

# Verification of Parameterized Systems by Dynamic Induction on Diagrams <sup>\*</sup>

Zohar Manna and Henny B. Sipma

Computer Science Department  
Stanford University  
Stanford, CA. 94305-9045  
{`manna,sipma`}@cs.stanford.edu

**Abstract.** In this paper we present a visual approach to proving progress properties of parameterized systems using induction on verification diagrams. The inductive hypothesis is represented by an automaton and is based on a state-dependent order on process indices, for increased flexibility. This approach yields more intuitive proofs for progress properties and simpler verification conditions that are more likely to be proved automatically.

## 1 Introduction

*Verification diagrams* represent a proof that a reactive system satisfies its temporal specification; they were proposed in [MP94] and generalized in [BMS95]. The purpose of a diagram is to provide a high-level proof outline that makes explicit the difficult parts of the proof, while hiding the tedious details.

Parameterized systems are systems that consist of an unbounded number of processes that differ only in their process identifiers (process indices). Proofs of safety properties over parameterized systems introduce universal force quantifiers in the verification conditions. On the other hand, proofs of progress properties for such systems usually introduce both universal force and existential force quantifiers in the verification conditions, making these proofs considerably harder.

The validity of a progress property usually relies on the fairness of certain transitions. In the proof, these transitions must be identified, and they are represented explicitly in a verification diagram. However, for parameterized systems, the validity of a progress property may depend on an unbounded number of distinct fair transitions, so an alternative representation must be used.

One solution, proposed in [MP96], is to assert the existence of such transitions in the diagram without explicitly identifying them. However, this partly defeats

---

<sup>\*</sup> This research was supported in part by the National Science Foundation under grant CCR-98-04100, ARO under grants DAAH04-95-1-0317, DAAH04-96-1-0122 and DAAG55-98-1-0471, ARO under MURI grant DAAH04-96-1-0341, and by Army contract DABT63-96-C-0096 (DARPA).

the purpose of diagrams: the transitions now have to be identified at the theorem-proving level, by instantiating the existential quantifiers. This usually requires a substantial amount of user input at a level where intuition and insight into the program are of little help.

In this paper, we suggest that progress properties can often be proven by induction on the process identifier. In many programs, processes are waiting for each other to achieve their goal in turn. A natural inductive hypothesis is thus that processes with higher priority than the current process will achieve their goal. In some cases, for example in the proof of progress properties for a leader election algorithm presented in [BLM97], standard mathematical induction with a fixed order on the process indices suffices. However, in many cases a more flexible order is required.

The induction principle for diagrams proposed here extends the regular induction principle over the natural numbers by allowing a state-dependent order on the process indices. While in a proof of  $\varphi[i]$  by regular induction,  $\varphi[j]$  may be assumed only if  $j \prec i$ , in our diagram induction  $\varphi[j]$  may be assumed if for every computation of the system *eventually* always  $j \prec i$  holds. In a proof by diagram induction, this condition is incorporated in an automaton for the inductive hypothesis that constrains the set of computations for an arbitrary value of the parameter; this automaton is then conjoined with the main diagram.

We illustrate the diagram induction principle by proving a progress property for a very simple parameterized system. In the last section we demonstrate a more complex application in the proof of accessibility for a fine-grained parameterized algorithm for mutual exclusion.

## 2 Preliminaries

### 2.1 Computational Model

Our computational model is that of *fair transition systems* [MP95]. A fair transition system (FTS)  $\mathcal{S} : \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle$  consists of

- $V$ : A finite set of typed *system variables*. A *state* is a type-consistent interpretation of the system variables. The set of all states is called the *state space* and is designated by  $\Sigma$ . A first-order formula with free variables in  $V$  is called an *assertion*. We write  $s \models p$  if  $s$  satisfies  $p$ .
- $\Theta$ : The *initial condition*, an assertion characterizing the initial states.
- $\mathcal{T}$ : A finite set of *transitions*. Each transition  $\tau \in \mathcal{T}$  is a function  $\tau : \Sigma \mapsto 2^\Sigma$  mapping each state  $s \in \Sigma$  into a (possibly empty) set of  $\tau$ -successor states,  $\tau(s) \subseteq \Sigma$ . Each transition  $\tau$  is defined by a *transition relation*  $\rho_\tau(V, V')$ , a first-order formula in which the unprimed variables refer to the values in the current state  $s$ , and the primed variables refer to the values in the next state  $s'$ . Transitions may be parameterized, thus representing a possibly unbounded set of transitions differing only in their parameter.
- $\mathcal{J} \subseteq \mathcal{T}$ : A set of *just* (weakly fair) transitions<sup>1</sup>.

<sup>1</sup> To simplify the presentation we omit compassion (strong fairness) in this paper.

A *run* of a fair transition system  $\mathcal{S} : \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle$  is an infinite sequence of states  $\sigma : s_0, s_1, s_2, \dots$ , such that  $s_0 \models \Theta$ , and for each  $j \geq 0$ ,  $s_{j+1}$  is a  $\tau$ -*successor* of  $s_j$ , that is,  $s_{j+1} \in \tau(s_j)$  for some  $\tau \in \mathcal{T}$ . If  $s_{j+1}$  is a  $\tau$ -successor of  $s_j$  we say that transition  $\tau$  is *taken* at position  $j$ . The set of runs of  $\mathcal{S}$  is written  $\mathcal{L}_R(\mathcal{S})$ .

A *computation* of a fair transition system  $\mathcal{S}$  is a run  $\sigma$  of  $\mathcal{S}$  that satisfies the *fairness requirement*: for each transition  $\tau \in \mathcal{J}$  it is not the case that  $\tau$  is continuously enabled beyond some position  $j$  in  $\sigma$ , but not taken beyond  $j$ . The set of computations of a system  $\mathcal{S}$  is denoted by  $\mathcal{L}(\mathcal{S})$ , called the *language* of  $\mathcal{S}$ . A state is called  $\mathcal{S}$ -*accessible* if it appears in some computation of  $\mathcal{S}$ .

### Example 1

Figure 1 shows program SIMPLE, parameterized by  $M$ , written in the SPL language of [MP95]. Its body is the parallel composition of  $M$  processes, indexed by  $i$ . The program has a global array variable  $a$  that can be observed by all processes, all of whose elements are initialized to *false*. In addition, each process has a local variable  $j$  that cannot be observed by any other process.

