an exponential upper bound on the number of different plays up to renaming of constants of such a game. Property checking of finite state processes via game playing is therefore decidable. However this is not a very time efficient method as the length of a play may be exponential in the number of fixed point operators in a formula. In section 4.3 we provide less costly techniques based on games. For the remainder of this section we illustrate game playing.

[23] These two conditions 3 and $3'$ are in fact redundant. We only include them because they guarantee that any play of (E, Φ) has finite length when E is finite state.

Example 1 Player II has a winning strategy for the game $(Cl, \nu Z. \langle \texttt{tick} \rangle Z)$. The only possible play is $(Cl, \nu Z. \langle \texttt{tick} \rangle Z)\,(Cl, U)\;(Cl, \langle \texttt{tick} \rangle U)\,(Cl, U)$ up to renaming the constant: the winning strategy here is, in effect, the empty set of rules, as player II has to make the move (Cl, U) from the configuration $(Cl, \langle \texttt{tick} \rangle U)$, and choice of fixed point constants does not affect play. Player II also has a winning strategy for $(Cl_5, \nu Z. \langle \texttt{tick} \rangle Z)$ with respect to the slightly different clock $Cl_5 \overset{\text{def}}{=} \texttt{tick}.Cl_5 + \texttt{tick}.\mathbf{0}$. A winning play is almost identical to the previous game play, $(Cl_5, \nu Z. \langle \texttt{tick} \rangle Z)\,(Cl_5, U)\,(Cl_5, \langle \texttt{tick} \rangle U)\,(Cl_5, U)$: the important part of the winning strategy is the rule, if $(Cl_5, \langle \texttt{tick} \rangle U)$ is the current configuration then choose (Cl_5, U) as the next move. $\square$

Example 2 *Cnt* is a simple infinite state system, $Cnt \overset{\text{def}}{=} \texttt{up}.(Cnt \mid \texttt{down}.\mathbf{0})$. It has the property that it may do $\texttt{up}$ forever, as the single play of $(Cnt, \nu Z. \langle \texttt{up} \rangle Z)$ is won by player II. However this play has infinite length:

$$(Cnt, \nu Z. \langle \texttt{up} \rangle Z)\,(Cnt, U)(Cnt, \langle \texttt{up} \rangle U)\,(Cnt \mid \texttt{down}.\mathbf{0}, U)\ldots$$

Player II wins because the constant U recurs infinitely often. For a similar reason player I wins the only play of $(Cnt, \mu Z. [\texttt{up}]Z)$ as then a least fixed point constant occurs infinitely often. $\square$

Example 3 Assume that D and D' are as in example 1 of section 3.4, with $D \xrightarrow{a} D'$, $D' \xrightarrow{a} D$ and $D' \xrightarrow{b} \mathbf{0}$, and let Ψ be the formula:

$$\mu Y. \nu Z. [a]((\langle b \rangle \texttt{tt} \vee Y) \wedge Z)$$

D' (and D) fails to have the property Ψ, and so player I has a winning strategy for the game (D', Ψ). The important rules in this strategy are: if $(D, (\langle b \rangle \texttt{tt} \vee U_1) \wedge U_2)$ is the current configuration, for any U_1, U_2, then choose $(D, \langle b \rangle \texttt{tt} \vee U_1)$ and if it is $(D', (\langle b \rangle \texttt{tt} \vee U_1) \wedge U_2)$ then choose (D', U_2). The play proceeds:

$$\begin{gathered}(D', \Psi)\,(D', U)\,(D', \nu Z. [a]((\langle b \rangle \texttt{tt} \vee U) \wedge Z))\,(D', V)\\ (D', [a]((\langle b \rangle \texttt{tt} \vee U) \wedge V))\,(D, (\langle b \rangle \texttt{tt} \vee U) \wedge V)\,(D, \langle b \rangle \texttt{tt} \vee U)\end{gathered}$$

At this last configuration 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:

$$\begin{gathered}(D, U)\,(D, \nu Z. [a]((\langle b \rangle \texttt{tt} \vee U) \wedge Z))\,(D, W)\\ (D, [a]((\langle b \rangle \texttt{tt} \vee U) \wedge W))\,(D', (\langle b \rangle \texttt{tt} \vee U) \wedge W)\,(D', W)\\ (D', [a]((\langle b \rangle \texttt{tt} \vee U) \wedge W))\,(D, (\langle b \rangle \texttt{tt} \vee U) \wedge W)\,(D, \langle b \rangle \texttt{tt} \vee U)\end{gathered}$$

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 the essential requirement in game playing that a new constant is introduced when a fixed point is met, and so both V and W are introduced for the same fixed point formula. $\square$

4.2 Tableaux

When processes are game equivalent, as developed in section 2, a bisimulation relation expresses player II's winning strategy. We now develop a *tableau* proof system for property checking so that a successful proof tree captures a winning strategy for player II. Here we restrict ourselves to finitely branching processes. The proof system here was first presented in [66]. 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 in section 1. 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. 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 18. Again we use

$$\wedge \; \frac{E \vdash \Phi \wedge \Psi}{E \vdash \Phi \quad\quad E \vdash \Psi}$$

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

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

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

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

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

Fig. 18. Tableau rules

auxiliary propositional constants, ranged over by U, as abbreviations of fixed point formulas.

The rules are backwards sound in the sense that if all the consequents of any rule are true then so is the premise goal. This is clear for fixed point formulas 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. As the rules are backwards sound, 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 **n** labelled with the sequent $F \vdash \Psi$ is *terminal* if one of the following conditions hold:

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

It is clear that nodes labelled with sequents which obey 1 or 2 are successful as then F has the property Ψ relative to $\mathcal{V}$, and similarly nodes labelled with sequents fulfilling $1'$ or $2'$ are not true. The remaining two conditions are analogues of termination of a finite length game play through repetition, and are pictured in figure 19. It is at this point that we distinguish in the proof theory

$$\begin{array}{ll@{\qquad}ll}
\vdots & & \vdots & \\
\vdots & U \overset{\text{def}}{=} \nu Z.\Phi & \vdots & U \overset{\text{def}}{=} \mu Z.\Phi \\
\vdots & & \vdots & \\
F \vdash U & & F \vdash U & \\
\vdots & & \vdots & \\
F \vdash U & & F \vdash U & \\
\textbf{Successful} & & \textbf{Unsuccessful} &
\end{array}$$

Fig. 19. Termination through repetition

between the two kinds of fixed points as they are not differentiated by the rules earlier.

Definition 1 A *successful* tableau for $E \vdash_{\mathcal{V}} \Phi$ is a finite proof tree whose root is labelled with $E \vdash_{\mathcal{V}} \Phi$, 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 conditions 1, 2 or 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 a path is the game play $(E_0, \Phi_0) \ldots (E_n, \Phi_n)$. In the case of a successful tableau, player II wins each of these plays. Moreover *all* the possible choices player I can make are present in the proof tree (up to renaming of constants). A successful tableau for $E \vdash \Phi$ is a concrete expression of player II's winning strategy for the game (E, Φ), and is therefore a proof that E has the temporal property 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, $\nu Z. ([-\texttt{tick}]\texttt{ff} \wedge \langle - \rangle \texttt{tt}) \wedge [-]Z$. Below is a proof that *Cl* has this property, where we omit the side conditions on the $[K]$, $\langle K \rangle$, σZ, and constant rules.

