Expressing Iterative Properties Logically in a Symbolic Setting

C. Shankland[1], J. Bryans[2], and L. Morel[3]

[1] Department of Computing Science and Mathematics, University of Stirling, UK
[2] Centre for Software Reliability, University of Newcastle, UK
[3] Verimag, France

Abstract. We present a logic for reasoning about LOTOS behaviours which allows properties involving repeated patterns over actions and data to be expressed. The semantics of the logic is given with respect to symbolic transition systems. Several motivational examples are included. The reader is assumed to have passing familiarity with LOTOS.
Keywords: LOTOS, Formal Verification, Temporal logics, Infinite State Systems, Symbolic representation.

1 Introduction

When describing a system formally it is often useful to be able to do so at different levels of abstraction. This enhances our confidence in the correctness of our mental model of the system, especially if the different descriptions can be related mathematically to each other. An obvious example of this activity is seen in the use of modal and temporal logics to describe abstract properties of a system which can then be validated with respect to a more concrete, implementation focussed, description. For example, the use of PROMELA and CTL in the model checker SPIN [11].

In previous work we have established a symbolic framework for reasoning about Full LOTOS [12] based on *Symbolic Transition Systems* (STS) [4]. This new semantics for LOTOS allows behaviour to be expressed in a more compact, elegant fashion than is possible using the standard semantics [12]. In particular, this is a way of avoiding the state explosion caused by the use of data in LOTOS. A modal logic called FULL has been defined [3], but although this logic has the desirable theoretical property of adequacy with respect to bisimulation [2], it lacks the expressiveness required in many applications.

For example, consider the simple buffer which accepts data at gate `in` and outputs that data at gate `out` (in pseudo LOTOS, `B = in?x:Nat; out!x; B`). An abstract property of this buffer is *"after an* `in` *event, an* `out` *event occurs"*. Of course, it would be useful to express the repetition of this pattern *"after the* `in` *action, the* `out` *action happens, and then repeat"*. We also want to express something about the data involved in the transaction *"if we input the value x then what is output is also x, and never anything else"*. Again, it is desirable to express this over a number of iterations.

Another simple example is the communication protocol dealing with lossy channels in which an item is sent and resent until an acknowledgement is received "*repeatedly send m until ack m*". Here the goal is to repeat a single action an indeterminite (but finite) number of times.

Admittedly these are simple examples, but they capture the principle that such repetitions are central to defining the behaviour of many systems, including communications protocols, games, hardware systems, operating systems, and telecommunications. Moreover, it is useful to be able to express these properties in a high level, abstract language such as temporal logic. Logics of this nature have been defined for LOTOS, most notably in association with the CADP toolset [7, 15, 16], and used in, for example, verifications of the reconfiguration protocol [5] and the IEEE 1394 tree identify protocol [17].

Our goal here is to provide a logic which has the expressivity described above, and is based on symbolic transition systems, allowing data to be dealt with in a more efficient fashion and avoiding some forms of state explosion. We present a logic, FULL*, which extends FULL [3]. FULL* is designed to be expressive: we want the ability to formulate as many useful queries as possible since we feel this is more important for the software development process than the theoretically desirable adequacy of FULL. The grammar of FULL* draws features familiar from traditional modal logics [10, 3], and from XTL [14] which in turn draws on the language of regular expressions. This paper explores the expressivity of the resulting logic via a series of simple examples. In particular, core safety, liveness, response and reachability properties are examined.

2 Reasoning about Infinite State Systems

Over a number of years we have developed a simple and elegant framework in which it is possible to reason about both data and processes (or flow of control). The core of our framework (see Figure 1) is a symbolic semantics for Full LOTOS [4] using Symbolic Transition Systems (STS) [9].

Although our main interest is in LOTOS, in fact STS may be used to express the semantics of many languages, therefore FULL* may also be used to reason about behaviours expressed in those other languages. The reader is assumed to be familiar with LOTOS, but an excellent tutorial is available [13].

The standard semantics [12] are represented in the leftmost portion, where labelled transition systems give meaning to LOTOS specifications. This is the world in which CADP and XTL are based. The righthand portion of the Figure represents our contribution; namely, the symbolic semantics for LOTOS, an HML-like logic (FULL) [3] and its extension FULL* (described in this paper), and various flavours of bisimulation [4] (not discussed further here). The arrows between the components represent relationships. For example, we have proved that the standard, concrete and symbolic bisimulations are all equivalent for closed processes (i.e. those with no free variables), and further, that the same equivalence relation is induced by the modal logic FULL. These results are all essential to showing the strength and self-consistency of our symbolic frame-

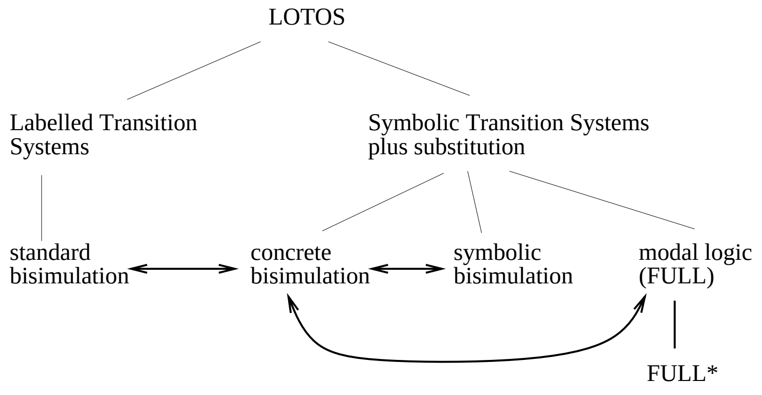

Fig. 1. Symbolic Framework for LOTOS

work, and its consistency with the standard semantics over closed behaviours (i.e. without free variables). In other words, we have not simply re-defined the LOTOS language in different terms, yielding a different language from that of the standard; we have striven to ensure that the two semantics can be used interchangeably, with ours offering computational advantages for reasoning about processes.