It is straightforward to translate this program into an FTS. The system variables are an integer  $M$ , a boolean array  $a$ , an integer array  $j$ , containing the value of the local variable  $j$  of each process, and an array  $\pi$ , containing the label,  $\pi[i] \in \{\ell_0, \ell_1, \ell_2, \ell_3\}$ , of the current statement of each process. Each program statement can be represented by a parameterized transition. For example, the statement labeled by  $\ell_1$  is represented by the parameterized transition with transition relation

$$\rho_{\ell_1}[i] : \pi[i] = \ell_1 \wedge \pi'[i] = \ell_0 \wedge (a[j[i]] \vee i \leq j[i]) \wedge a' = a \wedge j' = j$$

The objective of this simple program is for each process  $i$  to set  $a[i]$  to *true*, but

```

in     $M$  : integer where  $M > 0$ 
local  $a$  : array [1.. $M$ ] of boolean where  $a = \text{false}$ 

     $\parallel_{i=1}^M P[i] :: \left[ \begin{array}{l} \text{local } j : \text{integer where } j = 1 \\ \ell_0: \text{for } j = 1 \text{ to } M \text{ do} \\ \ell_1: \text{await } a[j] \vee i \leq j \\ \ell_2: a[i] := \text{true} \\ \ell_3: \end{array} \right]$ 

```

**Fig. 1.** Program SIMPLE

only after all processes with smaller process indices have done so. We will prove that eventually each process completes its mission. ■

## 2.2 Specification Language

As specification language we use *linear-time temporal logic* (LTL). LTL formulas are interpreted over infinite sequences of states. A temporal formula [MP95] is constructed out of state formulas and temporal operators. Below we only give the semantics of those operators used in our examples.

For an infinite sequence of states  $\sigma : s_0, s_1, \dots$ , an assertion  $p$ , and temporal formulas  $\varphi$  and  $\psi$ ,

$$\begin{aligned} (\sigma, j) \models p & \quad \text{iff } s_j \models p \text{ that is, } p \text{ holds on state } s_j \\ (\sigma, j) \models \Box \varphi & \quad \text{iff } (\sigma, i) \models \varphi \text{ for all } i \geq j \\ (\sigma, j) \models \Diamond \varphi & \quad \text{iff } (\sigma, i) \models \varphi \text{ for some } i \geq j \end{aligned}$$

An infinite sequence of states  $\sigma$  *satisfies* a temporal formula  $\varphi$ , written  $\sigma \models \varphi$ , if  $(\sigma, 0) \models \varphi$ . Given an FTS  $\mathcal{S}$ , we say that an LTL formula  $\varphi$  is  $\mathcal{S}$ -*valid*, written  $\mathcal{S} \models \varphi$ , if every computation of  $\mathcal{S}$  satisfies  $\varphi$ .

The *safety closure*[AL90] of a temporal formula  $\varphi$ , is the smallest safety property,  $\varphi_S$  such that  $\varphi$  implies  $\varphi_S$ . For example  $(\Box \varphi)_S = \Box \varphi$  and  $(\Diamond \varphi)_S = \text{true}$ .

### Example 2

The temporal formula

$$\psi[i] : \Box \Diamond \neg \ell_1[i] \rightarrow \Diamond \Box a[i] ,$$

parameterized by  $i$ , states that if process  $i$  is not in location  $\ell_1$  infinitely often, then array element  $a[i]$  will eventually become *true* and stay *true*. ■

## 3 Verification Diagrams

A verification diagram  $\mathcal{G}$  represents a proof that a possibly infinite-state system  $\mathcal{S}$  satisfies a property  $\varphi$  if it can be shown that  $\mathcal{G}$  is both an overapproximation of the system and an underapproximation of the property. In other words,

$$\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\varphi)$$

where  $\mathcal{L}(\mathcal{S})$ ,  $\mathcal{L}(\mathcal{G})$ , and  $\mathcal{L}(\varphi)$  denote the languages of the system, diagram and property, respectively, each of which is a set of infinite sequences of states.

The language inclusion  $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G})$ , which states that every computation of  $\mathcal{S}$  is a model of the diagram  $\mathcal{G}$ , is justified by proving a set of first-order verification conditions, using deductive techniques. On the other hand, the inclusion  $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\varphi)$ , which states that every model of the diagram satisfies the property, is a decidable language inclusion check that can be established automatically using language-inclusion algorithms for  $\omega$ -automata. Thus, verification diagrams reduce the proof of an arbitrary temporal property over a system to the proof of a set of first-order verification conditions and an algorithmic check.

### 3.1 Definition

Verification diagrams are  $\omega$ -automata [Tho90] augmented with an additional node labeling  $\mu$ , to establish their relation with the FTS that they verify. The diagrams used in this paper are a modified version of those presented in [BMS95] and are described in detail in [MBSU98].

A diagram  $\mathcal{G} : \langle N, N_0, E, \mu, \nu, \mathcal{F} \rangle$  over an FTS  $\mathcal{S} : \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle$  and a property  $\varphi$  consists of the following components:

- $N$ : a finite set of nodes;
- $N_0 \subseteq N$ : a set of initial nodes;
- $E \subseteq N \times N$ : a set of edges connecting nodes;
- $\mu$ : a node labeling function that assigns to each node an assertion over  $V$ ;
- $\nu$ : a node labeling function, called the *property labeling*, that assigns to each node a boolean combination of the atomic assertions appearing in  $\varphi$ ;
- $\mathcal{F} \subseteq 2^{2^N}$ : a (Müller) acceptance condition given by a set of set of nodes.

A *path* of a diagram is an infinite sequence of nodes  $\pi : n_0, n_1, \dots$ , such that  $n_0 \in N_0$  and for each  $i \geq 0$ ,  $\langle n_i, n_{i+1} \rangle \in E$ . Given a path  $\pi$ , its *limit set*, written  $\text{inf}(\pi)$ , is the set of nodes that occur infinitely often in  $\pi$ . Note that the limit set of an infinite path must be nonempty since the diagram is finite, and that it must be a strongly connected subgraph (SCS) of the diagram. A path  $\pi$  of a diagram is *accepting* if  $\text{inf}(\pi) \in \mathcal{F}$ .

Given an infinite sequence of states  $\sigma : s_0, s_1, \dots$ , a path  $\pi : n_0, n_1, \dots$  is called a *trail* of  $\sigma$  in the diagram if  $s_i \models \mu(n_i)$  for all  $i \geq 0$ . A sequence of states  $\sigma$  is a *run* of a diagram if there exists a trail of  $\sigma$  in the diagram. The set of runs of a diagram  $\mathcal{G}$  is written  $\mathcal{L}_R(\mathcal{G})$ . A sequence of states  $\sigma : s_0, s_1, \dots$  is a *model* of a diagram if there exists an accepting trail of  $\sigma$  in the diagram. The set of models of a diagram  $\mathcal{G}$  is written  $\mathcal{L}(\mathcal{G})$ .