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

There are three kinds of successful leaf here. □

Example 2 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 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 20 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. □

Example 3 Let D and Φ be as in example 1 of section 3.4. The resulting successful tableau for $D \vdash \Phi$ is presented in figure 21. 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. □

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

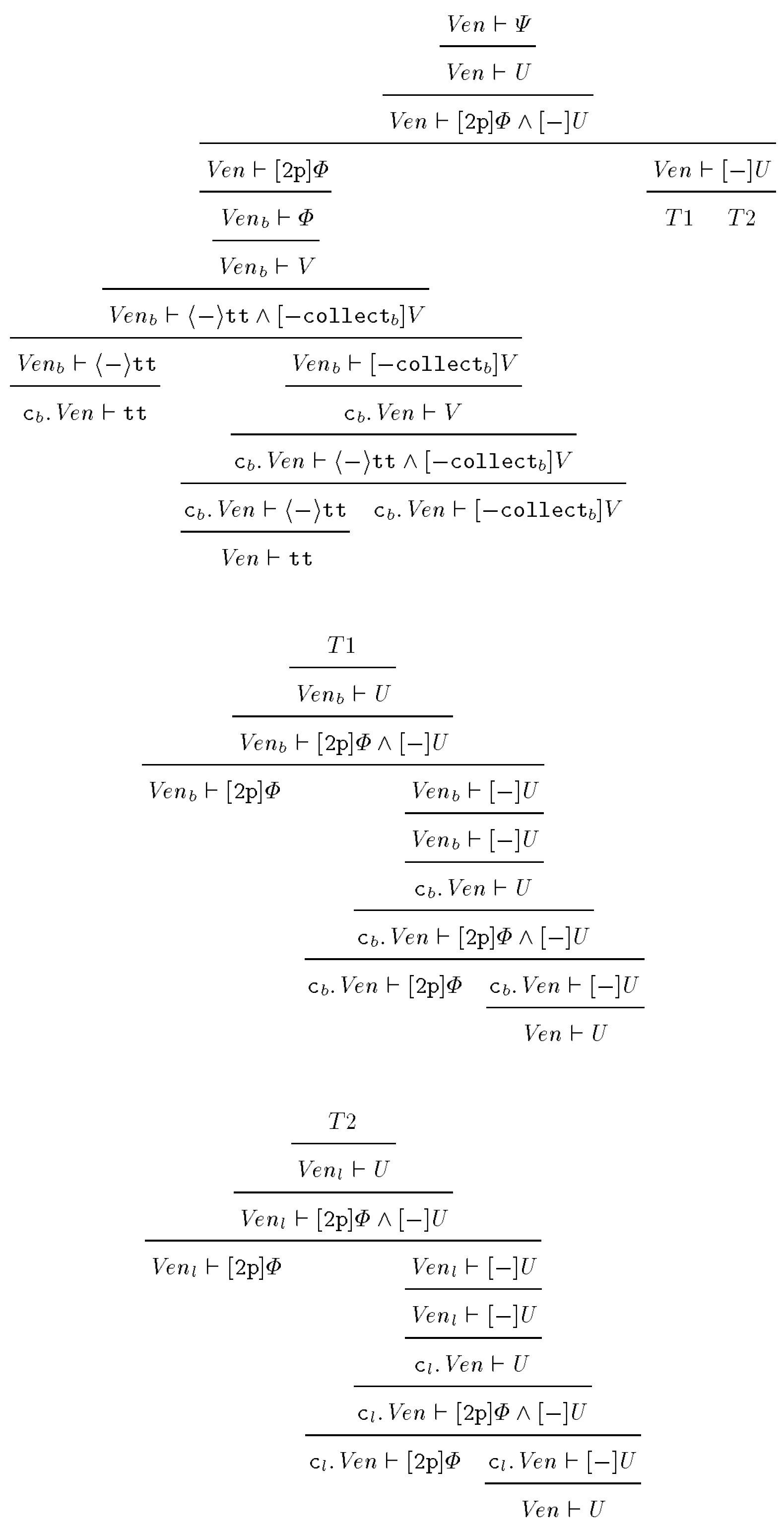


Fig. 20. A successful tableau for *Ven*

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

Fig. 21. A successful tableau for $D \vdash \Phi$

Proposition 2 *If E is a finite state process 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 show how this refinement affects the construction of tableaux.

Figure 16 contains the rules for the next move in a play whose initial part is $(E_0, \Phi_0) \ldots (E_j, \Phi_j)$. We now change the rules for introducing constants for fixed points, and divide each of them into two cases.

– if $\Phi_j = \nu Z.\Psi$ and player I has not previously introduced a constant $V \overset{\mathrm{def}}{=}$

$\nu Z.\Psi$ then player I chooses a *new* constant U and sets $U \stackrel{\rm def}{=} \nu Z.\Psi$: the process E_{j+1} is E_j and Φ_{j+1} is U.

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

This change means that constants are reintroduced as abbreviations for the same fixed point formula. All the other moves and who is responsible for them remain unchanged.

We also need to change the criteria for when a player wins a play. The winning conditions for the earlier games are given in figure 17. We retain the conditions 1, 2, $1'$ and $2'$: for instance if the configuration reached in a play is $(F, [K]\Psi)$ and the set $\{E : F \xrightarrow{a} E \text{ and } a \in K\}$ is empty then player II wins. The other conditions 3, 4, $3'$ and $4'$ need to be redefined because constants are reintroduced. An infinite length play may now contain more than one constant that recurs infinitely often.

The following definition provides a useful notion that will be used in the reformulated termination conditions.

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

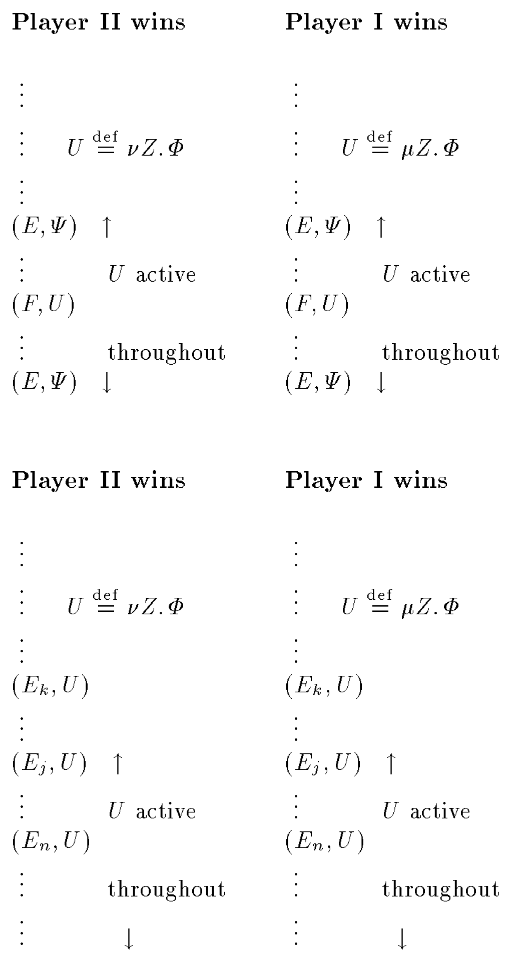
The discipline of introducing constants ensures that being active is well defined. If $U_1 \stackrel{\rm def}{=} \sigma Z_1.\Psi_1 \ldots U_n \stackrel{\rm def}{=} \sigma Z_n.\Psi_n$ is the sequence of declarations of distinct constants in order of their initial introduction then although U_i can be active in U_j when $i < j$ it is not possible for U_j to be active in U_i. Activity of a constant can be extended to finite or infinite length sequences of formulas: we say that U is *active* throughout $\Phi_0 \ldots \Phi_n \ldots$ if it is active in each Φ_i.

