SOFTWARE VERIFICATION RESEARCH CENTRE

SCHOOL OF INFORMATION TECHNOLOGY

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 00-10

Property Verification within a Process Algebra Framework

Antonio Cerone George Milne

April 2000

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533
http://svrc.it.uq.edu.au

In Dan Ionescu and Aurel Cornell, editors, *Real-Time Systems: Modeling, Design, and Applications*, pp. 153-174, World Scientific, 2000.

Property Verification within a Process Algebra Framework

Antonio Cerone George Milne

1 Introduction

In the last fifteen years process algebras have been succesfully used to model concurrent systems and to verify equivalence between the specification and the implementation of a system. More recently the combination of process algebra and temporal logic through the technique of *model checking* [7] has allowed the development of tools for the automatic verification of properties of systems. The most recent tools are based on a symbolic representation of the state space (*symbolic model checking*) [12] and allow the verification of systems with a large number of states.

Since such tools involve two different formalisms, the process algebra and the logic, their use is often very hard for non expert users. The critical step is the specification within the logic of the properties of a system that is modelled within the process algebra. Properties based on events performed by processes have to be thought again in terms of logical formulas and this procedure can easily generate errors.

In this paper we present a methodology for the automatic verification of concurrent systems, which is based on the characterization of temporal properties within *Circal* [13, 15], a process algebra in which processes can be guarded by sets of simultaneous actions and such actions can be shared over an arbitrary number of processes.

We introduce *SAUB* [6], a branching time temporal logic whose temporal operators are based on actions rather than on states. Several modal and temporal logics based on actions [8, 9, 16] have been developed to describe properties of processes. With respect to the previous logics, SAUB has the additional feature of expressing also simultaneity between actions and this makes it suitable to describe properties of Circal processes.

The distinctive features of Circal support *constraint-based* modelling [17, 13]: the behaviour of a process may be constrained simply by composing

it with another process, which represents the constraints. This modelling technique is the basis for verifying properties within the process algebra. A property of a process can be characterized in terms of another process so that one of the two processes constrains the other only when the property does not hold. This approach can be applied when the property to be tested is characterized in terms of a well-known model that satisfies it (*model-based* characterization) and is very useful in the verification of safety properties.

A more sophisticated approach leads to a characterization of general SAUB formulas in terms of Circal processes (*formula-based* characterization). A property can be characterized within Circal through a translation of the SAUB formula that describes it into a set of processes and into the rules to compose these processes with one another. The two processes obtained by the application of the rules are then tested for equivalence.

In both approaches the property verification is reduced to an equivalence between processes that can be automatically verified within the *Circal System*, the mechanisation of the Circal process algebra. This methodology has been successfully applied to the specification and automatic verification of an audio control protocol developed by Philips [1, 2] and to handshaking control circuits of asynchronous micropipelined processors [4, 5].

2 The Circal Process Algebra

In this section we give a brief description of the Circal operators and their semantics and refer the reader to the book by Milne [13] and the paper by Moller [15] for further explanations. To describe and analyze a system, the user works with the language XCircal, the extention of the Circal process algebra that is automated by the Circal System [13].

The syntax of Circal processes is summarized by the following BNF expressions, where P is a process, D a process definition, $\mathcal{A}$ is the set of possible actions, $m \subseteq \mathcal{A}$, $a, b \in \mathcal{A}$ and I is a process variable:

$$\begin{array}{lcl} P & ::= & /\backslash \mid mP \mid P + P \mid P \,\&\, P \mid I \mid P * P \mid P - m \mid P\,[a/b] \\ D & ::= & I <- P \end{array}$$

2.1 Informal Semantics

Each Circal process has associated with it a *sort*, which specifies the set of actions (ports) through which it may interact with other processes. Every sort will be a non empty subset of $\mathcal{A}$, the collection of all available actions. The set of processes of sort L is denoted by $\mathcal{P}_L$. The set of all processes

is $\mathcal{P} \stackrel{\text{def}}{=} \bigcup_{L \subseteq \mathcal{A}} \mathcal{P}_L$. If some action $a \in \mathcal{A}$ is in the sort of some interacting process, then a communication (synchronization) cannot occur via the port a unless the process in question allows it. A synchronization event involving any port $a \in \mathcal{A}$ not in the sort of an interacting process may occur without regard to the process in question. Sets of actions occur simultaneously in the communication and may be shared over arbitrary numbers of processes. The rôle of each Circal operator can be described as follows:

Termination. $/\backslash$ is a constant representing a process which can participate in no communication. This is a process that has terminated or deadlocked. There is actually a whole family $\{/\backslash_L\}_{L \subseteq \mathcal{A}}$ of such constants.

Guarding. Single or simultaneous guards are permitted. For example

$$P \; <\!- \; (a \; b) \; P'$$

represents the process P which will perform a and b simultaneously, and then evolve into P'. The fact that processes may be guarded by sets of simultaneously occurring actions is a key feature of Circal which greatly enriches the modelling potential of the algebra in contrast to process algebras such as CCS [14] and CSP [10] which only permit a single action to occur at one computation instant. An example of a single guard is

$$P \; <\!- \; a \; P'$$

in which process P performs the single action a and evolves to P'. In this case a is a shorthand for (a).

Definition. A name can be given to a Circal process with the definition operator. Recursive process definitions are permitted; for example

$$P \; <\!- \; (a \; b) \; P$$

is interpreted as a process that continuously repeats the simultaneous actions $(a \; b)$.

Choice. The term $P + Q$ represents the process which can perform either the actions of the subterm P or the actions of the subterm Q. The choice is decided by the environment in which the process is executed. For example, the following process can perform a or b or c

$$P \; <\!- \; a \; P \; + \; b \; P \; + \; c \; P'$$

The choice is made depending on the action available from the environment.

Non-determinism. The term $P\&Q$ represents the process which can perform either the actions of the subterm P or the actions of the subterm Q. The computation path is decided autonomously by the process itself without any influence from its environment. For example, the following process can perform a or b or c

$$P \; <\!\!- \; a\,P \;\&\; b\,P \;\&\; c\,P'$$

independently of which actions are supplied by the environment. If P chooses to perform a, and a is not supplied by the environment, then P terminates.