### 3.2 Verification Conditions

Associated with a diagram is a set of first-order verification conditions that, if valid, prove

$$\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G}) .$$

In this case we say that  $\mathcal{G}$  is  *$\mathcal{S}$ -valid*. We use the following notation:

For a set of nodes  $M = \{n_0, \dots, n_k\}$ , we define

$$\mu(M) \stackrel{\text{def}}{=} \mu(n_0) \vee \dots \vee \mu(n_k)$$

where  $\mu(\{\}) = \text{false}$ . For a node  $n$ , the set of successor nodes of  $n$  is  $\text{succ}(n)$ . We use *Hoare triple* notation to state that a parameterized transition  $\tau$  leads from a state satisfying  $\varphi$  to a state satisfying  $\psi$ :

$$\{\varphi\} \tau \{\psi\} \stackrel{\text{def}}{=} \forall i . (\varphi \wedge \rho_\tau[i] \rightarrow \psi')$$

A diagram  $\mathcal{G}$  is  *$\mathcal{S}$ -valid* if it satisfies the following conditions:

- *Initiation*: Every initial state of  $S$  must be covered by some initial node of the diagram, that is  $\Theta \rightarrow \mu(N_0)$ .
- *Consecution*: For every node  $n$  and every transition  $\tau$ , there is a successor node that can be reached by taking  $\tau$ , that is

$$\{ \mu(n) \} \tau \{ \mu(\text{succ}(n)) \} .$$

The Initiation and Consecution conditions, if valid, prove that every run of  $S$  is a run of the diagram, that is,  $\mathcal{L}_R(S) \subseteq \mathcal{L}_R(\mathcal{G})$

A second set of verification conditions ensures that every computation of the system has an accepting trail in the diagram. Thus, if an SCS  $S$  is not accepting, we must show that computations can always leave  $S$  or cannot stay in  $S$  forever.

We say that an SCS  $S$  has a *fair exit transition*  $\tau$ , if the following verification conditions are valid for every node  $m \in S$

$$\mu(m) \rightarrow \text{enabled}(\tau) \quad \text{and} \quad \{ \mu(m) \} \tau \{ \mu(\text{succ}(m) - S) \} ,$$

that is,  $\tau$  is enabled on every node in  $S$ , and from every node in  $S$  transition  $\tau$  can be taken to leave  $S$ . Thus if an SCS has a fair exit transition, there is at least one trail of every computation that can leave the SCS.

We say that an SCS  $S : \{n_1, \dots, n_k\}$  is *well-founded* if there exist ranking functions  $\{\delta_1, \dots, \delta_k\}$ , mapping the system states into a well-founded domain  $(\mathcal{D}, \succ)$ , such that the following verification conditions hold: there is a *cut-set*<sup>2</sup>  $C$  of edges in  $S$  such that for all edges  $\langle n_1, n_2 \rangle \in C$  and every transition  $\tau$ ,

$$\mu(n_1) \wedge \rho_\tau \wedge \mu'(n_2) \rightarrow \delta_1(n_1) \succ \delta_2'(n_2) ,$$

and for all other edges  $\langle n_1, n_2 \rangle \notin C$  in  $S$  and for all transitions  $\tau$ ,

$$\mu(n_1) \wedge \rho_\tau \wedge \mu'(n_2) \rightarrow \delta_1(n_1) \succeq \delta_2'(n_2) .$$

Thus, if an SCS  $S$  is well-founded, no run can have a trail with limit set  $S$ , since it would violate the well-founded order.

- *Acceptance*: Every nonaccepting SCS  $S$  ( $S \notin \mathcal{F}$ ), has a fair exit transition or is well-founded.

The Acceptance condition ensures that every computation of the system has at least one accepting trail in the diagram, that is,  $\mathcal{L}(S) \subseteq \mathcal{L}(\mathcal{G})$ .

### 3.3 Property Satisfaction

It remains to justify that  $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\varphi)$ , which is done using the property labeling  $\nu$  of the diagram. Recall that the property labeling assigns to each node of the diagram a boolean combination of atomic assertions appearing in  $\varphi$ , the property to be proven.

<sup>2</sup> A cut-set of an SCS  $S$  is a set of edges  $C$  such that the removal of  $C$  from  $S$  results in a subgraph that is not strongly connected.

We say that a path  $\pi : n_0, n_1, \dots$  is a *property trail* of an infinite sequence of states  $\sigma : s_0, s_1, \dots$  if for all  $i \geq 0$ ,  $s_i \models \nu(i)$ . An infinite sequence of states  $\sigma$  is a *property model* of a diagram if it has an accepting property trail in the diagram. The set of property models of  $\mathcal{G}$  is written  $\mathcal{L}_p(\mathcal{G})$ .

Given a property labeling  $\nu$ , a diagram  $\mathcal{G}$  defines a finite-state  $\omega$ -automaton  $\mathcal{A}_{\mathcal{G}}$  by interpreting the atomic assertions of  $\nu$  as propositions. Similarly, the property  $\varphi$  defines a finite-state  $\omega$ -automaton  $\mathcal{A}_{\varphi}$ . The models of both  $\mathcal{A}_{\mathcal{G}}$  and  $\mathcal{A}_{\varphi}$  are infinite sequences of states that assign truth values to these atomic assertions.

The verification conditions to prove Property Satisfaction are

- S1** for every node  $n \in N$ ,  $\mu(n) \rightarrow \nu(n)$ , which can be shown deductively.
- S2** the language inclusion  $\mathcal{L}(\mathcal{A}_{\mathcal{G}}) \subseteq \mathcal{L}(\mathcal{A}_{\varphi})$  holds, which can be shown by standard decidable  $\omega$ -automata techniques.

Condition **S1** proves  $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}_p(\mathcal{G})$ ; from **S2** the inclusion  $\mathcal{L}_p(\mathcal{G}) \subseteq \mathcal{L}(\varphi)$  follows, and by transitivity we have  $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\varphi)$ .

### Example 3

Returning to program SIMPLE of Figure 1, it is our goal to verify that each process  $i$  eventually sets  $a[i]$  to true. That is, we want to prove

$$\varphi[i] : \diamond \square a[i] \quad \text{for all } i \in [1..M] .$$

However, we first prove the weaker property

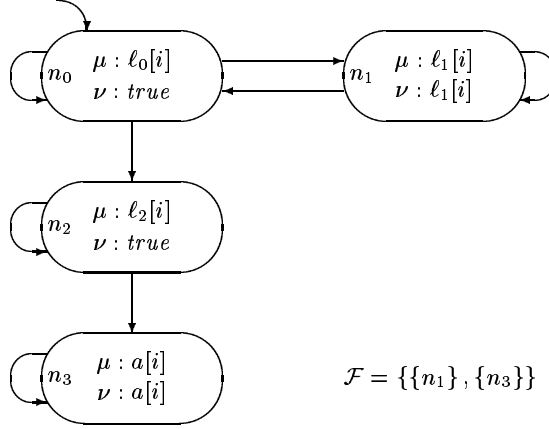
$$\psi[i] : \square \diamond \neg \ell_1[i] \rightarrow \diamond \square a[i] \quad \text{for all } i \in [1..M] ,$$

given in Example 2.

Figure 2 shows the verification diagram  $\mathcal{G}_1[i]$ , parameterized by  $i$ . In the diagram, initial nodes are indicated by a sourceless edge going to the node. The diagram  $\mathcal{G}_1[i]$  represents a proof of  $\psi[i]$ , for all  $i \in [1..M]$ . That is, for an arbitrary  $i \in [1..M]$ ,  $\mathcal{L}(\text{SIMPLE}) \subseteq \mathcal{L}(\mathcal{G}_1[i]) \subseteq \mathcal{L}(\psi[i])$ .