The following lemma governs the remaining winning conditions.

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 $\Phi_j \ldots \Phi_m \ldots$ for some $j \geq 0$.*

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 throughout this cycle and which occurs within it: if it abbreviates a maximal fixed point formula then player II wins and otherwise it abbreviates a least fixed point formula and player I wins. This replaces conditions 3 and $3'$ of figure 17. 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. This replaces conditions 4 and $4'$ of figure 17. These revised termination conditions are pictured in figure 22. Again the conditions 3 and $3'$ are redundant, and are only included because they guarantee that any play from a finite state process has finite length.



Player II wins
Player I wins
$U \overset{\text{def}}{=} \nu Z.\Phi$
$U \overset{\text{def}}{=} \mu Z.\Phi$
(E, Ψ)
(F, U)
(E, Ψ)
U active
throughout
Player II wins
Player I wins
$U \overset{\text{def}}{=} \nu Z.\Phi$
$U \overset{\text{def}}{=} \mu Z.\Phi$
(E_k, U)
(E_j, U)
(E_n, U)
U active
throughout

Fig. 22. Revised repeat winning conditions

A strategy is again a set of rules which dictates how a player should move, and it is winning if the player wins every play in which she uses it. For each game (E, Φ) one of the players has a winning strategy, which is again history free.

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

When E is a finite state process let $|E|$ be the number of processes in $\mathcal{P}(E)$, and let $|\Phi|$ be the size of Φ (the number of symbols within it). There are at most $|E| \times |\Phi|$ different configurations in any game play (up to renaming of constants). This means that any play of (E, Φ) has length at most $1 + (|E| \times |\Phi|)$.

Example 1 A case where game playing is shorter is example 3 of section 4.1. Let $D \xrightarrow{a} D'$, $D' \xrightarrow{a} D$ and $D' \xrightarrow{b} \mathbf{0}$, and let Ψ be $\mu Y.\, \nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee Y) \wedge Z)$. Player I's winning strategy for (D', Ψ) is the same as in that example. The play proceeds:

$$\begin{gathered}(D', \Psi)\, (D', U)\, (D', \nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge Z))\, (D', V) \\ (D', [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge V))\, (D, (\langle b\rangle \mathtt{tt} \vee U) \wedge V)\, (D, \langle b\rangle \mathtt{tt} \vee U)\end{gathered}$$

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:

$$\begin{gathered}(D, U)\, (D, \nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge Z))\, (D, V) \\ (D, [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge V))\, (D', (\langle b\rangle \mathtt{tt} \vee U) \wedge V)\, (D', V)\end{gathered}$$

There is now a repeat configuration (D', V). Consider the sequence of formulas in this game cycle, between and including these identical configurations:

$$\begin{gathered}V \;\; [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge V) \;\; (\langle b\rangle \mathtt{tt} \vee U) \wedge V \;\; \langle b\rangle \mathtt{tt} \vee U \;\; U \\ \nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge Z) \;\; [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge V) \;\; (\langle b\rangle \mathtt{tt} \vee U) \wedge V \;\; V\end{gathered}$$

The constant V is not active throughout this sequence (because it is not active in the formula $\nu Z.\, [a]((\langle b\rangle \mathtt{tt} \vee U) \wedge Z)$). However U is active throughout and also occurs there, 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 concretely represents a winning strategy for player II (when the game process is finite state). Given the refinements above to game playing we now redefine this tableau proof system. The only change to the rules 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:

$$\begin{array}{llll}
\vdots & U \overset{\text{def}}{=} \nu Z.\Phi & \vdots & U \overset{\text{def}}{=} \mu Z.\Phi \\
\vdots & & \vdots & \\
E \vdash \Psi & \uparrow & E \vdash \Psi & \uparrow \\
\vdots & \quad U \text{ active} & \vdots & \quad U \text{ active} \\
F \vdash U & & F \vdash U & \\
\vdots & \quad \text{throughout} & \vdots & \quad \text{throughout} \\
E \vdash \Psi & \downarrow & E \vdash \Psi & \downarrow \\
 & & & \\
\textbf{Successful} & & \textbf{Unsuccessful} &
\end{array}$$

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. As before a successful tableau for $E \vdash \Phi$ expresses player II's winning strategy for the game (E, Φ) and is therefore a proof that E has the property Φ.

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

Example 2 The tableau proof that D satisfies $\Phi = \nu Z.\, \mu Y.\, [a]((\langle b \rangle \mathtt{tt} \wedge Z) \vee Y)$ when $D \xrightarrow{a} D' \xrightarrow{a} D$ and $D' \xrightarrow{b} \mathbf{0}$ is given in figure 23. This proof is shorter than the previous proof in figure 21. 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. $\Box$

The tableau proof system presented here is again complete for finite state processes. This is again a consequence of the game characterization of satisfaction.

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

4.4 Game graphs and algorithms

Assume that E is a finite state process. The *game graph* for (E, Φ) relative to $\mathcal{V}$ is the graph representing all possible plays of the game (E, Φ), of the previous section, modulo a canonical means of choosing constants. The vertices are pairs (F, Ψ), configurations of a possible game play, and 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. Let $\mathcal{G}(E, \Phi)$ be the game graph for (E, Φ), and let $|\mathcal{G}(E, \Phi)|$ be its vertex size which we know, from the previous section, is no more than $|E| \times |\Phi|$. The complexity of property checking is NP $\cap$ co-NP. This follows from the observation that given a strategy for player II or player I it is straightforward to check in polynomial time whether or not it is winning: the technique in [29] is easily convertible into game playing.

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

Fig. 23. A successful tableau for $D \vdash \Phi$

We can easily ensure that game playing must proceed to infinity by adding extra moves when a player is stuck (and removing the redundant repeat termination conditions 3 and $3'$ of the previous section). The resulting game graph is then an alternating automaton: the and-vertices are the configurations from which player I proceeds and the or-vertices are those from which player II moves, and the acceptance condition is given in terms of active constants. See [10] which directly uses alternating automata for property checking.

An important open question is whether model checking modal mu-calculus formulas can be done in polynomial time (with respect to $|E| \times |\Phi|$). One direction for research is to provide a finer analysis of successful strategies, and to be able to describe optimizations of them. New insights may come from the relationship between the games developed here and other graph games where there are such descriptions.

The property checking game can be (easily) abstracted into the following graph game. A game is a graph with vertices $\{1, \ldots, n\}$ where each vertex i has two directed edges $i \longrightarrow j_1$ and $i \longrightarrow j_2$. Each vertex is labelled I or II. A play is an infinite path through the graph starting at vertex 1, and player I moves from vertices labelled I and player II from vertices labelled II. The winner of a play is determined by the label of the *least* vertex i which is traversed infinitely often: if i is labelled I then player I wins, and if it is labelled II then player II wins. A player wins the game if she has a winning strategy (which again is history free). For each game one of the players has a winning strategy. Notice that this graph game is described without mention of property checking.