In this section we reproduce the definitions associated with *symbolic transition systems* and FULL in order to give a basis for the definition of FULL*.

2.1 Symbolic Transition Systems

Labelled transition systems are commonly used for specification (e.g. process algebra semantics, IO automata). Hennessy and Lin [9] developed a *symbolic* version of transition systems to combat the problems of infinite breadth introduced by transitions over data, and of partial specification introduced through parameterised processes. Other work in symbolic transition systems includes the work of Eertink [6] and the NTIF work of Garavel and Lang [8]. Both of these approaches are quite different from ours, having as a goal a more efficient implementation of simulation, equivalence checking, and/or model checking. In particular, the NTIF development is aimed at interpretation of E-LOTOS behaviours.

Here we use a version of the Hennessy and Lin symbolic transition systems customised to fit the philosophy of communication in LOTOS (multiway synchronisation, i.e. more than two processes may communicate, and early acquisition of values, i.e. binding of values to variables at the point of synchronisation).

The main features of the STS are that transitions are decorated with a Boolean, b, and an action, α. The Boolean expresses under which conditions the transition may occur. Actions in LOTOS must be associated with a *gate*, and may additionally offer some *data*. Therefore we split actions into two sets: *SimpleEv* and *StructEv*. An action in *SimpleEv* consists only of a gate name, g.

An action in *StructEv* consists of a gate name g and a data expression d. We use α to denote an action in *SimpleEv* $\cup$ *StructEv*.

The data expression may contain free variables. In this case we refer to it as *open*. Alternatively, it has no free variables and is therefore *closed*. The same is true of LOTOS behaviours, or states in a transition system. Either they contain free variables and are open, or they contain no free variables and are closed. The function $fv()$ can be applied to a behaviour expression to determine the free variables of that expression. We do not repeat the definition of $fv()$ here.

Note there is no syntactic distinction between input and output events since this goes against the LOTOS interpretation of events. We can think of data offers such as `g!42` as output events, and `g?x:Nat` as input events. It is more accurate to think of these as denoting different sets of values. For "output" events the data expression will evaluate to a single value, while for "input" events the data expression will be a variable name, potentially instantiated by many possible values. In STS the Boolean condition and data expression together encapsulate the set of possible values offered at a gate.

We repeat here the definition given in [4]:

Definition 1. *Symbolic Transition Systems*
An STS is composed of :

- *a non-empty set of states. To each state S, we associate a set of free variables, $fv(S)$. This can be computed from the syntactic behaviour associated with S.*
- *an initial state, S_0.*
- *a set of transitions $S \xrightarrow{b \quad \alpha} S'$ such that $fv(S') \subseteq fv(S) \cup fv(\alpha)$ and $fv(b) \subseteq fv(S) \cup fv(\alpha)$ and $\sharp(fv(\alpha) - fv(S)) \leq 1$. b is a boolean expression and $\alpha \in$ SimpleEv $\cup$ StructEv.*

That is, any new names in S' must come from the action α, the Boolean condition b may refer to variables in S and any new variable introduced in α, and only one variable may be introduced by α. The last part of the restriction is an artificial imposition by us to make the semantics clearer. In fact, multiple variables could be introduced by using a list mechanism.

Operationally, states are insufficient as a basis for our framework. Consider the simple buffer, repeating the actions `in?x:Nat; out!x`. To evaluate this process (e.g. with respect to another process for bisimulation, or with respect to a modal formula) a substitution of value for the variable `x` is required; however, the substitution must change at every iteration. In order to accommodate this, the framework for reasoning must be based on *terms*.

A term T_σ is a pair (T,σ) where T is a node in the STS being considered and σ is a substitution. σ is a partial function from variables to new variables, or to values, i.e. from *Var* to *Var* $\cup$ *Val*. We require $range(\sigma) \subseteq fv(T)$, this means parts of the substitution which are no longer relevant are discarded. In the definition of both FULL and FULL* the substitutions always map variables to values at the time of their first use, therefore for subsequent transitions, the data item associated with an action is either a value, or a single variable name, indicating the introduction of that variable.