The acceptance condition,  $\mathcal{F} = \{\{n_1\}, \{n_3\}\}$  states that every trail of a computation must eventually end up in nodes  $n_1$  or  $n_3$ . To justify  $\mathcal{L}(\mathcal{G}_1[i]) \subseteq \mathcal{L}(\psi[i])$ , we have to show: **S1** the property labeling  $\nu$  is implied by the node labeling  $\mu$ , which is trivial in this case, and **S2** the inclusion  $\mathcal{L}(\mathcal{A}_{\mathcal{G}_1[i]}) \subseteq \mathcal{L}(\mathcal{A}_{\psi[i]})$  holds, which is obvious (note that  $\psi[i]$  can be rewritten into  $\diamond \square \ell_1[i] \vee \diamond \square a[i]$ , to make the connection between the acceptance condition and the property more obvious).

To justify  $\mathcal{L}(\text{SIMPLE}) \subseteq \mathcal{L}(\mathcal{G}_1[i])$ , we have to show Initiation, Consecution and Acceptance. Initiation and Consecution are easily shown. To show Acceptance, we have to show that the three nonaccepting SCSs are well-founded or have a fair exit transition. The SCS  $\{n_0, n_1\}$  is shown to be well-founded with the ranking function  $\delta : M + 1 - j[i]$  defined on both nodes, and the SCSs  $\{n_0\}$  and  $\{n_2\}$  have fair exit transitions  $\ell_0[i]$  and  $\ell_2[i]$  respectively. Therefore  $\mathcal{L}(\text{SIMPLE}) \subseteq \mathcal{L}(\psi[i])$  for  $i \in [1..M]$ .



**Fig. 2.** Verification diagram  $\mathcal{G}_1[i]$ , proving  $\psi[i] : \square \diamond \neg l_1[i] \rightarrow \diamond \square a[i]$

Note that we would not have been able to justify a nonaccepting  $\{n_1\}$ , since transition  $l_1[i]$  is not guaranteed to be enabled on  $n_1$ , and therefore is not a fair exit transition. This led us to include  $\{n_1\}$  in the acceptance condition, and thus prove the weaker property. ■

### 3.4 Previously Proven Properties

Diagram verification enables the use of previously proven properties in several ways. As in verification by verification rules, invariants of the program can be added to the antecedents of all verification conditions. However, previously proven temporal properties can be used as well.

Arbitrary temporal properties can be used to relax the Property Satisfaction condition as follows [BMS96]. Let  $\mathcal{S} \models \varphi_1, \dots, \mathcal{S} \models \varphi_n$ , and let  $\mathcal{G}$  be a diagram for  $\mathcal{S}$  and  $\varphi$ . Then, for Property Satisfaction, it suffices to show

$$\mathcal{L}(\varphi_1) \cap \dots \cap \mathcal{L}(\varphi_n) \cap \mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\varphi) ,$$

instead of  $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\varphi)$ . To perform this check, additional propositions, appearing in  $\varphi_1 \dots \varphi_n$ , may have to be added to the diagram.

Simple temporal properties can also be used to show that a diagram is  $\mathcal{S}$ -valid. We say that an SCS  $\mathcal{S} : \{n_1, \dots, n_k\}$  is *terminating* if

$$\square \diamond \neg(\mu(n_1) \vee \dots \vee \mu(n_k))$$

is  $\mathcal{S}$ -valid, that is every computation will always eventually leave the SCS. The Acceptance condition can now be relaxed to

- *Acceptance*: Every nonaccepting SCS  $\mathcal{S}$  ( $\mathcal{S} \notin \mathcal{F}$ ), has a fair exit transition, is well-founded, or is terminating.

## 4 Diagram Induction

Proofs of progress properties of concurrent systems usually require the explicit identification of the transitions that ensure progress, called *helpful transitions*. For nonparameterized systems the number of distinct helpful transitions is bounded; *ranking functions* are used if these helpful transitions have to be taken an unknown number of times to achieve the goal. Therefore all helpful transitions can be explicitly represented in the diagram.

The situation is different when the system is parameterized. In this case, achieving the goal may depend on an unbounded number of distinct helpful transitions, and thus they cannot be represented explicitly in the diagram. For example, in program SIMPLE, achieving  $\Box \Diamond \neg \ell_1[i]$  may require a number of distinct transitions proportional to  $M$ .

A possible solution in this case is to use *existential diagrams*, which assert the existence of a process index for which the transition guarantees progress. Existence must then be shown as part of the proof of the verification conditions.

### Example 4

In Example 3 we succeeded in proving

$$\psi[i] : \Box \Diamond \neg \ell_1[i] \rightarrow \Diamond \Box a[i] \quad \text{for all } i \in [1..M] .$$

If we are able to prove

$$\chi[i] : \Box \Diamond \neg \ell_1[i] \quad \text{for all } i \in [1..M] ,$$

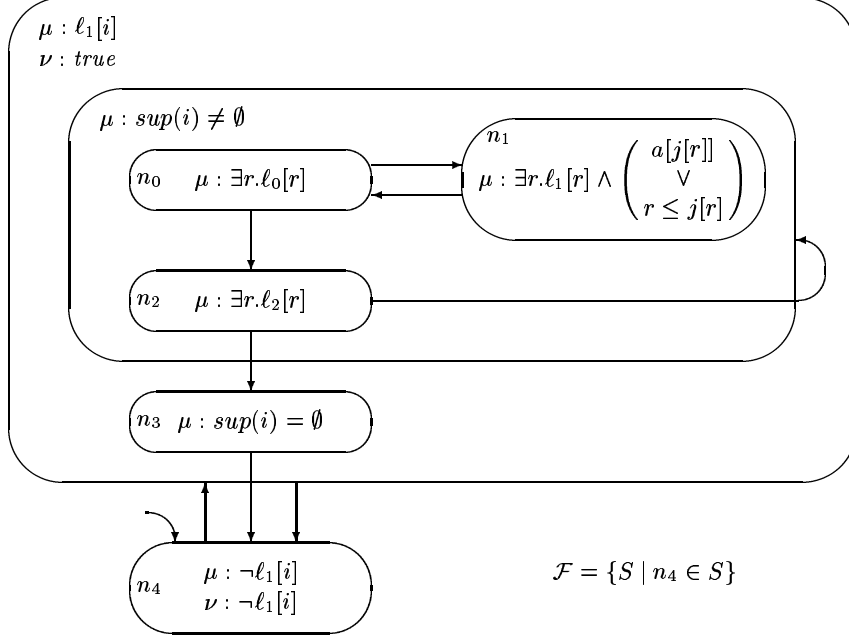
we can conclude that the desired property,  $\varphi[i] : \Diamond \Box a[i]$ , holds.

Figure 3 shows the existential diagram  $\mathcal{G}_2[i]$ , which could be used to prove  $\chi[i]$ . The diagram uses *encapsulation conventions*: a compound node labeled by an assertion  $p$  adds  $p$  as a conjunct to all of its subnodes, and an edge arriving at (or departing from) a compound node represents a set of edges that arrive at (or depart from) each of its subnodes.