Simple stochastic games [26] are graph games where the vertices are labelled I, II or A (average), and where there are two special vertices I-sink and II-sink (which have no outgoing edges). As above each I, II (and A) vertex has two outgoing edges. At an average vertex during a game play a coin is tossed to determine which of the two edges is traversed each having probability $\frac{1}{2}$. More generally one can assume that the two edges are labelled with probabilities of the form $\frac{p}{q}$ where $0 \leq p \leq q \leq 2^m$ for some m, as long as their sum is 1. A game play ends when a sink vertex is reached: player II wins if it is the II-sink, and player I otherwise. The decision question is whether the probability that player II wins is greater than $\frac{1}{2}$. It is not known whether this problem can be solved in polynomial time, although in [26] it is shown that it does belong to to NP∩co-NP. In [48] a "subexponential" ($2^{O(\sqrt{n})}$) algorithm is presented, which works by refining optimal strategies. A polynomial time algorithm for simple stochastic games would imply that extending space bounded alternating Turing machines with randomness does not increase the class of languages that they accept.

Mark Jerrum noted that there is a reduction from the graph game to the simple stochastic game. The idea is to add the two sink vertices, and an average vertex $i1$ for each vertex i for which there is an edge $j \longrightarrow i$ with $j \geq i$. Each such edge $j \longrightarrow i$ when $j \geq i$ is changed to $j \longrightarrow i1$. And the vertex $i1$ has an edge to i, and to I-sink if i is labelled I or to II-sink otherwise. With suitable rational probabilities on the edges, player II has a winning strategy for the graph game iff she has one for the simple stochastic game.

4.5 Generalizing tableaux

We have presented characterizations of when a process has a temporal property using games. We also developed tableau proof systems for verifying that processes have properties. However, a successful proof is only guaranteed in the case that a property holds of a finite state process. We would like a more general proof technique that also allows us to show properties of processes that are infinite state. Even in the finite state case a more general proof method may be useful, as a tableau proof may become too unwieldy because of its size.

There are various classes of infinite state system. Process definitions may involve explicit parameterization: examples include the counter Ct_i and register Reg_i of section 1.1, and the slot machine SM_n of section 1.2. Each instantiation of these processes is itself infinite state and contains the other family members within its transition graph. However the parameterization is very useful as it reveals straightforward structural similarities within these families of processes.

Another class of processes that is infinite state is entirely due to the presence of data values. The protocol of section 1.2 is a paradigm example. However there are different degrees of involvement of data within these processes, depending on the extent to which data determines future behaviour. At one extreme are examples such as the *Protocol* which pass data items through the system oblivious of their particular values. A number of authors has identified classes of processes which are in this sense data independent. At the other extreme are systems such as $T(i)$ of section 1.1 where future behaviour strongly depends on the value i. In between are systems such as the register where particular values are essential to change of state.

A third class of processes is infinite state independently of parameterization. An instance is the counter *Count* of section 1.5. Here the system evolves its structure as it performs actions. In certain cases processes that are infinite state in that they determine an infinite state transition graph are in fact bisimulation equivalent to a finite state process. A simple example is that $C \stackrel{\text{def}}{=} a.C \mid b.C$ is bisimilar to $C' \stackrel{\text{def}}{=} a.C' + b.C'$. Another interesting subclass of infinite state processes are those for which bisimulation equivalence is decidable. Two examples are the context free processes and the basic parallel processes [23, 22].

A final class of systems is also parameterized. However for each instance of the parameter the system is finite state. Two paradigm examples are the buffer Buff_n and the scheduler Sched_n, both from section 1.4. Although the techniques for verification of temporal properties apply to instances they do not apply to the general families. In such cases we would like to prove properties generally, to show for instance that for each $n > 1$ Sched_n is free from deadlock. The proof of this requires exposing structure that is common to this whole family.

In this section we present a simple generalization of satisfaction, and examine how it can be used to provide a tableau proof system. The full story of this proof system (presented with Julian Bradfield in [18]) continues into the next section.

A straightforward generalization of satisfaction is as a relation between a *set* of processes and a formula. We use the same relation $\models_{\mathcal{V}}$ for this extension. If $\mathcal{P}$ is a transition closed set with $\mathcal{E} \subseteq \mathcal{P}$ then

$$\boxed{\mathcal{E} \models_{\mathcal{V}} \Phi \;\text{ iff }\; \mathcal{E} \subseteq \|\Phi\|^{\mathcal{P}}_{\mathcal{V}}}$$

As before we write $\mathcal{E} \models \Phi$ when there is no valuation, or when the valuation can be understood from the context.

Example 1 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 uses the expected inductive definition of $\mathcal{E} \models \Phi$ (which is discussed below):

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

This proof is more direct than appealing to induction on process indices. □

Example 1 utilizes some obvious properties of satisfaction between sets of processes and formulas. Below are necessary and sufficient conditions for all the connectives using a little notation which we shall explain.

$$\boxed{\begin{array}{ll}\mathcal{E} \models_{\mathcal{V}} \Phi_1 \wedge \Phi_2 \text{ iff } \mathcal{E} \models_{\mathcal{V}} \Phi_1 \text{ and } \mathcal{E} \models_{\mathcal{V}} \Phi_2 & \\ \mathcal{E} \models_{\mathcal{V}} \Phi_1 \vee \Phi_2 \text{ iff } \exists \mathcal{E}_1, \mathcal{E}_2 .\ \mathcal{E}_1 \cup \mathcal{E}_2 = \mathcal{E} \text{ and } \mathcal{E}_1 \models_{\mathcal{V}} \Phi_1 \text{ and } \mathcal{E}_2 \models_{\mathcal{V}} \Phi_2 & \\ \mathcal{E} \models_{\mathcal{V}} [K]\Phi \quad \text{iff } K(\mathcal{E}) \models_{\mathcal{V}} \Phi & \\ \mathcal{E} \models_{\mathcal{V}} \langle K \rangle \Phi \quad \text{iff there is a choice function } f : \mathcal{E} \longrightarrow K(\mathcal{E}) \text{ and } f(\mathcal{E}) \models_{\mathcal{V}} \Phi & \\ \mathcal{E} \models_{\mathcal{V}} \sigma Z.\Phi \quad \text{iff } \mathcal{E} \models_{\mathcal{V}} \Phi\{\sigma Z.\Phi/Z\} & \end{array}}$$

The case of conjunction is the most straightforward. Disjunction is more complex. When $\mathcal{E} \models_{\mathcal{V}} \Phi_1 \vee \Phi_2$ the set $\mathcal{E}$ can be spit into two subsets $\mathcal{E}_1$ and $\mathcal{E}_2$ with $\mathcal{E}_i \models \Phi_i$: note that one of these sets could be empty[24]. For the modal cases we utilise notation. If $\mathcal{E}$ is a set of processes and K is a set of actions then $K(\mathcal{E})$ is the set $\{F : \exists E \in \mathcal{E}. \exists a \in K.\ E \xrightarrow{a} F\}$, which is the set of processes reachable by K transitions from members of $\mathcal{E}$. In example 1 $\{\texttt{up}\}(\{Ct_i : i \geq 0\}) = \{Ct_i : i \geq 1\}$. The set $\mathcal{E}$ has the property $[K]\Phi$ iff $K(\mathcal{E})$ satisfies Φ. For the diamond modality we appeal to functions. A function $f : \mathcal{E} \longrightarrow K(\mathcal{E})$ 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 $f(\mathcal{E}) = \{f(E) : E \in \mathcal{E}\}$. The set $\mathcal{E}$ satisfies $\langle K \rangle \Phi$ just in case there is a choice function $f : \mathcal{E} \longrightarrow K(\mathcal{E})$ where $f(\mathcal{E})$ satisfies Φ. In example 1 there is the function $f : \{Ct_i \,;\, i \geq 2\} \longrightarrow \{\texttt{down}\}(\{Ct_i \,;\, i \geq 2\})$ given by $f(Ct_{i+1}) = Ct_i$, which justifies one of the proof steps. This leaves the difficult cases of the fixed points. We shall make use of the principles developed in previous sections which appeal to games. Note however that the fixed point unfolding principle holds for the generalized satisfaction relation.