The notion of transitions between states in a symbolic transition system must be transposed to the notion of transitions between terms:

Definition 2. *Transitions Over Terms*

$$
\begin{array}{ll}
T \xrightarrow{b \quad a} T' \textit{ implies } T_\sigma \xrightarrow{b\sigma \quad a} T'_{\sigma'} & \\
T \xrightarrow{b \quad gE} T' \textit{ implies } T_\sigma \xrightarrow{b\sigma \quad gE\sigma} T'_{\sigma'} & \\
\qquad\qquad \textit{where } fv(E) \subseteq fv(T) & \\
T \xrightarrow{b \quad gx} T' \textit{ implies } T_\sigma \xrightarrow{b\sigma[z/x] \quad gz} T'_{\sigma'[z/x]} & \\
\qquad\qquad \textit{where } x \notin fv(T) \textit{ and } z \notin fv(T_\sigma) &
\end{array}
$$

In all cases, $\sigma' = fv(T') \lhd \sigma$*, that is to say the restriction of* σ *only containing elements of* $fv(T')$*.*

In [3] the logic FULL was defined over transitions on terms. Similarly, the logic FULL* will be presented over transitions on terms.

2.2 Paths

Given that FULL* will include iterative operators we also need to define the notion of a path over terms, denoted π, as a finite sequence of transitions: $\pi = t_1 \xrightarrow{b_1 \quad \alpha_1} t_2 \xrightarrow{b_2 \quad \alpha_2} \cdots t_{n-1} \xrightarrow{b_{n-1} \quad \alpha_{n-1}} t_n$. A path can have free variables, a maximum of one of which can be introduced at each transition, for convenience, as with transitions over STS.

Given π as above, operators on paths include:

$\mathit{first}(\pi)$	the first term of a path: $\mathit{first}(\pi) = t_1$
$\mathit{last}(\pi)$	the last term of a path: $\mathit{last}(\pi) = t_n$
$\mathit{init}(\pi)$	the initial transition on a path: $\mathit{init}(\pi) = t_1 \xrightarrow{b_1 \quad \alpha_1} t_2$
$\pi \cdot \pi_2$	If $\pi_2 = s_1 \xrightarrow{c_1 \quad \beta_1} s_2 \cdots s_{m-1} \xrightarrow{c_{m-1} \quad \beta_{m-1}} s_m$, and $t_n = s_1$, the concatenation operator $\pi \cdot \pi_2$ (or $\pi\pi_2$) is defined as $t_1 \xrightarrow{b_1 \quad \alpha_1} t_2 \cdots t_{n-1} \xrightarrow{b_{n-1} \quad \alpha_{n-1}} t_n \xrightarrow{c_1 \quad \beta_1} s_2 \cdots s_{m-1} \xrightarrow{c_{m-1} \quad \beta_{m-1}} s_m$

Substitutions over paths are dealt with in the same way as substitutions over terms; on taking each transition the substitution is applied if it can be. It disappears as soon as it is no longer required.

2.3 FULL: A modal logic for Full LOTOS

FULL was proposed in [3] in order to express properties of STS and to capture the same equivalence as symbolic bisimulation over STS [2]. The form of the logic is inspired by HML (Hennessy-Milner Logic [10]), with the addition of quantifiers over data. FULL has the following syntax :

$$
\begin{array}{l}
\Phi ::= b \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]\Phi \mid \langle a \rangle \Phi \mid [\exists x\ g]\Phi \\
\qquad \mid [\forall x\ g]\Phi \mid \langle \exists x\ g \rangle \Phi \mid \langle \forall x\ g \rangle \Phi
\end{array}
$$

where b is a Boolean expression, $a \in \mathit{SimpleEv}$, $g \in G$ and $x \in \mathit{Var}$.

Formulae are evaluated over terms (defined above), pattern matching on the structure of the logic operators and over the transitions of the system. To illustrate the logic we give the semantics of the different versions of the $\langle\rangle$ operator. The complete semantics of FULL is given in [3].

$$
\begin{aligned}
t \models \langle a \rangle \Phi &= \exists t', t \xrightarrow{\mathrm{tt}\quad a} t' \text{ and } t' \models \Phi \\
t \models \langle \exists x\ g \rangle \Phi &= \exists v \text{ s.t.} \\
&\qquad \text{either } \exists t',\ t \xrightarrow{\mathrm{tt}\quad gv} t' \text{ and } t' \models \Phi_{[v/x]} \\
&\qquad \text{or } \exists t',\ t \xrightarrow{b\quad gz} t' \text{ and } b_{[v/z]} \equiv \mathrm{tt} \text{ and } \mathrm{t}'_{[\mathrm{v/z}]} \models \Phi_{[\mathrm{v/x}]} \\
t \models \langle \forall x\ g \rangle \Phi &= \forall v \\
&\qquad \text{either } \exists t', t \xrightarrow{\mathrm{tt}\quad gv} t' \text{ and } t' \models \Phi_{[v/x]} \\
&\qquad \text{or } \exists t',\ t \xrightarrow{b\quad gz} t' \text{ and } b_{[v/z]} \equiv \mathrm{tt} \text{ and } \mathrm{t}'_{[\mathrm{v/z}]} \models \Phi_{[\mathrm{v/x}]}
\end{aligned}
$$

where t is a term, z,$x \in \mathit{Var}$ and $v \in \mathit{Val}$.

The features of the logic are that the data and transition quantifiers are tightly tied together, with the data quantifier always coming first. That is, when proving a property holds we first choose the data at this step which satisfies the formula, and then we choose a transition to match that data. This logic captures all the discriminatory power of symbolic bisimulation, and with it we are able to express certain simple properties capturing ordering of events, and manipulation of data.