Concurrent Composition. Given processes P and Q, the term $P * Q$ represents the process which can perform the actions of the subterms P and Q together (*composition*). Any synchronization which can be made between two terms, due to some atomic action being common to the sorts of both subterms, must be made, otherwise the actions of the subterms may occur asynchronously.

Abstraction. The term $P - a$ represents the process that is identical to P except that the port a is invisible to the environment. We will use $P - a\,b\,c$ as a shorthand for $((P - a) - b) - c$.

Relabelling. The term $P[b/a]$ is the new version of P in which the action b replaces a wherever a occurs in P.

2.2 Formal Semantics

The formal semantics of Circal [15] is given in terms of transition systems labelled with set of actions.

Definition 2.1

A *Labelled Transition System* (LTS) is a structure $\mathcal{T} = \langle \mathcal{S}, \mathcal{A}, \rightarrow, s_0 \rangle$ where

- $\mathcal{S}$ is a finite set of *states*;
- $\mathcal{A}$ is a finite, non empty set of *actions*;
- $\rightarrow \subseteq \mathcal{S} \times 2^{\mathcal{A}} \times \mathcal{S}$ is a *transition relation*, whose generic element (s_1, μ, s_2) is usually written as $s_1 \stackrel{\mu}{\rightarrow} s_2$.
- s_0 is called the *initial state*

A *generalized transition* $p \stackrel{\mu}{\Rightarrow}_{\mathcal{T}} q$ is defined by

$$p \stackrel{\mu}{\Rightarrow}_{\mathcal{T}} q \quad \Longleftrightarrow \quad p \stackrel{\emptyset}{\rightarrow}* p' \stackrel{\mu}{\rightarrow} p'' \stackrel{\emptyset}{\rightarrow}* q \quad \text{for some } p', p''$$

where $\stackrel{\emptyset}{\rightarrow}*$ is the reflexive and transitive closure of $\stackrel{\emptyset}{\rightarrow}$. A path on $\mathcal{T}$ is an infinite sequence of states $s_1 s_2 s_3 ...$ such that for each $i \geq 1$ there exists $\mu \in 2^{\mathcal{A}}$ such that $s_i \stackrel{\mu}{\rightarrow} s_{i+1}$. The set of paths on $\mathcal{T}$ starting from $s \in \mathcal{S}$ is denoted by $\Pi_{\mathcal{T}}(s)$.
For each $s \in \mathcal{S}$ the *successor set* of s is

$$\Theta_{\mathcal{T}}(s) \stackrel{\text{def}}{=} \{\mu \in 2^{\mathcal{A}} \mid \mu \neq \emptyset \quad \text{and} \quad s \stackrel{\mu}{\Rightarrow}_{\mathcal{T}} s' \quad \text{for some } s' \in \mathcal{S}\}.$$

Definition 2.2

Let $\mathcal{T}_1 = \langle \mathcal{S}_1, \mathcal{A}_1, \rightarrow_1, s_1 \rangle$ and $\mathcal{T}_2 = \langle \mathcal{S}_2, \mathcal{A}_2, \rightarrow_2, s_2 \rangle$ be two LTSs. Then

- $\mathcal{T}_1 \sqsubseteq_{\text{may}} \mathcal{T}_2$ if and only if for each $\mu \in 2^{\mathcal{A}}$, for each $p_1 \in \mathcal{S}_1$, $s_1 \stackrel{\mu}{\Rightarrow}_{\mathcal{T}_1} p_1$ implies $s_2 \stackrel{\mu}{\Rightarrow}_{\mathcal{T}_2} p_2$ for some $p_2 \in \mathcal{S}_2$;
- $\mathcal{T}_1 \sqsubseteq_{\text{must}} \mathcal{T}_2$ if and only if for each $\mu \in 2^{\mathcal{A}}$, for each $p_2 \in \mathcal{S}_2$, $s_2 \stackrel{\mu}{\Rightarrow}_{\mathcal{T}_2} p_2$ implies $s_1 \stackrel{\mu}{\Rightarrow}_{\mathcal{T}_1} p_1$ for some $p_1 \in \mathcal{S}_1$ such that $\Theta_{\mathcal{T}_1}(p_1) \subseteq \Theta_{\mathcal{T}_2}(p_2)$;
- $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ if and only if $\mathcal{T}_1 \sqsubseteq_{\text{may}} \mathcal{T}_2$ and $\mathcal{T}_1 \sqsubseteq_{\text{must}} \mathcal{T}_2$.

Definition 2.3

Let $\mathcal{A}$ the set of all available actions. The semantics of a Circal program $R \in \mathcal{P}_L$ is the labelled transition system

$$\mathcal{B}(R) = \langle \mathcal{S}, \mathcal{A}, \rightarrow, R \rangle$$

such that $\rightarrow$ is the least relation that satisfies the following conditions:

- $P \rightarrow P$ and $\mu P \stackrel{\mu}{\rightarrow} P$;
- if $P \rightarrow P'$, then $P + Q \rightarrow P' + Q$ and $Q + P \rightarrow Q + P'$;
- if $P \stackrel{\mu}{\rightarrow} P'$ for $\mu \neq \emptyset$, then $P + Q \stackrel{\mu}{\rightarrow} P'$ and $Q + P \stackrel{\mu}{\rightarrow} P'$;
- $P \& Q \rightarrow P$ and $P \& Q \rightarrow Q$;
- if $P \in \mathcal{P}_{L_1}$, $Q \in \mathcal{P}_{L_2}$, $P \stackrel{\mu}{\rightarrow} P'$, $Q \stackrel{\nu}{\rightarrow} Q'$ and $\mu \cap L_2 = \nu \cap L_1$, then $P * Q \stackrel{\mu \cup \nu}{\rightarrow} P' * Q'$;
- for any $L \subseteq \mathcal{A}$ if $P \stackrel{\mu}{\rightarrow} P'$, then $P - L \stackrel{\mu \setminus L}{\rightarrow} P'$.