We wish to extend the tableau proof system of section 4.2 to encompass sets of processes having a property. Therefore we extend the basic sequent $E \vdash_{\mathcal{V}} \Phi$ to $\mathcal{E} \vdash_{\mathcal{V}} \Phi$, and 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. The tableau proof rules are presented in figure 24. As we are generalizing the proof system of section 4.2 new constants are introduced 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 show that all the processes in $\mathcal{E}$ have the property Φ relative to $\mathcal{V}$, one tries to achieve the goal $\mathcal{E} \vdash_{\mathcal{V}} \Phi$ by building a successful tableau. As before a successful

[24] By definition $\emptyset \models_{\mathcal{V}} \Phi$ for any Φ.

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

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

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

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

Fig. 24. Tableaux rules

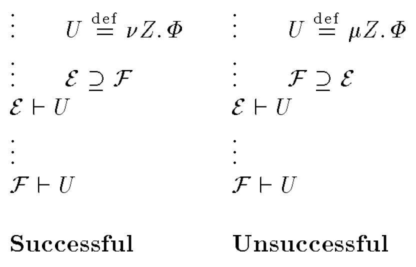
tableau is a finite proof tree whose root is labelled with this initial sequent, and where all the leaves are labelled by sequents that are successful. Sequents labelling the immediate successors of a node labelled $\mathcal{F} \vdash \Psi$ are determined by an application of one of the rules, either by Thin or by the rule for the main connective of Ψ. The crucial missing ingredient is when a node counts as a terminal node.

The definition of a leaf in a tableau is, as we shall see in the next section, underpinned by the game theoretic characterization of satisfaction. A tableau now captures a whole family of games. For each process in the set of processes on the left hand side of a sequent determines a play from it and the property on the right hand side. A node **n** labelled by the sequent $\mathcal{F} \vdash \Psi$ is terminal in the circumstances described in figure 25. Clearly a node labelled with a sequent

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 $U \overset{\text{def}}{=} \nu Z.\Phi$ and there is a node above **n** labelled $\mathcal{E} \vdash U$ with $\mathcal{E} \supseteq \mathcal{F}$	3.′ $\Psi = U$ and $U \overset{\text{def}}{=} \mu Z.\Phi$ and there is a node above **n** labelled $\mathcal{E} \vdash U$ with $\mathcal{F} \supseteq \mathcal{E}$

Fig. 25. Tableau terminals

fulfilling 1 or 2 is successful, and similarly any node labelled with 1′ or 2′ is not true. The other two conditions are generalizations of those for the proof system



$U \overset{\text{def}}{=} \nu Z.\Phi$
$\mathcal{E} \supseteq \mathcal{F}$
$\mathcal{E} \vdash U$
$\mathcal{F} \vdash U$
Successful
$U \overset{\text{def}}{=} \mu Z.\Phi$
$\mathcal{F} \supseteq \mathcal{E}$
$\mathcal{E} \vdash U$
$\mathcal{F} \vdash U$
Unsuccessful

Fig. 26. "Repeat" termination conditions

of section 4.2, and are pictured in figure 26. The justification for the success of condition 3 can be seen by considering any infinite length game play from a process in $\mathcal{E}$ with respect to the property U which cycles through the leaf $\mathcal{F} \vdash U$. As $\mathcal{F} \subseteq \mathcal{E}$ the play continues from this companion node. Such an infinite play must pass through U infinitely often, and is therefore a win for player II.

A successful tableau for $\mathcal{E} \vdash_{\mathcal{V}} \Phi$ is a finite proof tree all of whose leaves are successful. A successful tableau only contains true leaves.

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

However as the proof system stands, the converse is not true. A further termination condition is needed 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 write $E \vdash \Phi$ instead of $\{E\} \vdash \Phi$.

Example 2 It is not possible to show that Cnt has the property $\nu Z.\,\langle \texttt{up} \rangle Z$ using the tableau proof system of section 4.2, when Cnt is the infinite state process, $Cnt \overset{\text{def}}{=} \texttt{up}.(Cnt \mid \texttt{down}.\mathbf{0})$. There is a very simple proof within this more general proof system. Let Cnt_0 be Cnt and let Cnt_{i+1} be $Cnt_i \mid \texttt{down}.\mathbf{0}$ for any $i \geq 0$.

$$\begin{array}{c}
Cnt \vdash \nu Z.\,\langle \texttt{up} \rangle Z \\ \hline
\{Cnt_i \,:\, i \geq 0\} \vdash \nu Z.\,\langle \texttt{up} \rangle Z \\ \hline
\{Cnt_i \,:\, i \geq 0\} \vdash U \\ \hline
\{Cnt_i \,:\, i \geq 0\} \vdash \langle \texttt{up} \rangle U \\ \hline
\{Cnt_i \,:\, i \geq 1\} \vdash U
\end{array}$$

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}. □

Example 3 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 3.8, has the following tableau where the falsity of Q indicates that the slot machine owes money:

$$\dfrac{\{SM_n \,:\, n \geq 0\} \vdash \nu Z.Q \wedge [-]Z}{\dfrac{\mathcal{E} \vdash \nu Z.Q \wedge [-]Z}{\dfrac{\mathcal{E} \vdash U}{\dfrac{\mathcal{E} \vdash Q \wedge [-]U}{\mathcal{E} \vdash Q \qquad \dfrac{\mathcal{E} \vdash [-]U}{\mathcal{E} \vdash U}}}}}$$

Here the Thin rule is used to enlarge the set of slot machines to the set $\mathcal{E}$ which is $\mathcal{P}(SM_n)$. □

The proof system is also applicable to finite state examples, providing a much more succinct presentation of player II's winning strategy for a game.

Example 4 Recall the level crossing of figure 7. Its safety property is expressed as $\nu Z.([\overline{\texttt{tcross}}]\texttt{ff} \vee [\overline{\texttt{ccross}}]\texttt{ff}) \wedge [-]Z$. Let Φ be this formula. We employ the abbreviations in figure 8, and we let $\mathcal{E}$ be the full set $\{E_0, \ldots E_{11}\}$. Below is a successful tableau showing that the crossing has this property.

$$\dfrac{\mathit{Crossing} \vdash \Phi}{\dfrac{\mathcal{E} \vdash \Phi}{\dfrac{\mathcal{E} \vdash U}{\dfrac{\mathcal{E} \vdash ([\overline{\texttt{tcross}}]\texttt{ff} \vee [\overline{\texttt{ccross}}]\texttt{ff}) \wedge [-]U}{\dfrac{\mathcal{E} \vdash [\overline{\texttt{tcross}}]\texttt{ff} \vee [\overline{\texttt{ccross}}]\texttt{ff}}{\dfrac{\mathcal{E} - \{E_5, E_7\} \vdash [\overline{\texttt{tcross}}]\texttt{ff}}{\emptyset \vdash \texttt{ff}} \quad \dfrac{\mathcal{E} - \{E_4, E_6\} \vdash [\overline{\texttt{ccross}}]\texttt{ff}}{\emptyset \vdash \texttt{ff}}} \quad \dfrac{\mathcal{E} \vdash [-]U}{\mathcal{E} \vdash U}}}}}$$