3 FULL*: Adding iteration operators to FULL

Clearly, a major drawback of FULL is that one cannot express properties that manipulate sequences of actions whose length is unknown. In turn this means essential properties of liveness and safety cannot be expressed, and yet FULL has a very desirable theoretical property; it captures exactly symbolic bisimulation over STS.

Here we propose some iteration operators to extend FULL to allow such properties to be expressed. The inspiration for the form of the new logic comes mainly from the work of Mateescu [14] on XTL, which is in turn influenced by the language of regular expressions. We believe this is a natural and familiar way to express repetition, making this logic more accessible to LOTOS practitioners. The novel contribution of our work is interpreting these operators over symbolic transition systems.

The basic innovation of both XTL and FULL* is to allow descriptions of properties of *paths* inside modal operators. A property over a path is expressed as a sequence of actions, possibly involving the repetition operators "$^+$" (1 or more repetitions) and "*" (zero or more repetitions). We also take this opportunity to rationalise the semantics of the FULL, introducing the $\neg$ operator and separating the definitions of quantification over data from that over transitions. In other ways the logic becomes more complex; the position of quantification has to be carefully considered.

Before presenting the formal details of FULL*, we present some examples to illustrate the form of the logic. Recall the simple buffer of the Introduction. Firstly we wish to write properties with sequences of events, such as

$$\langle(\mathtt{in} \cdot \mathtt{out})\rangle \mathrm{tt}$$

(i.e. one repetition of the actions `in` and `out`) but we also wish to repeat patterns of actions

$$\langle(\mathtt{in} \cdot \mathtt{out})^*\rangle \mathrm{tt}$$

(i.e. one or more repetitions of the actions `in` and `out`) Adding data we may write

$$\langle(\exists x \ \mathtt{in} \ x \cdot \exists y(x = y) \ \mathtt{out} \ y)^*\rangle \mathrm{tt}$$

which we expect to hold, or

$$\langle(\exists x \ \mathtt{in} \ x \cdot \exists y(x \neq y)\mathtt{out} \ y)^*\rangle \mathrm{tt}$$

which we expect *not* to hold since it says the buffer outputs a different value from that input.

3.1 FULL*

Definition 3. *FULL* Grammar*

$$\begin{aligned}
\Phi &::= b \mid \Phi_1 \wedge \Phi_2 \mid \langle R\rangle\Phi \mid \exists x(C)\Gamma \mid \neg\Phi \\
\Gamma &::= b \mid \Gamma_1 \wedge \Gamma_2 \mid \langle Q\rangle\Phi \mid \neg\Gamma \\
R &::= \alpha \mid \neg R \mid R_1 \wedge R_2 \mid R_1 \cdot R_2 \mid R^* \mid R^+ \mid \exists x(C)Q \\
Q &::= \alpha \mid \neg Q \mid Q_1 \wedge Q_2 \mid Q \cdot R \mid Q^* \mid Q^+
\end{aligned}$$

where b and C are Boolean expressions, and $\alpha \in SimpleEv \cup StructEv$.

Given these operators, $\vee$, $\forall$ and $\langle\ \rangle$ can all be defined as the duals of $\wedge$, $\exists$ and $[\,]$.

The grammar describes modal formulae: Φ and Γ, and patterns (path properties) inside modal operators: R and Q. Both Γ and Q are auxiliary definitions; they are identical to Φ and R except that they may not include quantification as their first operator. This is to make the handling of quantification simpler. By imposing this structure on the grammar we ensure that a quantifier is always followed directly by an instance of the data being bound. For example, we allow

$$\exists x\langle \mathtt{in} \ x\rangle \exists y\langle \mathtt{out} \ y\rangle \mathrm{tt}$$

but we do not allow

$$\exists x \exists y\langle \mathtt{in} \ x\rangle\langle \mathtt{out} \ y\rangle \mathrm{tt}$$

Although an artificial constraint to make the association of quantifiers and variables more straightforward this does not seem unreasonable since the data must be associated with an action in any case, and this will not affect the expressivity of the logic.

Expressions within the logic FULL* are derived by beginning with Φ. We now explain the meaning of these expressions with respect to a Symbolic Transition System, via terms.

We split the definition of the semantics into four parts corresponding to Φ, Γ, R and Q. In each part the semantics is given inductively over the structure of the logical formulae. The definitions are mutually recursive.

Definition 4. *FULL* Semantics: Φ*

$$\begin{array}{lcl}
t \models b & = & b \\
t \models \Phi_1 \wedge \Phi_2 & = & t \models \Phi_1 \ \wedge t \models \Phi_2 \\
t \models \langle R\rangle\Phi & = & \exists\pi \ s.t. \ \mathit{first}(\pi) = t \ \mathit{and} \ \pi \models R \ \wedge \mathit{last}(\pi) \models \Phi \\
t \models \exists x(C)\Gamma & = & \exists v : \mathit{Val} \ s.t. \ C[v/x] \equiv \mathrm{tt} \ \wedge \ \mathrm{t, v} \models \Gamma[\mathrm{v/x}] \\
t \models \neg\Phi & = & t \not\models \Phi
\end{array}$$

Note in particular that the rule for existential quantification requires satisfaction via the rules for Γ expressions. A value has been chosen and must be carried into the next stage of the evaluation so it can be tied to a corresponding datum in the transition system. This binding is not possible in the Φ rules since there may be several transitions (with different actions) from t and only the first part of Γ will determine which particular action is of interest.