In the diagram  $sup(i)$  stands for the set of process indices for which process  $i$  is waiting, that is, those processes that have priority over  $i$ ,

$$sup(i) \stackrel{\text{def}}{=} \{r \mid r < i \wedge \neg a[r]\} .$$

The diagram states that as long as  $sup(i) \neq \emptyset$ , there exists a process  $r$  at location  $\ell_0$ ,  $\ell_1$  or  $\ell_2$ , and, if at  $\ell_1$ , the process is enabled. Proving Initiation for this diagram is straightforward. However, proving Consecution is much harder than for the usual diagrams, due to the existential quantifiers in the verification conditions. For example, using Hoare triple notation, the consecution condition for  $n_2$  and transition  $\ell_2$  is



**Fig. 3.** Existential diagram  $\mathcal{G}_2[i]$ , proving  $\chi[i] : \square \diamond \neg \ell_1[i]$

$$\begin{array}{c}
 \{ \exists r. \ell_2[r] \wedge \ell_1[i] \wedge \text{sup}(i) \neq \emptyset \} \\
 \ell_2 \\
 \left\{ \begin{array}{l}
 \ell_1[i] \wedge \text{sup}(i) \neq \emptyset \wedge \exists r. \ell_0[r] \quad \vee \\
 \ell_1[i] \wedge \text{sup}(i) \neq \emptyset \wedge \exists r. \ell_1[r] \wedge (a[j[r]] \vee r \leq j[r]) \quad \vee \\
 \ell_1[i] \wedge \text{sup}(i) \neq \emptyset \wedge \exists r. \ell_2[r] \quad \vee \\
 \ell_1[i] \wedge \text{sup}(i) = \emptyset \quad \vee \\
 \neg \ell_1[i] \quad \vee
 \end{array} \right\}
 \end{array}$$

The proof of this verification condition requires the instantiation of  $r$  with the process index of the process that is enabled if  $\text{sup}(i) \neq \emptyset$ .

The verification conditions to justify Acceptance are slightly different from those presented in this paper, to ensure that identity of transitions is preserved for fairness. The full definition of existential diagrams is given in [MP96].  $\blacksquare$

We now describe our new approach, showing how mathematical induction on a process index can be used to simplify the diagrams and the corresponding verification conditions needed to prove a progress property  $\varphi[i]$ , over a system  $S$  consisting of  $M$  processes.

The standard mathematical induction principle states that to prove  $P[i]$  for all natural numbers, it suffices to prove  $P[i]$  for an arbitrary  $i$ , assuming  $\forall k < i. P[k]$  holds. This principle is directly applicable to the proof of temporal properties. However, the requirement of a fixed order on the process indices severely limits its applicability. Therefore we introduce a principle of mathematical induction for diagrams that allows a state-dependent order, that is, the truth value of  $k < i$  may change from state to state in a computation.

Diagram induction requires an *order function*  $\prec : \Sigma \mapsto 2^{[1..M] \times [1..M]}$  that maps each state  $s$  to a relation  $\prec(s)$ , such that for every  $s \in \Sigma$ , the relation  $\prec(s)$  is a partial order on  $[1..M]$  (that is,  $\prec(s)$  is transitive and irreflexive). The order function  $\prec$  is incorporated in the inductive hypothesis as follows.

Let  $\varphi[i]$  be the property to be proven for all  $i \in [1..M]$ , let  $\prec$  be an order function, and let  $\mathcal{A}_{\varphi[i]} : \langle N, N_0, E, \nu, \mathcal{F} \rangle$  be an  $\omega$ -automaton for  $\varphi[i]$ . Then the automaton for the inductive hypothesis for  $\varphi[i]$  and  $\prec$  is the  $\omega$ -automaton  $\mathcal{A}_{\varphi[i]}^{\prec}[k] : \langle N^i, N_0^i, E^i, \nu^i, \mathcal{F}^i \rangle$  obtained from  $\mathcal{A}_{\varphi[i]}$  as follows:

$$\begin{aligned} N^i &= N \cup \{n_e\} \\ N_0^i &= N_0 \cup \{n_e\} \\ E^i &= E \cup \{(n, n_e) \mid n \in N\} \cup \{(n_e, n) \mid n \in N\} \\ \nu^i(n) &= \nu(n)[k/i] \wedge k < i && \text{for } n \in N \\ \nu^i(n_e) &= \neg(k < i) \\ \mathcal{F}^i &= \mathcal{F} \cup \{S \mid n_e \in S\} \end{aligned}$$

Informally, this *inductive hypothesis automaton*  $\mathcal{A}_{\varphi[i]}^{\prec}[k]$  includes those sequences of states that satisfy  $\varphi[k]$  or that satisfy  $\neg(k < i)$  infinitely often.

### Example 5

Figure 4 shows the  $\omega$ -automaton and inductive hypothesis automaton for the property  $\varphi[i] : \diamond \square a[i]$  and the order function  $\prec$ . ■

**Lemma 1.** *The set of models of the inductive hypothesis of  $\varphi[i]$  for  $k$  with order function  $\prec$  includes all models of  $\varphi[k]$ , that is,*

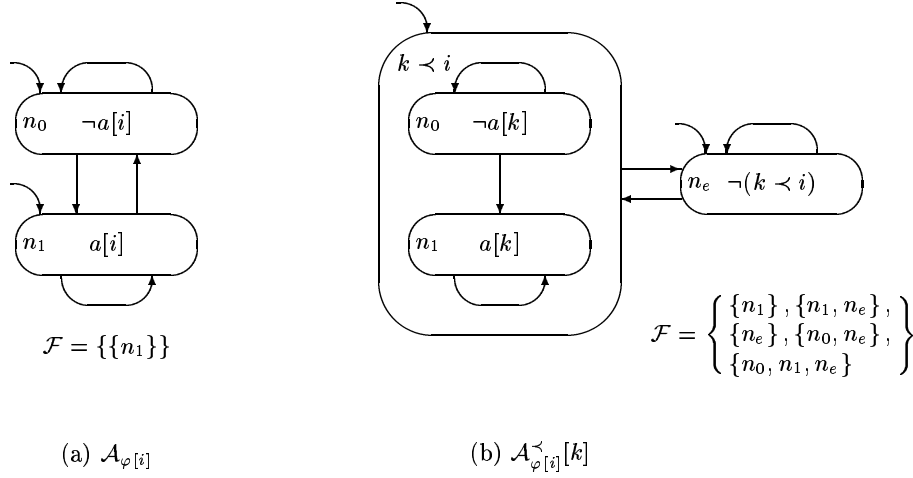
$$\mathcal{L}(\varphi[k]) \subseteq \mathcal{L}(\mathcal{A}_{\varphi[i]}^{\prec}[k])$$

**Lemma 2.** *For every order function  $\prec$ , every sequence of states  $\sigma \in \mathcal{L}(\varphi[k]_S)$  has a trail in  $\mathcal{A}_{\varphi[i]}^{\prec}[k]$ .*

With this definition of inductive hypothesis, we can now formulate the induction principle for diagrams:

### Diagram Induction Principle

Consider a parameterized system  $\mathcal{S}$ , consisting of  $M$  processes, and a property  $\varphi[i]$  to be proven for all  $i \in [1..M]$ . Assume there exists a diagram  $\mathcal{G}[i]$ , parameterized by  $i$ , and an order function  $\prec$  such that the following conditions hold for all  $i \in [1..M]$ :



**Fig. 4.**  $\omega$ -Automaton and inductive hypothesis automaton for  $\varphi[i] : \diamond \square a[i]$

- I1**  $\prec(s)$  is a partial order on  $[1..M]$  for each  $\mathcal{S}$ -accessible state  $s$ ;
- I2**  $\mathcal{S}$  satisfies the safety closure of  $\varphi[i]$ , that is,  $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\varphi[i]_S)$ ;
- I3**  $\mathcal{G}[i]$  is  $\mathcal{S}$ -valid, that is  $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G}[i])$ ;
- I4** there exists a set of indices  $K \subseteq [1..M]$  such that the product of  $\mathcal{G}[i]$  and the inductive hypothesis automata  $\mathcal{A}_{\varphi[i]}^{\prec}[k]$ , for  $k \in K$ , is included in  $\varphi[i]$ , that is,

$$\left( \mathcal{L}(\mathcal{G}[i]) \cap \bigcap_{k \in K} \mathcal{L}(\mathcal{A}_{\varphi[i]}^{\prec}[k]) \right) \subseteq \mathcal{L}(\varphi[i]) \quad \text{for some } K \subseteq [1..M] .$$

Then  $\mathcal{S} \models \varphi[i]$ , for all  $i \in [1..M]$ .

### Example 6

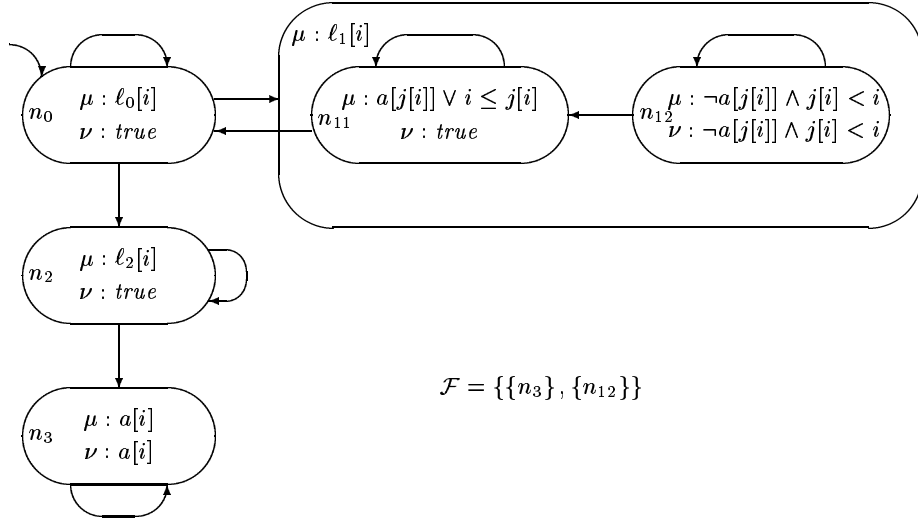
Returning to program SIMPLE, we refine diagram  $\mathcal{G}_1[i]$  shown in Figure 2 by splitting node  $n_1$  into two nodes:  $n_{11}$  where  $\ell_1[i]$  is guaranteed to be enabled and  $n_{12}$  where it is not enabled. The result is the verification diagram  $\mathcal{G}_3[i]$ , shown in Figure 5. The acceptance condition is  $\mathcal{F} = \{\{n_3\}, \{n_{12}\}\}$ . Initiation, Consecution and Acceptance are easily shown for this diagram, and thus  $\mathcal{L}(\text{SIMPLE}) \subseteq \mathcal{L}(\mathcal{G}_3[i])$ . For the property

$$\psi[i] : \square \diamond a[j[i]] \rightarrow \diamond \square a[i] \quad \text{for all } i \in [1..M] ,$$

Property Satisfaction is also easy to show, and thus  $\mathcal{L}(\mathcal{G}_3[i]) \subseteq \mathcal{L}(\psi[i])$ , and therefore  $\text{SIMPLE} \models \psi[i]$  for all  $i \in [1..M]$ .

However, we claim that by diagram induction diagram  $\mathcal{G}_3[i]$  also represents a proof of the desired property

$$\varphi[i] : \diamond \square a[i] \quad \text{for all } i \in [1..M] ,$$



**Fig. 5.** Verification diagram  $\mathcal{G}_3[i]$ , proving  $\varphi[i] : \square \diamond a[i]$  by diagram induction

using the order function  $\prec$  defined as the less-than relation ( $<$ ) on  $[1..M]$  at all states. Premise **I1** clearly holds. The safety closure of  $\varphi$  is  $\varphi_S : true$ , so premise **I2** holds trivially. The diagram was shown to be  $\mathcal{S}$ -valid earlier, thus satisfying premise **I3**. Finally, by taking  $K$  to be the singleton set  $\{j[i]\}$  the intersection of  $\mathcal{G}_3[i]$  of Figure 5 and  $\mathcal{A}_{\varphi[i]}^{\prec}[j[i]]$  of Figure 4(b) is included in the property  $\diamond \square a[i]$  of Figure 4(a), since the inductive hypothesis for  $j[i]$  eliminates the SCS  $\{n_{12}\}$ . Therefore, by the diagram induction principle, we can conclude that  $SIMPLE \models \diamond \square a[i]$  for all  $i \in [1..M]$ . ■

**Theorem 1 (Soundness).** *The Diagram Induction Principle is sound.*

**Proof**

Assume that the premises **I1** through **I4** hold, and suppose, for a contradiction, that  $\mathcal{S} \not\models \varphi[k_1]$  for some  $k_1 \in [1..M]$ . Then there exists a computation  $\sigma : s_0, s_1, \dots$  of  $\mathcal{S}$  such that  $\sigma \not\models \varphi[k_1]$ , that is,  $\sigma \notin \mathcal{L}(\varphi[k_1])$ . By premise **I3** we have  $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G}[k_1])$ , and therefore  $\sigma \in \mathcal{L}(\mathcal{G}[k_1])$ . But then, by premise **I4**, there must exist some  $k_2$  such that  $\sigma \notin \mathcal{L}(\mathcal{A}_{\varphi[k_1]}^{\prec}[k_2])$ . By premise **I2**,  $\sigma \in \mathcal{L}(\varphi[k_2]_S)$ , and thus by Lemma 2 the sequence  $\sigma$  has a trail in  $\mathcal{A}_{\varphi[k_1]}^{\prec}[k_2]$ . From the fact that the trail is not accepting we can conclude that the trail eventually has to end up outside the added node  $n_e$ , since all sets that include this node are accepting. All nodes in  $\mathcal{A}_{\varphi[k_1]}^{\prec}[k_2]$  outside  $n_e$  are labeled by  $k_2 \prec k_1$  and therefore we have  $\sigma \models \diamond \square (k_2 \prec k_1)$ . In addition, by Lemma 1, we know that  $\sigma \not\models \varphi[k_2]$ .