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

Example 5 In example 2 of section 4.2 we noted how similar the two subtableaux $T1$ and $T2$ are. These can be amalgamated as follows (where the same abbreviations are used):

$$\dfrac{\{\,Ven_b,\, Ven_l\} \vdash U}{\dfrac{\{\,Ven_b,\, Ven_l\} \vdash [\mathsf{2p}]\Phi \wedge [-]U}{\dfrac{\{\,Ven_b,\, Ven_l\} \vdash [\mathsf{2p}]\Phi}{\emptyset \vdash \Phi} \qquad \dfrac{\{\,Ven_b,\, Ven_l\} \vdash [-]U}{\dfrac{\{\,Ven_b,\, Ven_l\} \vdash [-]U}{\dfrac{\{\mathtt{c}_b.\,Ven,\, \mathtt{c}_l.\,Ven\} \vdash U}{\dfrac{\{\mathtt{c}_b.\,Ven,\, \mathtt{c}_l.\,Ven\} \vdash [\mathsf{2p}]\Phi \wedge [-]U}{\dfrac{\{\mathtt{c}_b.\,Ven,\, \mathtt{c}_l.\,Ven\} \vdash [\mathsf{2p}]\Phi}{\emptyset \vdash \Phi} \quad \dfrac{\{\mathtt{c}_b.\,Ven,\, \mathtt{c}_l.\,Ven\} \vdash [-]U}{Ven \vdash U}}}}}}}$$

The terminals have changed slightly. □

4.6 Well foundedness

The proof system of the previous section is not complete. An example for which there is not a successful tableau is given by the following cell, $C \overset{\text{def}}{=} \mathtt{in}(x).B_x$ when $x : \mathbb{N}$ and $B_{n+1} \overset{\text{def}}{=} \mathtt{down}.B_n$. This cell has the property of eventual termination, $\mu Z.\,[-]Z$. The only possible tableau for $C \vdash \mu Z.\,[-]Z$ up to renaming the constant, and inessential applications of Thin is:

$$\dfrac{C \vdash \mu Z.\,[-]Z}{\dfrac{C \vdash U}{\dfrac{C \vdash [-]U}{\dfrac{(\mathbf{1})\;\; \{B_i \;:\; i \geq 0\} \vdash U}{\dfrac{(\mathbf{2})\; \{B_i \;:\; i \geq 0\} \vdash [-]U}{(\mathbf{3})\; \{B_i \;:\; i \geq 0\} \vdash U}}}}}$$

The final node (**3**) is terminal because of the node labelled (**1**) above it, and it is unsuccessful because U is a least fixed point constant. However any play of the game $(C, \mu Z.\,[-]Z)$ is won by player II. One solution to the problem is to permit induction on top of the current proof system by showing that each B_i has the property U. However we would like to avoid explicit induction principles. Instead we shall present criteria for success which captures player II's winning strategy. This requires 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 $\mathcal{F} \vdash U$ is also a terminal if it obeys the (almost repeat) condition of figure 27. This circumstance is very similar to condition 3 of the previous section except it is with respect

$$\begin{array}{ll} \vdots & U \overset{\mathrm{def}}{=} \mu Z.\Phi \\ \vdots & \mathcal{F} \subseteq \mathcal{E} \\ \mathcal{E} \vdash U & \\ \vdots & \\ \mathcal{F} \vdash U & \end{array}$$

Fig. 27. New terminal

to a least fixed point constant, and it is also similar to $3'$ except that the set of processes $\mathcal{F}$ at the leaf is a subset of $\mathcal{E}$. Not all nodes that obey this new condition are successful. The definition of success (taken from [18]) 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 $\mathbf{n}'$ is labelled $\mathcal{E}' \vdash \Phi'$.

$$\frac{\mathbf{n}: \quad \mathcal{E} \vdash \Phi}{\mathbf{n}': \quad \mathcal{E}' \vdash \Phi'}$$

A game play proceeding through (E, Φ) where $E \in \mathcal{E}$ can have as its next configuration (E', Φ') where $E' \in \mathcal{E}'$ provided the rule applied at $\mathbf{n}$ is not Thin. Which possible processes $E' \in \mathcal{E}'$ can be in this next configuration depend on the structure of Φ. This motivates the following notion. 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.

All the possibilities are covered here. An example is that each B_i at node (**2**) in the tableau earlier is a dependant of the same B_i at node (**1**), and each B_i at (**1**) is a dependant of C at the node directly above it.

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 Assume that node $\mathbf{n_k}$ is a μ-terminal and node $\mathbf{n_1}$ is its companion. A *trail* from process E_1 at $\mathbf{n_1}$ to E_k at $\mathbf{n_k}$ 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

1. E_{i+1} at $\mathbf{n_{i+1}}$ is a dependant of E_i at $\mathbf{n_i}$, or

2. $\mathbf{n_i}$ 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 $j : 1 \leq j \leq i$, and $\mathbf{n_{i+1}} = \mathbf{n_j}$ and E_{i+1} at $\mathbf{n}'$ is a dependant of E_i at $\mathbf{n_i}$.

$((\mathbf{1}), B_2)\,((\mathbf{2}), B_2)\,((\mathbf{3}), B_1)$ is a simple trail from B_2 at $(\mathbf{1})$ to B_1 at $(\mathbf{3})$ in the tableau earlier. In this case B_2 at $(\mathbf{2})$ is a dependant of B_2 at $(\mathbf{1})$, and B_1 at $(\mathbf{3})$ is a dependant of B_2 at $(\mathbf{2})$. Condition 2 of Definition 1 is needed to take account of the possibility of embedded fixed points as pictured in figure 28. A trail from $(\mathbf{n_1}, E_1)$ to $(\mathbf{n_k}, E_k)$ may pass through node $\mathbf{n_j}$ repeatedly before

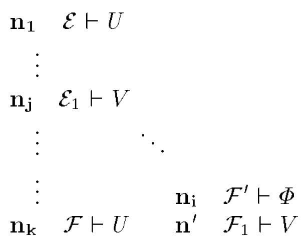

$\mathbf{n_1}$ $\mathcal{E} \vdash U$
$\mathbf{n_j}$ $\mathcal{E}_1 \vdash V$
$\mathbf{n_i}$ $\mathcal{F}' \vdash \Phi$
$\mathbf{n_k}$ $\mathcal{F} \vdash U$
$\mathbf{n}'$ $\mathcal{F}_1 \vdash V$

Fig. 28. Embedded terminals: $\mathcal{F} \subseteq \mathcal{E}$ and $\mathcal{F}_1 \subseteq \mathcal{E}_1$.

continuing to $\mathbf{n_k}$. In this case $\mathbf{n_k}$ is a μ-terminal but $\mathbf{n}'$ may be either a μ or a ν-terminal. In fact node $\mathbf{n_j}$ here could be $\mathbf{n_1}$ with $\mathbf{n_k}$ and $\mathbf{n}'$ both sharing the same companion. This can be further iterated when there is another σ-companion along the path from $\mathbf{n_j}$ to $\mathbf{n_i}$ and a further leaf $\mathbf{n}''$, and so on. Note that there is an intimate relationship between the sequence of processes in a trail and a sequence of processes in part of a game play from from the companion node to the terminal.

Each companion node $\mathbf{n}$ of a μ-terminal induces a relation $\rhd_{\mathbf{n}}$:

Definition 2 $E \rhd_{\mathbf{n}} F$ if there is a trail from E at $\mathbf{n}$ to F at a μ-terminal whose companion is $\mathbf{n}$.