Γ expressions are evaluated over a pair (term, value), but note that Γ has already had the appropriate substitution applied $[v/x]$.

Definition 5. *FULL* Semantics: Γ*

$$\begin{array}{lcl}
t, v \models b & = & b \\
t, v \models \Gamma_1 \wedge \Gamma_2 & = & t, v \models \Gamma_1 \ \mathit{and} \ t, v \models \Gamma_2 \\
t, v \models \langle Q\rangle\Phi & = & \exists\pi \ s.t. \ \mathit{first}(\pi) = t \ \wedge \ \pi_\sigma \models Q \ \wedge \ \mathit{last}(\pi)_\sigma \models \Phi \\
t, v \models \neg\Gamma & = & t, v \not\models \Gamma
\end{array}$$

where $\sigma = [v/z]$ if $\mathit{first\_data}(\pi) = z$, empty otherwise.

The auxiliary function $\mathit{first\_data}(\pi)$ is defined as:

$$\begin{array}{ll}
\mathit{first\_data}(\pi) = & \text{if } \mathit{init}(\pi) = t_1 \xrightarrow{b \quad gz} t_2 \text{ then } z \\
 & \text{if } \mathit{init}(\pi) = t_1 \xrightarrow{b \quad gw} t_2 \text{ then } w
\end{array}$$

As soon as the path is chosen in the rule for $\langle Q\rangle$ it is possible to match v with a corresponding datum in the transition system. Recall that we distinguish between variables (z) and data values (w). This is done both for the evaluation of Q and for the evaluation of the next section of modal formula Φ. The latter is important, since if there is no substitution here then any matching done in Q is forgotten, and the value v is not propagated. This will be illustrated in Section 4. If the datum returned by $\mathit{first\_data}()$ is a value w, then no substitution is applied in the $\langle Q\rangle\Phi$ rule above. This is because either $v = w$ and the substitution has no effect, or $v \neq w$ and applying a substitution would mean overwriting one value with another, possibly leading to erroneous conclusions.

We now define formulae over paths described by R. We assume that any path chosen in the previous steps is an exact match for the length of R (so we don't have to use $init(\pi)$ for example to extract the first part). This is essential. Consider the case where a long path is extracted to match only a single action R. The rules then ask us to continue evaluating from $last(\pi)$. Clearly this misses out large chunks of behaviour and is undesirable.

Definition 6. *FULL* Semantics: R*

$$
\begin{array}{lll}
\pi \models \alpha & = & \pi = t_1 \xrightarrow{tt \quad \alpha_1} t_2 \wedge \ match(\alpha, \alpha_1) \\
\pi \models \neg R & = & \pi \not\models R \\
\pi \models R_1 \wedge R_2 & = & \pi \models R_1 \ \wedge \pi \models R_2 \\
\pi \models R_1 \cdot R_2 & = & \pi = \pi_1 \pi_2 \ \wedge \ \pi_1 \models R_1 \ \wedge \ \pi_2 \models R_2 \\
\pi \models R^* & = & \pi = \pi_1 \cdots \pi_p, \ p \geq 0 \wedge \forall i.1 \leq i \leq p. \ \pi_i \models R \\
\pi \models R^+ & = & \pi = \pi_1 \cdots \pi_p, \ p > 0 \wedge \forall i.1 \leq i \leq p. \ \pi_i \models R \\
\pi \models \exists x(C)Q & = & \exists v \ s.t. \ C[v/x] \equiv \mathrm{tt} \ \wedge \ \pi_\sigma \models \mathrm{Q[v/x]}
\end{array}
$$

where p is an integer, and $\sigma = [v/z]$ if first_data$(\pi) = z$, empty otherwise.

The *match* function takes two actions, matching the names of the actions, and data if there is data associated with the first action, ignoring it otherwise. This allows inexact matching of actions. If the logic specifies both gate and data then the transition system must match that exactly, but if only a gate is given in the logic then any data in the transition system is ignored.

Lastly we define formulae over Q patterns. Due to substitutions applied above α here can only be of the form gv if the formula is well formed (there must be data, since the first part of a Q formula has to match a quantifier).

Definition 7. *FULL* Semantics: Q*

$$
\begin{array}{lll}
\pi \models \alpha & = & \pi = t_1 \xrightarrow{\mathrm{tt} \quad \alpha} t_2 \\
\pi \models \neg Q & = & \pi \not\models Q \\
\pi \models Q_1 \wedge Q_2 & = & \pi \models Q_1 \ \wedge \pi \models Q_2 \\
\pi \models Q \cdot R & = & \pi = \pi_1 \pi_2 \ \wedge \ \pi_1 \models Q \ \wedge \ \pi_2 \models R \\
\pi \models Q^* & = & \pi = \pi_1 \cdots \pi_p, \ p \geq 0 \wedge \forall i.1 \leq i \leq p. \ \pi_i \models Q \\
\pi \models Q^+ & = & \pi = \pi_1 \cdots \pi_p, \ p > 0 \wedge \forall i.1 \leq i \leq p. \ \pi_i \models Q
\end{array}
$$

It can be easily demonstrated all the properties which can be written with FULL can be expressed in FULL* just by not considering iterative mechanisms and by restricting the position of quantification. Therefore FULL* has at least the expressive power of FULL. More importantly, the addition of iterative operators give the extra flexibility required to express safety and liveness properties, as can be seen in the next section.