Definition 2.4

The *behavioural inclusion* of $P \in \mathcal{P}_L$ in $Q \in \mathcal{P}_M$ is defined as follows.

$$P \sqsubseteq Q \quad \Longleftrightarrow \quad \mathcal{B}(P) \sqsubseteq \mathcal{B}(Q).$$

Definition 2.5

The *equivalence* of $P \in \mathcal{P}_L$ and $Q \in \mathcal{P}_M$ is defined as follows.

$$P \cong Q \quad \Longleftrightarrow \quad P \sqsubseteq Q \quad \text{and} \quad Q \sqsubseteq P.$$

The equivalence between two processes is implemeted by the Circal System giving to the expression

$$P == Q$$

the result `true`, if $P \cong Q$ and `false`, otherwise.

3 The Methodology

As with any process algebra, the Circal process algebra is a low level language containing the primitive constructs to model concurrency. For this reason it is a very flexible language and it is possible to develop modelling styles that allow the description of high level aspects of concurrent systems, as well as application-oriented modelling styles.

One modelling style that is very useful in several application domains, and in particular in the description of communication protocols [1, 2] is the *constraint-based* modelling style [17, 13].

4 Constraint-based Modelling

The constraint-based modelling style is supported by the following distinctive features of the Circal process algebra:

- guarding of processes by sets of simultaneous actions;
- sharing of events over arbitrary numbers of processes;
- the particular nature of the composition operator which provides synchronization between processes without removal of the synchronising events in the resultant behaviour.

The behaviour of a process may be constrained simply by composing it with another Circal process which represents the constraint. As an example, consider the process $P \in \mathcal{P}_{\{a,b,c\}}$ defined as follows:

$$\begin{aligned} P &<- a\,B + b\,A + c\,P \\ A &<- a\,B + c\,A \\ B &<- b\,A + c\,B \end{aligned}$$

Process P generates all the finite strings on the alphabet $\{a, b, c\}$ that consist of alternating occurrences of a and b and arbitrary occurrences of c. Three constraints for process P are processes $C1 \in \mathcal{P}_{\{a,b,c\}}$ and $C2 \in \mathcal{P}_{\{b,c\}}$ and $C3 \in \mathcal{P}_{\{c\}}$ defined as follows

$$\begin{aligned} C1 &<- a\,S \\ S &<- a\,S + b\,S + c\,S \end{aligned}$$

$$\begin{aligned} C2 &<- b\,C + c\,D \\ C &<- c\,D \\ D &<- b\,C \end{aligned}$$

$$C3 <- /\backslash_{\{c\}}$$

Since P and $C1$ have the same sort and at the initial state $C1$ can perform only a the composite process $P * C1$ must perform a as the first action. Then $C1$ evolves to S, which can perform every possible sequence of action. Therefore, process $C1$ constrains the string to start with an occurrence of a with any arbitrary behaviour then following. Process $C2$ constrains b and c to occur in alternation, whereas it does not affect the occurences of action a, which does not belong to its sort. Process $C3$ has only c in its sort. Thus P can perform c only synchronously with $C3$, but $C3$ terminates immediately without performing any actions. Therefore C constrains P not to perform c.

Example 4.1

An important element in a protocol specification is the *assumptions* about the environment in which the protocol is executed [11]. The constraint-based modelling permits us to specify these assumptions within a process that will be composed with the specification of the protocol.

In the protocol verified by Cerone *et al.* [1] messages consist of finite sequences of 0 and 1 bits and are encoded by a sender, according to a Manchester encoding scheme, into transitions of the voltage between two levels on the single wire bus connecting the components. Since downgoing transitions take a significant time to change from high to low level, they do not appear to the receiver as edges. So the receiver can observe only upgoing edges and this causes a loss of information during the transmission, which results in an ambiguity in the decoding by the receiver. This ambiguity is overcome by *assuming* that the protocol will always work in an environment that guarantees the following constraints on the input:

- Every message starts with the bit 1;
- Every message either has an odd length or ends with 00.

A sequence of messages is readily modelled in Circal by a nested series of guarded processes where the guards are events consisting of single actions that can describe a bit 0, or a bit 1, or the "end of the message". Different actions are used for input and output message. We represent 0, 1 and "end of the message" in the input by i_0, i_1 and i_e, respectively, and in the output by o_0, o_1 and o_e, respectively. For example the sequence of 2 messages, given by the message 101 followed by the message 1100 is represented in the input by the process

$$M_i = i_1\, i_0\, i_1\, i_e\, i_1\, i_1\, i_0\, i_0\, i_e\, /\backslash$$

and in the output by the process

$$M_o = o_1\, o_0\, o_1\, o_e\, o_1\, o_1\, o_0\, o_0\, o_e\, /\backslash$$

The assumption about the environment is modelled by the process Con defined as follows:

$$\begin{aligned}
Con &<- i_1\, O_1 \\
O_0 &<- i_0\, E_{00} + i_1\, E + i_e\, Con \\
O_1 &<- i_0\, E + i_1\, E + i_e\, Con \\
E_{00} &<- i_0\, O_0 + i_1\, O_1 + i_e\, Con \\
E &<- i_0\, O_0 + i_1\, O_1
\end{aligned}$$

We can notice that the "end of the message" i_e appears only in the states that define an odd length (O_0 and O_1) or an even length with the last two bits equal to 0 (E_{00}).

The constaint-based modelling style supported by the Circal process algebra has been used in other application domains, such as asynchronous hardware [4, 5] and also allows the modelling of *timing constraints* [3, 4].

5 A Temporal Logic for Simultaneous Actions

In this section we define, the *Simultaneous Action Unified Branching* Logic (SAUB) [6], a branching time temporal logic whose temporal operators are based on actions. With respect to the previous logics based on actions, SAUB has the additional feature of expressing also simultaneity between actions. Therefore it permits us to discriminate between *true concurrency* and *non deterministic interleaving*. We give here only a brief introduction to the logic and we refer the reader to the technical report [6] for further explanations.