We now come to the definition of a successful μ-terminal.

Definition 3 A μ-terminal whose companion node is $\mathbf{n}$ is *successful* if there is no infinite "descending" chain $E_0 \rhd_{\mathbf{n}} E_1 \rhd_{\mathbf{n}} \ldots$.

Success means that the relation $\lhd_{\mathbf{n}}$ induced by the companion of a μ-terminal is well-founded. This precludes the possibility of an infinite game play asssociated with a tableau which cycles through the node $\mathbf{n}$ infinitely often.

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 state) processes. Again the result is proved using the game characterization of satisfaction.

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

Example 1 The tableau at the start of this section is successful. The only trail from B_{i+1} at $(\mathbf{1})$ to $(\mathbf{3})$ is $((\mathbf{1}), B_{i+1})\,((\mathbf{2}), B_{i+1})\,((\mathbf{3}), B_i)$, and therefore $B_{i+1} \rhd_{(\mathbf{1})} B_i$.

Suppose we amend the definition of B_{i+1} to $B_{i+1} \overset{\text{def}}{=} \mathtt{down}.B_i + \mathtt{up}.B_{i+2}$. Each B_{i+1} has the extra capability for performing $\mathtt{up}$. An attempt to prove that C eventually terminates yields the same tableau as at the beginning of the section. However this tableau is now unsuccessful. There are two two trails from B_{i+1} at $(\mathbf{1})$ to $(\mathbf{3})$: $((\mathbf{1}), B_{i+1})\,((\mathbf{2}), B_{i+1})\,((\mathbf{3}), B_i)$ and $((\mathbf{1}), B_{i+1})\,((\mathbf{2}), B_{i+1})\,((\mathbf{3}), B_{i+2})$. Hence, both $B_{i+1} \rhd_{(\mathbf{1})} B_i$ and $B_{i+1} \rhd_{(\mathbf{1})} B_{i+2}$. Consequently there is a variety of infinite decreasing sequences from B_{i+1} such as $B_{i+1} \rhd_{(\mathbf{1})} B_{i+2} \rhd_{(\mathbf{1})} B_{i+1} \ldots$. $\square$

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 3.8, 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 $\mathtt{green}$ and R holds in any state where *Road* can by itself perform $\mathtt{up}$. 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 29.

In this tableau there is one μ-terminal, labelled $(\mathbf{1})$ whose companion is $(\mathbf{c})$. The relation $\rhd_{\mathbf{c}}$ is well founded as we only have: $E_1 \rhd_{\mathbf{c}} E_4$, $E_4 \rhd_{\mathbf{c}} E_6$ and $E_3 \rhd_{\mathbf{c}} E_6$. Therefore the tableau is successful. $\square$

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:

$$\begin{array}{c}
\{SM_n \ : \ n \geq 0\} \vdash \nu Y.\mu Z.\langle\overline{\mathtt{win}}(10^6)\rangle Y \vee \langle -\rangle Z \\ \hline
\mathcal{E} \vdash \nu Y.\mu Z.\langle\overline{\mathtt{win}}(10^6)\rangle Y \vee \langle -\rangle Z \\ \hline
\mathcal{E} \vdash U \\ \hline
\mathcal{E} \vdash \mu Z.\langle\overline{\mathtt{win}}(10^6)\rangle U \vee \langle -\rangle Z \\ \hline
(\mathbf{1})\ \mathcal{E} \vdash V \\ \hline
(\mathbf{2})\ \mathcal{E} \vdash \langle\overline{\mathtt{win}}(10^6)\rangle U \vee \langle -\rangle V \\ \hline
\begin{array}{cc}
\mathcal{E}_1 \vdash \langle\overline{\mathtt{win}}(10^6)\rangle U & (\mathbf{3})\ \mathcal{E}_2 \vdash \langle -\rangle V \\ \hline
\mathcal{E}_1' \vdash U & (\mathbf{4})\ \mathcal{E}_2' \vdash V
\end{array}
\end{array}$$

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

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

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

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

Fig. 29. Liveness proof for the *Crossing*

$\mathcal{E}$ 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 $\overline{\mathtt{win}}(10^6)$ 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 a 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 $\overline{\mathtt{release}}(10^6)$. 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. $\square$

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

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

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

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

The problem is that we dont know if the relation induced by the companion of this leaf is well-founded. $\square$

5 Concluding Comments

In previous sections we used modal logic and modal mu-calculus for analysing properties of processes. We also noted the close relationship between bisimilarity and having the same properties. Some of the techniques mentioned, especially in the case of finite state processes, are implemented in the Edinburgh Concurrency Workbench[25]. Another tool is the infinite state model checker for Petri nets based on tableaux, described in [16].

An important topic which we have not discussed 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.

[25] Which is freely available by emailing Perdita.Stevens@dcs.ed.ac.uk, or from the WWW, http://www.dcs.ed.ac.uk/packages/cwb/index.html.

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

This is justified because as we saw in section 3.7 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 transition 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* $P \overset{\text{def}}{=} E$ *and* $E \models \Phi$ *then* $P \models \Phi$.

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, $\mathtt{tt}$, or $\mathtt{ff}$. 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 [20].

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 [63, 45, 2, 3].

References

1. Abramsky, S., (1995). Interaction categories, This Volume.
2. Andersen, H., and Winskel, G. (1992). Compositional checking of satisfaction. *Formal Methods in System Design*, 1.
3. Andersen, H., Stirling, C., and Winskel, G. (1994). A compositional proof system for the modal mu-calculus. *Procs 9th IEEE Symposium on Logic in Computer Science*, 144-153.
4. Austry, D., and Boudol, G. (1984). Algebra de processus et synchronisation. *Theoretical Computer Science*, 30, 90-131.
5. Baeten, J., Bergstra, J., and Klop, J. (1987). Decidability of bisimulation equivalence for processes generating context-free languages. *Lecture Notes in Computer Science*, 259, 94-113.
6. Baeten, J, and Weijland, W. (1990). *Process Algebra.* Cambridge University Press.
7. Baeten, J., Bergstra, J., Hoare, C., Milner, R., Parrow, J., and de Simone, R. (1992). The variety of process algebra. Manuscript.
8. De Bakker, J. (1980). *Mathematical Theory of Program Correctness*, Prentice-Hall.
9. 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.
10. Bernholtz, O., Vardi, M. and Wolper, P. (1994). An automata-theoretic approach to branching-time model checking. *Lecture Notes in Computer Science*, 818, 142-155.
11. Bergstra, J. and Klop, J. (1989). Process theory based on bisimulation semantics. *Lecture Notes in Computer Science*, 354, 50-122.
12. van Benthem, J. (1984). Correspondence theory. In *Handbook of Philosophical Logic*, Vol. II, ed. Gabbay, D. and Guenthner, F., 167-248, Reidel.
13. Bloom, B., Istrail, S., and Meyer A. (1988). Bisimulation cant be traced. In *15th Annual Symposium on the Principles of Programming Languages*, 229-239.
14. Boudol, G. (1985). Notes on algebraic calculi of processes, in *Logics and Models of Concurrent Systems*, Springer.
15. Bradfield, J. (1992). *Verifying Temporal Properties of Systems.* Birkhauser.
16. Bradfield, J. (1993). A proof assistant for symbolic model checking. *Lecture Notes in Computer Science*, 663, 316-329.
17. Bradfield, J. and Stirling, C. (1990). Verifying temporal properties of processes. *Lecture Notes in Computer Science*, 458, 115-125.
18. Bradfield, J. and Stirling, C. (1992). Local model checking for infinite state spaces. *Theoretical Computer Science*, 96, 157-174.
19. Browne, M., Clarke, E., and Grumberg, O. (1988). Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59, 115-131.
20. Bruns, G. (1993). A practical technique for process abstraction. *Lecture Notes in Computer Science*, 715, 37-49.
21. Camilleri, J., and Winskel, G. (1991). CCS with priority. *Procs 6th IEEE Symposium on Logic in Computer Science*, 246-255.
22. Christensen, S., Hirshfeld, Y., and Moller, F. (1993). Bisimulation is decidable for all basic parallel processes. *Lecture Notes in Computer Science*, 715, 143-157.