By the same reasoning we can conclude that there exists some  $k_3$  such that  $\sigma \notin \mathcal{A}_{\varphi[k_2]}^{\prec}[k_3]$ , and that  $\sigma \models \diamond \square (k_3 \prec k_2)$ . Repeating this argument  $M$  times

we can conclude that

$$\sigma \models \diamond \square (k_{M+1} \prec k_M) \wedge \dots \wedge \diamond \square (k_2 \prec k_1) ,$$

and therefore

$$\sigma \models \diamond \square ((k_{M+1} \prec k_M) \wedge \dots \wedge (k_2 \prec k_1)) ,$$

and thus there exists a particular state  $s_z$  in  $\sigma$  such that

$$s_z \models k_{M+1} \prec k_M \prec \dots \prec k_1$$

However, some process index  $k$  must appear twice in this sequence, violating premise **I1**, that  $\prec(s_z)$  is a partial order, a contradiction.  $\square$

## 5 Example: Accessibility for BAKERY-M

In this section we give an outline of the proof of accessibility for the parameterized system BAKERY-M, shown in Figure 6. This program, taken from [Pnu96], is based on Lamport's bakery algorithm [Lam74,Lam77] for mutual exclusion.

$\parallel_{i=1}^M P[i] ::$	<pre> <b>in</b>   <math>M</math>       : <b>integer where</b> <math>M &gt; 0</math> <b>local</b> <math>choosing</math> : <b>array</b> <math>[1..M]</math> <b>of boolean where</b> <math>choosing = false</math>        <math>number</math>   : <b>array</b> <math>[1..M]</math> <b>of integer where</b> <math>number = 0</math>         [          <b>local</b> <math>j, k, t</math> : <b>integer where</b> <math>j = 0, k = 0, t = 0</math>          <math>\ell_0</math>: <b>loop forever do</b>            [              <math>\ell_1</math>: <b>noncritical</b>              <math>\ell_2</math>: <math>choosing[i] := true</math>              <math>\ell_3</math>: <math>t := 0</math>              <math>\ell_4</math>: <b>for</b> <math>k = 1</math> <b>to</b> <math>M</math> <b>do</b>                <math>\ell_5</math>: <math>t := \max(t, number[k])</math>              <math>\ell_6</math>: <math>number[i] := t + 1</math>              <math>\ell_7</math>: <math>choosing[i] := false</math>              <math>\ell_8</math>: <b>for</b> <math>j = 1</math> <b>to</b> <math>M</math> <b>do</b>                [                  <math>\ell_9</math>: <b>await</b> <math>\neg choosing[j]</math>                  <math>\ell_{10}</math>: <b>await</b> <math>\left( \begin{array}{l} j = i \\ \vee number[j] = 0 \\ \vee (number[i], i) \prec (number[j], j) \end{array} \right)</math>                ]              <math>\ell_{11}</math>: <b>critical</b>              <math>\ell_{12}</math>: <math>number[i] := 0</math>            ]          ]        ] </pre>
-----------------------------	---

**Fig. 6.** Program BAKERY-M: molecular version

In the program, in statements  $\ell_2$  through  $\ell_7$ , process  $i$  determines the maximum ticket number currently held by any other process and assigns it, incremented by 1, to its own ticket number. In the coarser-grained atomic version of the program these statements are replaced by

$$\ell_1^a : \mathit{number}[i] := 1 + \mathit{max}(\mathit{number}) .$$

In statements  $\ell_8$  through  $\ell_{10}$  process  $i$  can proceed only if every other process has a ticket number 0 (meaning it is not interested in entering the critical section), or its ticket number is higher than that of process  $i$ . In the atomic version these statements become

$$\ell_2^a : \mathbf{await} \ \forall j : [1..M].(i \neq j \rightarrow \mathit{number}[j] = 0 \vee \mathit{number}[i] \leq \mathit{number}[j]) .$$

In [Lam77,Pnu96] it is shown that BAKERY-M guarantees mutual exclusion and accessibility, where in [Pnu96] accessibility is proven using existential diagrams. Here we present an alternative proof of accessibility that uses diagram induction.

The proof of accessibility, specified by

$$\mathit{acc}[i] : \square(\ell_2[i] \rightarrow \diamond \ell_{11}[i]) \quad \text{for all } i \in [1..M] ,$$

is represented by four verification diagrams. The  $\mathcal{S}$ -validity of the diagrams relies on the following invariants, taken from [Pnu96],

$$\begin{aligned} I_0 &: \mathit{choosing}[i] \leftrightarrow \ell_{3..7}[i] \\ I_1 &: \ell_{7..12}[i] \leq \mathit{number}[i] \\ I_2 &: (\ell_4 \rightarrow 1 \leq k \leq M + 1) \wedge (\ell_5 \rightarrow 1 \leq k \leq M) \\ I_3 &: \ell_6 \rightarrow k = M + 1 \\ I_4 &: (\ell_8 \rightarrow 1 \leq j \leq M + 1) \wedge (\ell_{9,10} \rightarrow 1 \leq j \leq M) \\ I_5 &: \ell_{11} \rightarrow j = M + 1 \\ I_6 &: \ell_{10}[i] \wedge \mathit{choosing}[j[i]] \rightarrow \mathit{superior}[i, j[i]] \end{aligned}$$

where

$$\mathit{superior}[i, r] : \left( \begin{array}{l} \ell_{0..3}[r] \\ \vee \ell_{4..6}[r] \wedge (k[r] \leq i \vee \mathit{number}[i] \leq t[r]) \\ \vee \ell_{7..12}[r] \wedge (\mathit{number}[i], i) \prec (\mathit{number}[r], r) \end{array} \right)$$

The first diagram,  $\mathcal{G}_4[i]$ , shown in Figure 7, proves that accessibility holds provided the program will always leave locations  $\ell_9$  and  $\ell_{10}$ :

$$\varphi_4[i] : (\square \diamond \neg \ell_9[i] \wedge \square \diamond \neg \ell_{10}[i]) \rightarrow \square(\ell_2[i] \rightarrow \diamond \ell_{11}[i])$$

The diagram is a straightforward representation of the path that leads from location  $\ell_2$  to  $\ell_{11}$ . Initiation and Consecution clearly hold for this diagram. To

justify the acceptance condition,  $\mathcal{F} = \{\{n_7\}, \{n_8\}, \{n_{10}\}\} \cup \{S \mid n_9 \in S\}$ , we have to show that all nonaccepting SCSs have a fair exit or are well-founded. It is easy to see that all nonaccepting single-node SCSs have a fair exit transition. The two remaining SCSs,  $\{n_2, n_3\}$ , and  $\{n_6, n_7, n_8\}$  are shown to be well-founded using the ranking functions  $\delta(n_2) = \delta(n_3) = M + 1 - k[i]$ , and  $\delta(n_6) = \delta(n_7) = \delta(n_8) = M + 1 - j[i]$ , respectively. The well-foundedness of  $M + 1 - k[i]$  and  $M + 1 - j[i]$  relies on the invariants  $I_2$  and  $I_4$  respectively.