Definition 5.1

The syntax of the *action formulas* on $\mathcal{A}$ is given by

$$\begin{array}{rcl} AF & ::= & \epsilon \mid A \mid \neg AF \mid AF \vee AF \\ A & ::= & a \quad \text{for each } a \in \mathcal{A} \end{array}$$

The set of all the action formulas on $\mathcal{A}$ is denoted by $\mathcal{AF}(\mathcal{A})$.

Definition 5.2

Let $\mathcal{A}$ be a finite, non empty set of actions. Then for each $a \in \mathcal{A}$, $f, f_1, f_2 \in \mathcal{AF}(\mathcal{A})$ and $\mu \in 2^{\mathcal{A}}$

$$\begin{array}{lcl} \mu, \mathcal{A} \models \epsilon & \Longleftrightarrow & \mu = \emptyset \\ \mu, \mathcal{A} \models a & \Longleftrightarrow & a \in \mu \\ \mu, \mathcal{A} \models \neg f & \Longleftrightarrow & \mu, \mathcal{A} \not\models f \\ \mu, \mathcal{A} \models f_1 \vee f_2 & \Longleftrightarrow & \mu, \mathcal{A} \models f_1 \quad \text{or} \quad \mu, \mathcal{A} \models f_2 \end{array}$$

Definition 5.3

For each $f \in \mathcal{A}F(\mathcal{A})$ we define

$$C_{\mathcal{A}}(f) \stackrel{\text{def}}{=} \{\mu \in 2^{\mathcal{A}} | \mu, \mathcal{A} \models f\}.$$

Definition 5.4

We define the following derived operators:

$$\begin{aligned}
\mathbf{1}_{\mathcal{A}} &\stackrel{\text{def}}{=} \epsilon \vee \bigvee \mathcal{A} \\
\mathbf{0}_{\mathcal{A}} &\stackrel{\text{def}}{=} \neg \mathbf{1}_{\mathcal{A}} \\
f_1 \wedge f_2 &\stackrel{\text{def}}{=} \neg(\neg f_1 \vee \neg f_2) \\
f_1 \rightarrow f_2 &\stackrel{\text{def}}{=} \neg f_1 \vee f_2 \\
f_1 \leftrightarrow f_2 &\stackrel{\text{def}}{=} (f_1 \rightarrow f_2) \wedge (f_1 \rightarrow f_2)
\end{aligned}$$

Definition 5.5

For each $\mu \in 2^{\mathcal{A}}$ we define the *characterizing formula* of μ as follows.

$$f_\mu = \begin{cases} \neg\epsilon \wedge \bigvee_{a \in \mathcal{A} \setminus \mu} \neg a \wedge \bigvee_{a \in \mu} a & \text{if } \mu \neq \emptyset \\ \epsilon \wedge \bigvee_{a \in \mathcal{A}} \neg a & \text{otherwise} \end{cases}$$

The following theorem shows that f_μ characterizes completely the set $\mu \in 2^{\mathcal{A}}$. It is a simple consequence of Definition 5.3 [6].

Theorem 5.1

For each $\mu \in 2^{\mathcal{A}}$

$$C_{\mathcal{A}}(f_\mu) = \{\mu\}.$$

Definition 5.6

The syntax of *temporal formulas* on $\mathcal{A}$ is given by

$$\begin{aligned}
F \quad ::= \quad & \forall AF \ldots AF \Box AF \\
| \quad & \exists AF \ldots AF \Box AF \\
| \quad & \neg F \mid F \vee F \\
| \quad & \forall AF \ldots AF \Box F \\
| \quad & \exists AF \ldots AF \Box F
\end{aligned}$$

The set of all the temporal formulas on $\mathcal{A}$ is denoted by $\mathcal{F}(\mathcal{A})$.

Definition 5.7

Let $\mathcal{T} = \langle \mathcal{S}, \mathcal{A}, \rightarrow, s_0 \rangle$ be a LTS. Then for each $f, f_1, \ldots, f_n \in \mathcal{A}F(\mathcal{A})$ and $F, F_1, F_2 \in \mathcal{F}(\mathcal{A})$

$s_0, \mathcal{T} \models \forall f_1, \ldots, f_n \Box f$
$\iff$ for all $s_1, \ldots, s_n \in \mathcal{S}$ for all $\mu_1, \ldots, \mu_n \in 2^{\mathcal{A}}$
if $\mu_i, \mathcal{A} \models f_i, \quad i = 1, \ldots, n, \quad$ and $\quad s_0 \stackrel{\mu_1}{\rightarrow} s_1 \stackrel{\mu_2}{\rightarrow} \ldots s_{n-1} \stackrel{\mu_n}{\rightarrow} s_n$
then $\mu_i, \mathcal{A} \models f, \quad i = 1, \ldots, n, \quad$ and
for all $\rho \in \Pi_{\mathcal{T}}(s_n)$ for all $k \geq 0$ for all $\nu \in 2^{\mathcal{A}}$
$\rho(k) \stackrel{\nu}{\rightarrow} \rho(k+1) \quad \Longrightarrow \quad \nu, \mathcal{A} \models f$

$s_0, \mathcal{T} \models \exists f_1, \ldots, f_n \Box f$
$\iff$ there exist $s_1, \ldots, s_n \in \mathcal{S}$ and $\mu_1, \ldots, \mu_n \in 2^{\mathcal{A}}$ such that
$\mu_i, \mathcal{A} \models f_i, \quad i = 1, \ldots, n, \quad \mu_i, \mathcal{A} \models f, \quad i = 1, \ldots, n,$
$s_0 \stackrel{\mu_1}{\rightarrow} s_1 \stackrel{\mu_2}{\rightarrow} \ldots s_{n-1} \stackrel{\mu_n}{\rightarrow} s_n$ and
there exists $\rho \in \Pi_{\mathcal{T}}(s_n)$ such that
for all $k \geq 0$ for all $\nu \in 2^{\mathcal{A}}$
$\rho(k) \stackrel{\nu}{\rightarrow} \rho(k+1) \quad \Longrightarrow \quad \nu, \mathcal{A} \models f$