23. Christensen, S., Hüttel, H., and Stirling, C. (1992). Bisimulation equivalence is decidable for all context-free processes. *Lecture Notes in Computer Science*, 630, 138-147.
24. Cleaveland, R., and Hennessy, M. (1988). Priorities in process algebra. *Proc. 3rd IEEE Symposium on Logic in Computer Science*, 193-202.
25. Cleaveland, R, Parrow, J, and Steffen, B. (1989). The concurrency workbench. *Lecture Notes in Computer Science*, 407, 24-37.
26. Condon, A. (1992). The complexity of stochastic games. *Information and Computation*, 96, 203-224.
27. De Nicola, R. and Vaandrager, V. (1990). Three logics for branching bisimulation. *Proc. 5th IEEE Symposium on Logic in Computer Science*, 118-129.
28. Emerson, E, and Clarke, E. (1980). Characterizing correctness properties of parallel programs using fixpoints. *Lecture Notes in Computer Science*, 85, 169-181.
29. Emerson, E., and Jutla, C. (1991). Tree automata, mu-calculus and determinacy. In *Proc. 32nd IEEE Foundations of Computer Science.*
30. Emerson, E, and Lei, C. (1986). Efficient model checking in fragments of the propositional mu-calculus. In *Proc. 1st IEEE Symposium on Logic in Computer Science*, 267-278.
31. Emerson, E, and Srinivasan, J. (1989). Branching time temporal logic. *Lecture Notes in Computer Science*, 354, 123-284.
32. van Glabbeek, J. (1990). The linear time–branching time spectrum. *Lecture Notes in Computer Science*, 458, 278-297.
33. van Glabbeek, J. F., and Weijland, W.P. (1989). Branching time and abstraction in bisimulation semantics. *Information Processing Letters*, 89, 613-618.
34. Groote, J. (1993). Transition system specifications with negative premises. *Theoretical Computer Science*, 118, 263-299.
35. Groote, J. and Vaandrager, F. (1989). Structured operational semantics and bisimulation as a congruence. *Lecture Notes in Computer Science*, 372, 423-438.
36. Hennessy, M. (1988). *An Algebraic Theory of Processes.* MIT Press.
37. Hennessy, M. and Ingolfsdottir. (1990). A theory of communicating processes with value-passing. *Lecture Notes in Computer Science*, 443, 209-220.
38. Hennessy, M. and Milner, R. (1980). On observing nondeterminism and concurrency. *Lecture Notes in Computer Science*, 85, 295-309.
39. Hennessy, M. and Milner, R. (1985). Algebraic laws for nondeterminism and concurrency. *Journal of Association of Computer Machinery*, 32, 137-162.
40. Hoare, C. (1985). *Communicating Sequential Processes.* Prentice Hall.
41. Kannellakis, P. and Smolka, S. (1990). CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86, 43-68.
42. Kozen, D. (1983). Results on the propositional mu-calculus. *Theoretical Computer Science*, 27, 333-354.
43. Lamport, L. (1983) Specifying concurrent program modules. *ACM Transactions of Programming Language Systems*, 6, 190-222.
44. Larsen, K. (1990). Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72, 265-288.
45. Larsen, K. (1990). Ideal specification formalism. *Lecture Notes in Computer Science*, 458, 33-56.
46. Larsen, K. and Skou. (1989). Bisimulation through probabilistic testing. In *16th Annual ACM Symposium on Principles of Programming Languages.*

47. Long, D., Browne, A., Clarke, E., Jha, S., and Marrero, W. (1994). An improved algorithm for the evaluation of fixpoint expressions. *Lecture Notes in Computer Science*, 818.
48. Ludwig, W. (1995). A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117, 151-155.
49. Manna, Z, and Pnueli, A. (1991). *The Temporal Logic of Reactive and Concurrent Systems.* Springer.
50. Milner, R. (1980). *A Calculus of Communicating Systems.* Lecture Notes in Computer Science, 92.
51. Milner, R. (1983). Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25, 267-310.
52. Milner, R. (1989). *Communication and Concurrency.* Prentice Hall.
53. Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, Parts I and II, *Information and Computation*, 100, 1-77.
54. Moller, F. and Tofts, C. (1990). A temporal calculus of communicating processes. *Lecture Notes in Computer Science*, 458, 401-415.
55. Nicollin, X. and Sifakis, J. (1992). An overview and synthesis on timed process algebras. *Lecture Notes in Computer Science*, 575, 376-398.
56. Park, D. (1969). Fixpoint induction and proofs of program properties. *Machine Intelligence*, 5, 59-78, Edinburgh University Press
57. Park, D. (1981). Concurrency and automata on infinite sequences. *Lecture Notes in Computer Science*, 154, 561-572.
58. Parrow, J. (1988). Verifying a CSMA/CD-Protocol with CCS. In *Protocol Specification, Testing, and Verification VIII*, 373-384. North-Holland.
59. Plotkin, G. (1981). A structural approach to operational semantics. *Technical Report*, DAIMI FN-19, Aarhus University.
60. Pratt, V. (1982). A decidable mu-calculus, *22nd IEEE Symposium on Foundations of Computer Science*, 421-427.
61. Simone, R. de (1985). Higher-level synchronizing devices in Meije-SCCS. *Theoretical Computer Science*, 37, 245-267.
62. Sistla, P., Clarke, E., Francez, N. and Meyer, A. (1984). Can message buffers be axiomatized in linear temporal logic? *Information and Control*, 68, 88-112.
63. Stirling, C. (1987). Modal logics for communicating systems, *Theoretical Computer Science*, 49, 311-347.
64. Stirling, C. (1992) Modal and temporal logics. In *Handbook of Logic in Computer Science* Vol. 2, ed. Abramsky, S, Gabbay, D, and Maibaum, T., 477-563, Oxford University Press.
65. Stirling, C. (1995). Local model checking games. *Lecture Notes in Computer Science*, 962, 1-11.
66. Stirling, C. and Walker, D. (1991). Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89, 161-177.
67. Streett, R. and Emerson, E. (1989). An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81, 249-264.
68. Taubner, D. (1989). *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets.* Lecture Notes in Computer Science, 369.
69. Walker, D. (1987). Introduction to a calculus of communicating systems. Technical Report ECS-LFCS-87-22, Dept. of Computer Science, Edinburgh University.
70. Walker, D. (1989). Automated analysis of mutual exclusion algorithms using CCS. *Formal Aspects of Computing*, 1, 273-292.

71. Walker, D. (1990). Bisimulations and divergence. *Information and Computation*, 85, 202-241.
72. Winskel, G. (1988). A category of labelled Petri Nets and compositional proof system. *Procs 3rd IEEE Symposium on Logic in Computer Science*, 142-153.
73. Wolper, P. (1983). Temporal logic can be more expressive. *Information and Control*, 56, 72-99.