4 Examples of the use of FULL*

The semantics presented above are complex, so we present some examples in detail. We sketch one example proof, but the remainder are left as exercises for the reader.

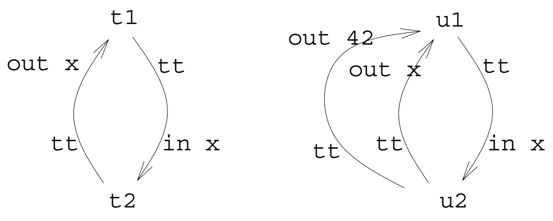

Fig. 2. Buffer processes t and u

A Simple Buffer We return again to the buffer example. Given the transition systems of Figure 2 we can say

$$t_1 \models \langle(\exists x \; \texttt{in} \; x \cdot \texttt{out} \; x)^*\rangle \mathrm{tt}$$

In fact this is rather weak, since * expressions can be satisfied vacuously.

$$t_1 \models \langle(\exists x \; \texttt{in} \; x \cdot \texttt{out} \; x)^+\rangle \mathrm{tt}$$

is better, but still only forces a single iteration of the pattern. A better formulation for this *liveness property* is

$$t_1 \models \neg\langle\neg(\exists x \; \texttt{in} \; x \cdot \texttt{out} \; x)^+\rangle \mathrm{tt}$$

i.e. it is not possible that the buffer does not perform a cycle of `in` and `out` actions. This property also holds for u_1 which is not the usual sort of buffer, since it sometimes outputs the value 42, regardless of the input, so to fully describe the buffer something stronger has to be said:

$$t_1 \models [(\exists x \; \texttt{in} \; x \cdot \exists y (x \neq y) \texttt{out} \; y)^+] \mathrm{ff}$$

This formula says that the pattern inside the box operator is not possible (followed by ff which cannot be satisfied). This is true for t since the pattern says it is possible to output a value different from the one input. *Safety properties* will in general have this form; however, the presence of the box operator means the first section could be vacuously true, so it is necessary to always combine this sort of property with ones expressing that some appropriate behaviour is possible.

Obviously,

$$u_1 \not\models [(\exists x \; \texttt{in} \; x \cdot \exists y (x \neq y) \texttt{out} \; y)^+] \mathrm{ff}$$

since u_1 can output `42`, regardless of the input value.

To focus on the importance of the position of quantifiers, and how substitution works in the rules for FULL*, consider the following

$$t \models [(\exists x \; \texttt{in} \; x)][\exists y (x = y) \texttt{out} \; y] tt$$

This seems to be a desirable property (similar to that above) but we have no way of giving a meaning to it. This is because the value of x is "forgotten" outside of its modal quantifier, so the expression $x = y$ cannot be evaluated. So, if properties are to relate different quantified variables, then those variables must be bound inside the same modal operator. Conversely, if we use one variable across a number of modal operators then it must be bound *outside* the modal operators. For example,

$$t_1 \models \exists x \langle \mathtt{in}\ x \rangle \langle \mathtt{out}\ x \rangle \mathrm{tt}$$

To evaluate this we obviously need to pass whatever value x was bound to into the evaluation of $\langle \mathtt{out}\ x \rangle \mathrm{tt}$. This means substitution both in the formula itself and in the transition system (see the rule for $\exists x \langle Q \rangle$ in Definition 4).

In the first unfolding we get

$$\exists v : \mathit{Val} \text{ s.t. } t_1, v \models \langle \mathtt{in}\ v \rangle \langle \mathtt{out}\ v \rangle \mathrm{tt}$$

where the value v is substituted for the variable x throughout the formula, followed by

$$\exists v : \mathit{Val} \text{ s.t. } \exists \pi\ \mathit{first}(\pi) = t_1\ \wedge\ \pi_\sigma \models \mathtt{in}\ v\ \wedge\ \mathit{last}(\pi)_\sigma \models \langle \mathtt{out}\ v \rangle \mathrm{tt}$$

in which σ has to carry the mapping information $[v/y]$ for the transition system. Otherwise $\mathit{last}(\pi)$ will not be a suitable model for $\langle \mathtt{out}\ v \rangle \mathrm{tt}$ since y would not be bound to any value.

Finally, consider the property

$$\exists x \langle (\mathtt{in}\ x \cdot \mathtt{out}\ x)^* \rangle \mathrm{tt}$$

This is unsatisfactory because it allows only one value to ever be input (repeatedly). This highlights a feature of the logic; in repeated patterns the quantifiers are most likely to be inside the repetition inside a modal operator, allowing new values in each iteration. Also, as was seen above, the scope of a quantifier inside an operator is just that operator, so many properties will consist only of one or two modal operators but with more complex patterns inside.

Communications Consider the classic communications protocol Alternating Bit Protocol (ABP). The idea is that a producer and consumer of data are communicating over lossy channels, therefore to ensure data sent is received some acknowledgements are set up. The protocol itself uses a sender and a receiver, and data is sent with a single bit tag. Each of the sender and receiver maintain a record of the tag, or what the tag ought to be in the case of the receiver, to help detect errors. For example, if the sender sends (d, b) and the receiver gets (d, b'), where b' denotes the inverse of b, then the receiver knows there has been an error and does not send its acknowledgement.