$s_0, \mathcal{T} \models \neg F$
$\iff$ $s_0, \mathcal{T} \not\models F$

$s_0, \mathcal{T} \models F_1 \vee F_2$
$\iff$ $s_0, \mathcal{T} \models F_1$ or $s_0, \mathcal{T} \models F_2$

$s_0, \mathcal{T} \models \forall f_1, \ldots, f_n \Box F$
$\iff$ for all $s_1, \ldots, s_n \in \mathcal{S}$ for all $\mu_1, \ldots, \mu_n \in 2^{\mathcal{A}}$
if $\mu_i, \mathcal{A} \models f_i, \quad i = 1, \ldots, n, \quad$ and $\quad s_0 \stackrel{\mu_1}{\rightarrow} s_1 \stackrel{\mu_2}{\rightarrow} \ldots s_{n-1} \stackrel{\mu_n}{\rightarrow} s_n$
then for all $\rho \in \Pi_{\mathcal{T}}(s_n)$
for all $k \geq 0 \quad \rho(k), \mathcal{T} \models F$

$s_0, \mathcal{T} \models \exists f_1, \ldots, f_n \Box F$
$\iff$ there exist $s_1, \ldots, s_n \in \mathcal{S}$ and $\mu_1, \ldots, \mu_n \in 2^{\mathcal{A}}$ such that
$\mu_i, \mathcal{A} \models f_i, \quad i = 1, \ldots, n, \quad$ and $\quad s_0 \stackrel{\mu_1}{\rightarrow} s_1 \stackrel{\mu_2}{\rightarrow} \ldots s_{n-1} \stackrel{\mu_n}{\rightarrow} s_n$
and there exists $\rho \in \Pi_{\mathcal{T}}(s_n)$ such that
for all $k \geq 0 \quad \rho(k), \mathcal{T} \models F$

Definition 5.8

Let be $f_1, \ldots, f_n \in \mathcal{A}F(\mathcal{A})$, $F, F_1, F_2 \in \mathcal{F}(\mathcal{A})$ and $\Phi \in \mathcal{A}F(\mathcal{A}) \cup \mathcal{F}(\mathcal{A})$ We define the following derived operators:

$$\top \stackrel{\text{def}}{=} \forall \mathbf{1}_{\mathcal{A}} \Box \mathbf{1}_{\mathcal{A}}$$

$$\bot \stackrel{\text{def}}{=} \neg\top$$
$$F_1 \wedge F_2 \stackrel{\text{def}}{=} \neg(\neg F_1 \vee \neg F_2)$$
$$F_1 \rightarrow F_2 \stackrel{\text{def}}{=} \neg F_1 \vee F_2$$
$$\exists f_1, \ldots, f_n \Diamond\Phi \stackrel{\text{def}}{=} \neg\forall f_1, \ldots, f_n \Box\neg\Phi$$
$$\forall f_1, \ldots, f_n \Diamond\Phi \stackrel{\text{def}}{=} \neg\exists f_1, \ldots, f_n \Box\neg\Phi$$
$$\exists \Diamond\Phi \stackrel{\text{def}}{=} \exists \mathbf{1}_{\mathcal{A}} \Diamond\Phi$$
$$\forall \Diamond\Phi \stackrel{\text{def}}{=} \neg\forall\top\Diamond\Phi$$
$$\forall \Box\Phi \stackrel{\text{def}}{=} \neg\exists\Diamond\neg\Phi$$
$$\exists \Box\Phi \stackrel{\text{def}}{=} \neg\forall\Diamond\neg\Phi$$
$$\exists X f_1, \ldots, f_n \stackrel{\text{def}}{=} \exists f_1, \ldots, f_n \Box\top$$
$$\forall X f_1, \ldots, f_n \stackrel{\text{def}}{=} \neg\exists\neg f_1 \ldots \neg f_n \Box\top$$

6 The Representation of Properties

A correctness concept that can be readily characterized in Circal is the behavioural equivalence between two processes. The Circal expression $P == Q$ is the implementation of the equivalence $P \cong Q$.

In verification, however, equivalence is often too strong a property. For certain systems, verifying their correctness consists of determining that certain properties hold, where these properties do not constitute a complete specification. This cannot be done in terms of equivalence, but rather involves the notion of behavioural inclusion.

Let $P \in \mathcal{P}_L$ and $Q \in \mathcal{P}_M$ be such that $L \subseteq M$ and P and Q are deterministic. Then

$$Q - (M \backslash L) \sqsubseteq P$$

if and only if Q can be seen to constrain P and restricts P to behave as Q, that is

$$P * Q \cong P.$$

The Circal expression $P * Q == Q$ characterizes the behavioural inclusion of Q in P.

When Q defines a model of a system, P can be seen as a property that Q must satisfy. However not all temporal properties can be verified through a behavioural equivalence.

A more general approach is the use of processes to *mark* with new *abstract actions* all the paths that satisfy the required property. The property

is then defined by a process whose sort consists only of marking actions. By composing the process that defines the model of the system with the marking process and abstracting away all actions apart from the marks, the composite process is equivalent to the process defining the property if and only if the model of the system satisfies the property.

6.1 Formula-based Characterization

In this section we define a characterization of a given property of a Circal process by means of an equivalence between processes that involve a representation of the temporal formula that defines the property. We start our characterization with the temporal formulas that do not contain more than one level of temporal operators. Notice that a formula like $\forall f \Box F$, with $F \in \mathcal{F}(\mathcal{A})$, has always more than one levels of temporal operators. We will use the following process definitions:

$$\begin{aligned} TL &<- \sum_{\mu \in 2^L} \mu\, TL \\ YN &<- yYN + nN \\ N &<- nN \\ Y &<- yY \end{aligned}$$

with $y, n \notin L$. The TL process permits all possible set of actions to occur anytime. The behaviour of the YN process consists of an infinite path labelled by y that has in any point an infinite branch labelled by n.

Theorem 6.1

The behaviour of a process $S \in \mathcal{P}_L$ satisfies the formula

$$\forall f_1, \ldots, f_r \Box f$$

if and only if

$$S * AL_{f_1,\ldots,f_r} * AGL_f - L \cong Y$$

where

$$\begin{aligned} AL_{f_1,\ldots,f_r} &<- AL_{f_1} \\ AL_{f_i} &<- \sum_{\mu \in C_L(f_i)} \mu\, AL_{f_{i+1}}, \qquad i = 1, \ldots, r-1 \\ AL_{f_r} &<- \sum_{\mu \in C_L(f_r)} \mu\, TL \\ AGL_f &<- \sum_{\mu \in C_L(f)} (\mu \cup \{y\})\, AGL_f \; + \sum_{\mu \notin C_L(f) \cup \emptyset} (\mu \cup \{n\})\, N \end{aligned}$$

Proof The $AL_{f_1 \ldots f_r}$ process constrains other processes to start with the occurrence of a sequence of sets of actions such that the i-th set satisfies the temporal formula f_i. The AGL_f process marks with the new action y all the action sets that satisfies f and with the new action n all the other action sets.

Thus, $S * AL_{f_1,\ldots,f_r} * AGL_f - L$ is equivalent to Y

if and only if

$S * AL_{f_1,\ldots,f_r} * AGL_f$ performs only action sets marked by y

if and only if

$S * AL_{f_1,\ldots,f_r}$ performs only action sets that satisfy f

if and only if

S performs only action sets that satisfy the temporal formula f after being constrained to start with the occurrence of a sequence of action sets such that the i-th set satisfies the temporal formula f_i

if and only if

$\mathcal{B}(S)$ satisfies $\forall f_1, \ldots, f_r \Box f$. □

Theorem 6.2

The behaviour of a process $S \in \mathcal{P}_L$ satisfies the formula

$$\exists f_1, \ldots, f_r \Box f$$

if and only if

$$S * EGL_f^{f_1,\ldots,f_r} - L \cong YN$$

where

$$\begin{aligned}
EGL_f^{f_1,\ldots,f_r} &< - \quad EGL_f^{f_1} \\
EGL_f^{f_i} &< - \sum_{\mu \in C_L(f_i) \cap C_L(f)} \mu \cup \{y\} EGL_f^{f_{i+1}} + \\
&\qquad \sum_{\mu \notin C_L(f_i) \cap C_L(f) \setminus \emptyset} \mu \cup \{n\}\, N, \qquad i = 1, \ldots, r-1 \\
EGL_f^{f_r} &< - \sum_{\mu \in C_L(f_i) \cap C_L(f)} \mu \cup \{y\}\, EGL_f +
\end{aligned}$$

$$EGL_f \quad <- \quad \sum_{\mu \notin C_L(f_i) \cap C_L(f) \setminus \emptyset} \mu \cup \{n\}\, N \;\; \sum_{\mu \in C_L(f)} (\mu \cup \{y\})\, EGL_f \; + \; \sum_{\mu \notin C_L(f) \cup \emptyset} (\mu \cup \{n\})\, N$$

Proof The $EGL_f^{f_1 \ldots f_r}$ process marks with the new action y all the action sets in an initial sequence of action sets such that the i-th set satisfies the temporal formula f_i and the temporal formula f. It marks all the other action sets with the new action n. After marking with n, every $EGL_f^{f_i}$ gives control to the N process, which performs n forever, whereas, after marking with y the last action set of the sequence, $EGL_f^{f_r}$ gives control to the EGL_f process. The EGL_f process marks with y all the action sets that satisfy the temporal formula f until an action set that does not satisfy f is performed. Such an action set and all the following ones are marked with n. Moreover, EGL_f allows at any point the occurrence of an action set containing only n.

Thus, $S * EGL_f^{f_1,\ldots,f_r} - L$ is equivalent to YN

if and only if

$\mathcal{B}(S * EGL_f^{f_1,\ldots,f_r} - L)$ consists of an infinite path labelled by y that has in any point an infinite branch labelled by n

if and only if

$\mathcal{B}(S * EGL_f^{f_1,\ldots,f_r})$ contains at least an infinite path whose action sets contain y and all these infinite paths have in any point an infinite branch whose action sets contain n

if and only if

$\mathcal{B}(S)$ contains a path that starts with a sequence of r action sets such that the i-th set satisfies the temporal formulas f_i and f and that after the r-th action set contains only action sets that satisfy f

if and only if

$\mathcal{B}(S)$ satisfies $\exists f_1, \ldots, f_r \Box f$. □

Formulas consisting of derived temporal operators can be characterized after transforming the derived operators in the primitive ones.

Example 6.1

Let $S \in \mathcal{P}_{\{a,b,c\}}$ be the process defined as follows:

$$S \;<-\; a\,S \;+\; (b\,c)\,S$$

We can notice that S satisfies

$$\forall c \Box (a \vee b) \quad \text{and} \quad \exists c \Box (b \wedge c)$$

but satisfies neither

$$\forall c \Box b \quad \text{nor} \quad \exists c \Box (a \wedge c)$$

In fact

$$S * AL_c * AGL_{a \vee b} - L \quad \text{and} \quad S * EGL^c_{b \wedge c} - L$$

are equivalent to Y and YN, respectively, whereas the behaviour of $S * AL_c * AGL_b - L$ is given by

$$\begin{aligned} S_0 &<- y\,S_1 \\ S_1 &<- y\,S_1 \;+\; n\,S_1 \end{aligned}$$

which is not equivalent to Y, and the behaviour of $S * EGL^c_{a \wedge c} - L$ is given by

$$\begin{aligned} S_0 &<- y\,S_1 \\ S_1 &<- n\,S_1 \end{aligned}$$

which is not equivalent to YN.

Now we show an example of characterization of a formula containing two levels of temporal operators.