From the outside, the ABP behaves just like the buffer: data items go in and come out reliably, therefore all of the properties described above can be applied if internal behaviour of the protocol is ignored.

Consider first of all a view of the sending and receiving messages. The protocol behaviour says that any particular message is sent repeatedly until correctly received. This *reachability property* is expressed:

$$abp \models \exists m \langle (\texttt{send } m)^* \cdot \texttt{receive } m \rangle \mathrm{tt}$$

From the sender's point of view, the successful `receive` is not apparent until an acknowledgement with the correct tag arrives, so we might require the following property of the sender:

$$sender \models \langle \exists (d,b)\ (\texttt{send } (d,b))^* \cdot \texttt{ack } b \rangle \mathrm{tt}$$

We take the liberty of using a pair type to represent the data and the tag sent.

We can be even more explicit about the behaviour at the sender end:

$$sender \models \langle \exists (d,b)\ (\texttt{send } (d,b) \cdot (\texttt{ack } err\ \vee\ \texttt{ack } b'))^* \cdot \texttt{ack } b \rangle \mathrm{tt}$$

reflecting the repetition of the pattern that a message is sent, and acknowledgement received (possibly of the error type, possibly with the wrong tag), until a correct acknowledgement is received.

Taking an even longer view, we can encode that the sender, once a message is succesfully sent, switches to using the tag b'. If the above pattern inside the $\langle\ \rangle$ operator is named *p_b* and the property with b and b' swapped named *p_not_b* then

$$sender \models \langle ((p\_b) \cdot (p\_not\_b))^* \rangle \mathrm{tt}$$

Now we look more generally at some communication based examples. Consider that the values sent have a particular sequence, in particular, that the value sent increases from each sending to its successor:

$$cp \models [(\forall x (\texttt{send } x \cdot \neg(\exists y\ (y < x) \texttt{send } y)))^*]\mathrm{tt}$$

That is, for all finite paths π beginning at *cp*, π can be segmented into $\pi_1 \cdots \pi_p$ where $\pi_i \models \forall x (\texttt{send } x \cdot \neg(\exists y\ (y < x) \texttt{send } y))$. However, if π is grouped into consecutive pairs starting with the first send, then we say nothing about the relationship between the second and the third sending. A better formulation may be

$$\begin{aligned} &\forall z\ [\texttt{send } z][(\neg(\exists x\ (x < z) \texttt{send } x)) \\ &\wedge\ (\forall x (\texttt{send } x \cdot \neg(\exists y\ (y < x) \texttt{send } y)))^*]\mathrm{tt} \end{aligned}$$

effectively staggering the property to consider alternate pairs (2,3), (4,5) and so on, as well as pairs (1,2), (3,4), etc.

Mutual Exclusion In the classic problem of mutual exclusion there are a number of processes executing which have *critical* and *non-critical* sections. It is required that only one process access its critical section at any one time. A master process therefore controls entry and exit from critical sections. A process

i may enter (`InCritical` i) or leave (`OutCritical` i) its critical section. The correctness property for mutual exclusion is then

$$\forall i\ [\texttt{InCritical}\ i]\ \forall j (j \neq i)\ [(\neg\texttt{InCritical}\ j)^*\ \texttt{OutCritical}\ i]\mathrm{tt}$$

Assume processes request permission to enter their critical section `Ask_Critical` i, and then `Wait` until permission is granted. Processes are guaranted access to their critical section by:

$$\forall j\ [(\texttt{Ask\_Critical}\ j\ \cdot (\texttt{Wait}\ j)^*\ \cdot \texttt{InCritical}\ j)^*]\mathrm{tt}$$

Note the use of nested occurrences of the * operator.

As with all [ ] formulae care must be taken to also express properties which can possibly do something, to avoid the case that the [ ] formula is satisfied vacuously.

Summary Some rules of thumb for expressing properties can be given:

- *Safety properties*: express that bad things do not happen, therefore can be described in the form $[\texttt{bad\_action}]\mathrm{ff}$.
- *Liveness properties*: express that good things do happen, therefore can be expressed in the form $\neg\langle\neg(\texttt{good\_action})^+\rangle\mathrm{tt}$.
- *Reachability properties*: eventually it is possible to do the good action desired, ignoring some sequence of other actions which may come first, can be expressed as $\langle(\texttt{other\_pattern})^* \cdot \texttt{good\_action}\rangle\mathrm{tt}$.
- *Response properties*: are similar to reachability properties. Either may also use the stronger pattern $[(\texttt{other\_pattern})^*]\langle\texttt{good\_action}\rangle\mathrm{tt}$ to indicate that on *all* paths from the current state the desired action is attainable.

5 Comparison with FULL

Clearly, FULL* has more expressive power than FULL. In particular, FULL* can distinguish between processes which are symbolically bisimilar.