Example 6.2

The formula

$$\forall f_1, \ldots, f_r \Box \forall g_1, \ldots, g_s \Diamond h$$

defines a schema of the important class of liveness properties. The formula is equivalent to

$$\forall f_1, \ldots, f_r \Box \neg \exists g_1, \ldots, g_s \Box \neg h$$

We can notice that the fragment "$\forall f_1, \ldots, f_r$" defines a constraint characterized by the process $AL_{f_1,\ldots,f_r}$. Therefore the behaviour of a process $S \in \mathcal{P}_L$ satisfies the formula if and only if

$$S * AL_{f_1,\ldots,f_r} * EGL^{g_1,\ldots,g_s}_h - L \;\ncong YN$$

In the protocol defined in Example 4.1, since the receiver can observe only upgoing edges, a sequence 01 encoded by the sender can be decoded by the receiver only when it observes the upgoing edge that encodes the bit 1. The formula $\forall \mathbf{1}_{\mathcal{A}} \Box \forall i_0, i_1 \Diamond d_{01}$ where action d_{01} defines the decoding of the sequence 01 by the receiver. However, the model of the protocol does not satisfy it because the decoding fails when the input message does not respect the constraints defined in Example 4.1.

6.2 Model-based Characterization

The following theorem shows an alternative characterization of the formula $\forall f_1, \ldots, f_r \Box f$.

Theorem 6.3

The behaviour of a process $S \in \mathcal{P}_L$ satisfies the formula

$$\forall f_1, \ldots, f_r \Box f$$

if and only if

$$S * AL_{f_1,\ldots,f_r} * G_f \;\cong\; S * AL_{f_1,\ldots,f_r}$$

where

$$G_f <- \sum_{\mu \in C_L(f)} \mu\, G_f$$

Proof The G_f process constrains other processes to perform only the action sets characterized by f. Thus, $S * AL_{f_1,\ldots,f_r} * G_f$ is equivalent to $S * AL_{f_1,\ldots,f_r}$ if and only if $S * AL_{f_1,\ldots,f_r}$ can perform only action sets satisfying f, that is if and only if S satisfies $\forall f_1, \ldots, f_r \Box f$. □

In this characterization the property is verified through a behavioural inclusion. This is possible only for safety properties. The approach becomes very interesting when the safety property is characterized in terms of a known model that satisfies it (*model-based characterization*). If process P is the model-based characterization of such a safety property, the equivalence given in Theorem 6.3 becomes

$$S * P \cong S \tag{1}$$

Therefore, system S satisfies property P if and only if the behaviour of P is included in the behaviour of S.

In most cases the correctness of a system depends on the assumptions made about its environment. Thus the process C defining these assumptions

must be composed with S, before the verification. Therefore, equivalence (1) becomes

$$S * C * P \cong S * C \tag{2}$$

This approach exploits the modelling power of a process algebra such as Circal and can overcome the problem of the characterization in terms of a very complex logical formula.

Example 6.3

A typical example that uses the model-based characterization is the correctness proof of a communication protocol. A necessary property for the correctness of a protocol is the following.

> Each sequence of messages that is accepted as an input by the protocol is output unchanged by the protocol.

This property is typical of a buffer and the specification of a buffer is well known. If process S describes the protocol and B describes the specification of the buffer, then the protocol satisfies the property if and only if the behaviour of B is included in the behaviour of S, which is expressed in Circal by

$$S * B \cong S \tag{3}$$

In general the size of the buffer, and therefore the number of its states, depends on the protocol. However, we can be certain that there exists a finite buffer size for each given protocol modelled in Circal, since Circal can describe only finite-state behaviours. Therefore the property can be proved by showing that the above equivalence is true for some finite size of B.

Since the correctness of the protocol defined in Example 4.1 depends on the assumptions defined by process Con, the equivalence to be verified becomes

$$S * Con * B \cong S * Con$$

However this is not yet sufficient to guarantee the correctness of the protocol, since also a process that accepts fewer inputs than the possible inputs that are characterized by Con satisfies the equivalence above. It is necessary to prove that S accepts at least all the inputs characterized by Con. This is true when the behaviour of the restriction of S to the input actions is included in the behaviour Con. Such a property is verified by the equivalence

$$Con * (S - O) \cong Con$$

where $O = o_0\ o_1\ o_e$ is the list of all output actions.

We can notice that in this case there is a behavioural inclusion of a restriction of the system (to input actions) in the property, which is a very simple liveness property. "The system accepts a class of inputs" means that the system eventually reacts to every input belonging to such a class. In some cases [1, 2] the two equivalences

$$S * Con * B \cong S * Con \quad \text{and} \quad Con * (S - O) \cong Con$$

are sufficient to verify the correctness of a protocol P. The property characterization is thus entirely model-based.

For a general approach to the verification of liveness properties the recourse to a formula-based characterization is a need. However, in the characterization

$$S * AL_{f_1,\ldots,f_r} * EGL_h^{g_1,\ldots,g_s} - L \ncong YN$$

of the formula

$$\forall f_1, \ldots, f_r \Box \forall g_1, \ldots, g_s \Diamond h$$

the constraint $AL_{f_1,\ldots,f_r}$ is a very simple assumption about the environment. In practical cases the assumptions about the environment can be very difficult to be characterized in terms of temporal logic formulas. The above characterization still works when $AL_{f_1,\ldots,f_r}$ is replaced by any other constraint. Therefore, we can replace $AL_{f_1,\ldots,f_r}$ by a process C that is a model-based characterization of the assumptions. This is equivalent to verify the constrained system $S * C$ rather than the unconstrained system S. In this way the combination of the formula-based and model-based characterizations increases the power of the verification methodology.

Example 6.4

In the protocol defined in Example 4.1 the assumptions about the environment are modelled by the Con process defined in Example 4.1. Therefore, if $P \in \mathcal{P}_L$ is the process that models the protocol, then $P * Con * EGL_{d_{01}}^{i_0,i_1} - L \cong /YN$ proves that "when the input satisfies the constraints modelled by Con always a sequence 01 is eventually decoded". Therefore, the composite system $P * Con$ satisfies $\forall \mathbf{1}_{\mathcal{A}} \Box \forall i_0, i_1 \Diamond d_{01}$ even if P does not.

The model-based characterization of property has also been applied to the verification of timing constraints [4] and performance properties [5] in the domain of asynchronous hardware.