Given the three bisimilar [4] symbolic transition systems of Figure 3, we can find FULL* formulae to distinguish them as follows:

$$\begin{aligned} a1 &\models [\forall x\ \texttt{send x}]tt \\ b1 &\not\models [\forall x\ \texttt{send x}]tt \\ b1 &\models [(\exists x(x \leq 4)\ \texttt{send x}]tt \\ c1 &\not\models [(\exists x(x \leq 4)\ \texttt{send x}]tt \end{aligned}$$

The crucial point in the formulation of the properties above is that the transition is chosen first (which may additionally constrain the data). This gives us the opportunity to choose the “wrong” transition. Symbolic bisimulation on the other hand allows us to ignore to some extent the way data is distributed across transitions. As long as the same set of data is used in total, then we can find a matching transition in the other process. In the logic, this translates to tightly associating quantifiers over data and those over transitions. In particular, the data quantifier *must* precede the associated transition quantifier to maintain the link with symbolic bisimulation. FULL* allows quantifiers to occur in any order.

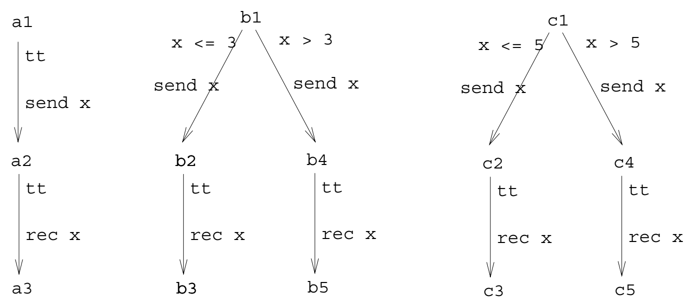

Fig. 3. Bisimilar processes A, B and C

6 Conclusion

We have presented an extension to the FULL logic with iteration operators. These operators allow properties over paths whose length is unknown to be expressed, something that was impossible with the original logic. The form of these operators was derived from XTL [14], extended with explicit quantifiers over data and interpreted over symbolic transition systems. In FULL* fundamental verification properties such as liveness, safety and response can be defined, as illustrated in Section 4. This is important since the uptake of formal methods for description of systems relies on the expressivity and ease of use of the language for realistic problems. Another key factor is automated analysis, and one of our goals in developing the symbolic framework is to make analysis more tracable by reducing the state space of the system.

The next step is to validate this logic by studying "real-life" examples of LOTOS specification using data, aided by a prototype model checking implementation in CADP. This does not reflect the symbolic nature of the transitions here since CADP is built on Binary Coded Graph representation, but the prototype is useful for gaining confidence in the logic. A larger example has been completed using the benchmark RPC case study [1]. The properties expressed concern the ability of the memory to respond appropriately to a call, and are rather similar in form to those given for the communications protocol above, therefore we do not repeat them here. We intend to carry out further case studies to demonstrate the expressivity and useful features of FULL*.

References

1. M. Broy, S. Merz, and K. Spies, editors. *Formal Systems Specification: The RPC-Memory Specification Case Study.* Number 1169 in Lecture Notes in Computer Science. Springer-Verlag, 1996.
2. M. Calder, S. Maharaj, and C. Shankland. An Adequate Logic for Full LOTOS. In J. Oliveira and P. Zave, editors, *Formal Methods Europe'01*, LNCS 2021, pages 384–395. Springer-Verlag, 2001.

3. M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Full LOTOS based on Symbolic Transition Systems. *The Computer Journal*, 45(1):55–61, 2002.
4. M. Calder and C. Shankland. A Symbolic Semantics and Bisimulation for Full LOTOS. In M. Kim, B. Chin, S. Kang, and D. Lee, editors, *Proceedings of FORTE 2001, 21st International Conference on Formal Techniques for Networked and Distributed Systems*, pages 184–200. Kluwer Academic Publishers, 2001.
5. M. Aguilar Cornejo, H. Garavel, R. Mateescu, and N. de Palma. Specification and Verification of a Dynamic Reconfiguration Protocol for Agent-Based Applications. Technical Report 4222, INRIA, 2001.
6. H. Eertink. *Simulation Techniques for the Validation of LOTOS Specifications*. PhD thesis, University of Twente, 1994.
7. J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (CAESAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In R. Alur and T.A. Henzinger, editors, *Proceedings of CAV'96*, number 1102 in Lecture Notes in Computer Science, pages 437–440. Springer-Verlag, 1996.
8. H. Garavel and F. Lang. NTIF: A General Symbolic Model for Communicating Sequential Processes with Data. In D.A. Peled and M.Y. Vardi, editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2002*, LNCS 2529, pages 276–291. Springer-Verlag, 2002.
9. M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.
10. M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.
11. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
12. International Organisation for Standardisation. *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1988.
13. L. Logrippo, M. Faci, and M. Haj-Hussein. An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems*, 23:325–342, 1992.
14. R. Mateescu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, Institut National Polytechnique de Grenoble, 1998.
15. R. Mateescu and H. Garavel. XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. In *Proceedings of the International Workshop on Software Tools for Technology Transfer STTT'98 (Aalborg, Denmark)*, 1998.
16. R. Mateescu and M. Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. In *Proceedings of the 5th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2000 (Berlin, Germany)*, 2000. Full version available as INRIA Research Report RR-3899.
17. M. Sighireanu and R. Mateescu. Verification of the Link Layer Protocol of the IEEE-1394 Serial Bus (FireWire): an Experiment with E-LOTOS. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(1):68–88, Dec. 1998.