Example 6.5

In an asynchronous micropipelined processor the evaluation of a pipeline stage is governed by local interactions with its neighbours using a request acknowledge handshaking protocol. It is possible that one stage is evaluating while at the same time a stage further on is transferring an instruction to its neighbour. Thus, whilst the performance of a synchronous pipeline is governed entirely by the clock rate that can be achieved with a particular logic design, the performance of an asynchronous pipeline depends as well on the design of the handshaking controls for each stage. In particular, if the asynchronous logic pipeline is to be as fast as the synchronous one it must be possible for all the evaluation of logic stages to overlap as in the synchronous case.

Equivalence 2 has been used in the verification of the simple property that "adjacent stages can never be occupied at the same time" [4, 5]. This property is verified by exploiting a distinctive features of the Circal process algebra: simultaneous event guards. A new *abstract action* is introduced in every stage to mark the interval where the stage is full. If we denote by y_i the abstract action introduced in the i-th stage then the process

$$Y_{i,j} \; <- \; y_i \, Y_{i,j} \; + \; y_j \, Y_{i,j}$$

characterizes the property that y_i and y_j never occur simultaneously, that is the i-th and j-th stages "can never be occupied at the same time". Therefore, if S models the handshaking control for the whole micropipeline, then checking for each i

$$S * Y_{i,i+1} \cong S$$

proves that "adjacent stages can never be occupied at the same time".

If such a property holds for adjacent stages, but not for alternate stages then the degree of parallelism achieved by the micropipeline is only 50% of the potential parallelism. If it does not hold for any pair of stages, then adjacent stages may be occupied at the same time. So no upper bound of 50% is given to the degree of parallelism. However, we would like to know whether all stages can be occupied at the same time.

In order to characterize such a property we combine the formula-based and the model-based characterizations. We use the same abstract action y for every stage of the micropipeline [5]. Then the process Y given in Section 6.1 characterizes the property that y occurs on an infinite path in the behaviour of the micropeline. In this case equivalence 2 cannot be applied. If L is list of all actions in the sort of S then

$$(S * Y) - L \cong Y \tag{4}$$

proves that after abstracting away all the actions in the sort of S from the composition $S * Y$, there is still an infinite sequence of actions y in the resultant behaviour. This means that there is a path where all stages are occupied at the same time. Therefore, the whole potential parallelism can be effectively reached [5].

7 Discussion and Future Work

In this paper we have developed a methodology for the automatic verification of concurrent systems by using the constraint-based modelling style available within the Circal process algebra. Three distinctive features of the Circal process algebra are the basis of this methodology, namely, simultaneous event guards, sharing of events over arbitrary numbers of processes and the nature of the composition operator which provides synchronisation of processes without the removal of the synchronising events in the resultant behaviour.

A temporal property of the system under analysis can be characterized either through a translation of the SAUB formula that defines the property into a set of processes and into the rules to compose these processes with one another (*formula-based* characterization), or by a well-known model that satisfies the property (*model-based* characterization). The formula-based characterization allows the verification of general temporal properties, whereas the model-based characterization can be applied only to the verification of safety properties and of some simple liveness properties. The two characterizations can also be combined together, so increasing the power of the verification methodology.

In future work we want to extend the logic with more powerful temporal operators and also would like to add compositionality to SAUB by developing *meta-rules* to combine properties of different processes, which have possibly different sorts, and infer a property of the composite process. This could be a first step towards a decomposition of a global property of the overall system into the local properties of its components, which can be verified on a much smaller state spaces.

Acknowledgements

We would like to thank Alex Cowie and Colin Fidge for helpful discussions. Colin also reviewed a draft of this paper providing many useful comments. This research has been partly supported by the Australian Research Council

and the Information Technology Division of the Defence Science Technology Organisation.

References

[1] A. Cerone, A.J. Cowie, G.J. Milne and P.A. Moseley. Description and verification of a time-sensitive protocol. Technical report CIS-96-009, University of Australia, School of Computer and Information Science, October 1996.

[2] A. Cerone, A.J. Cowie, G.J. Milne and P.A. Moseley. Modelling a time-dependent protocol using the Circal process algebra. In *Hybrid and Real-Time Systems*, volume 1201, Lecture Notes in Computer Science, pages 124–138. Springer Verlag, 1997.

[3] A. Cerone, and G.J. Milne. Modelling a time-dependent protocol using the Circal process algebra. In *Algebraic Methodology and Software Technology*, volume 1349, Lecture Notes in Computer Science, pages 108–122. Springer Verlag, 1997.

[4] A. Cerone, D.A. Kearney and G.J. Milne. Integrating the verification of timing, performance and correctness properties of concurrent systems. In *Proc. of the International Conference on Application of Concurrency to System Design*, pages 109–119. IEEE Comp. Soc. Press, 1998.

[5] A. Cerone and G.J. Milne. A methodology for the formal analysis of asynchronous micropipelines Technical report 99-33, The University of Queensland, Software Verification Research Centre, October 1999.

[6] A. Cerone. A temporal logic for simultaneous actions. Technical report 00-09, The University of Queensland, Software Verification Research Centre, March 2000.

[7] E.M. Clarke, E.A. Emerson and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[8] R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Soncurrent Processes*, volume 469, Lecture Notes in Computer Science, pages 407–419. Springer Verlag, 1990.

[9] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of ACM*, 32:137–162, 1985.

[10] C.A.R. Hoare. *Communication Sequential Processes.* International Series in Computer Science, Prentice Hall, 1985.

[11] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[12] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Press, 1993.

[13] G.J. Milne. *Formal Specification and Verification of Digital Systems.* McGraw-Hill, 1994.

[14] R. Milner. *Communication and Concurrency.* International Series in Computer Science, Prentice Hall, 1989.

[15] F. Moller. The semantic of Circal. Technical report HDV-3-89, University of Strathclyde, Department of Computer Science, 1989.

[16] C. Stirling. Modal and temporal logics for processes. In *Logics for Concurrency — Structure versus Automata*, volume 1043, Lecture Notes in Computer Science, pages 149–237. Springer Verlag, 1996.

[17] C.A. Vissers, G. Scollo, M. van Sinderen and E. Brinksma. Specification styles in distributed systems design and verification. *Theoretical Computer Science*, 89:179–206, 1991.