It remains to show that the system cannot forever stay in nodes  $n_7$  and  $n_8$ , that is,

$$\begin{aligned} \psi_1[i] &: \square \diamond \neg \ell_9[i] && \text{for all } i \in [1..M] \\ \psi_2[i] &: \square \diamond \neg \ell_{10}[i] && \text{for all } i \in [1..M] \end{aligned}$$

Two diagrams, not included in this paper, prove that for all  $i \in [1..M]$

$$\varphi_5[i] : \square \diamond \neg \text{choosing}[j[i]] \rightarrow \square \diamond \neg \ell_9[i]$$

and

$$\varphi_6[i] : \square \diamond \neg \text{choosing}[i]$$

respectively, from which  $\psi_1[i]$  can be concluded for all  $i \in [1..M]$ .

The diagram  $\mathcal{G}_7[i]$ , shown in Figure 8, represents a proof of  $\psi_2[i]$  using diagram induction. Informally, the nodes  $n_0, \dots, n_5$  represent the situation that process  $j[i]$  has priority over process  $i$  to enter its critical section. In node  $n_6$ , process  $i$  has priority over  $j[i]$  and on this node transition  $\ell_{10}[i]$  is guaranteed to be enabled, leading to the goal node  $n_7$ .

Initiation is easily shown for this diagram. Consecution requires several of the invariants. In particular the verification condition

$$\{ \mu(n_6) \} \ell_6 \{ \mu(n_6) \vee \mu(n_7) \}$$

represents a crucial part of the proof, namely that once process  $j[i]$  has left its critical section, it will not return with a lower ticket number than process  $i$  while process  $i$  is at  $\ell_{10}$ .

To justify the acceptance condition  $\mathcal{F} = \{\{n_3\}\} \cup \{S \mid n_7 \in S\}$ , all single-node, nonaccepting SCS's, except  $\{n_2\}$  are shown to have fair exit transitions;  $\{n_2\}$  is shown to be terminating by  $\psi_1[j[i]]$ ; and the SCS  $\{n_1, n_2, n_3\}$  is established to be well-founded, with ranking functions  $\delta(n_1) = \delta(n_2) = \delta(n_3) = M + 1 - j[j[i]]$ , whose well-foundedness relies on the invariant  $I_4$ .

Without using induction the diagram  $\mathcal{G}_7$  represents a proof of

$$\varphi_7[i] : \square \diamond \neg \ell_{10}[j[i]] \rightarrow \square \diamond \neg \ell_{10}[i] .$$

However, if we apply the diagram induction principle with order function  $\prec$ , defined by

$$i \prec k \quad \text{iff} \quad (\text{number}[i], i) < (\text{number}[k], k)$$

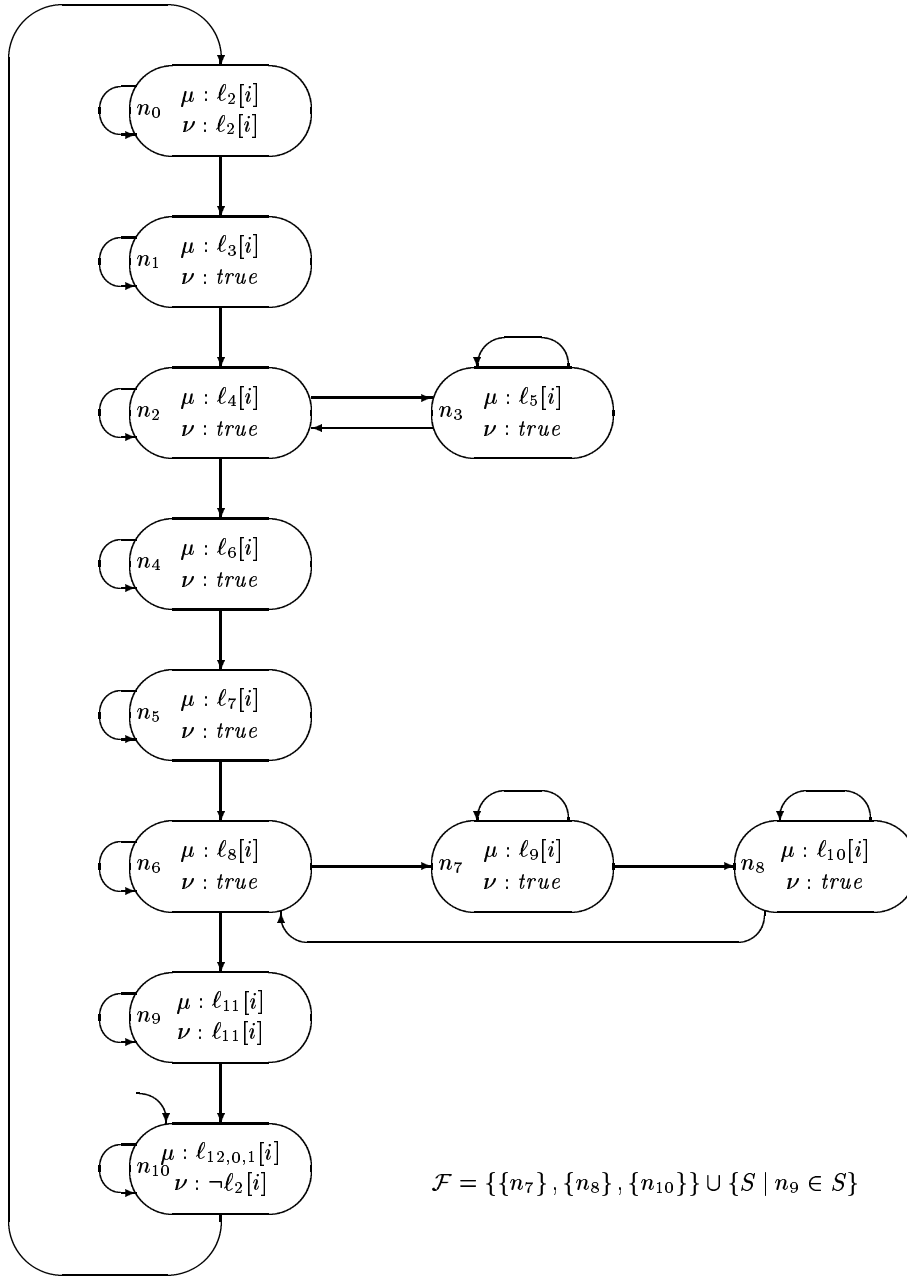
at each state, and take  $K$  to be the singleton set  $\{j[i]\}$ , the offending SCS  $\{n_3\}$  is eliminated and the diagram represents a proof of  $\psi_2[i]$  for all  $i \in [1..M]$ , as desired. This completes the proof of accessibility for BAKERY-M.

## Acknowledgements

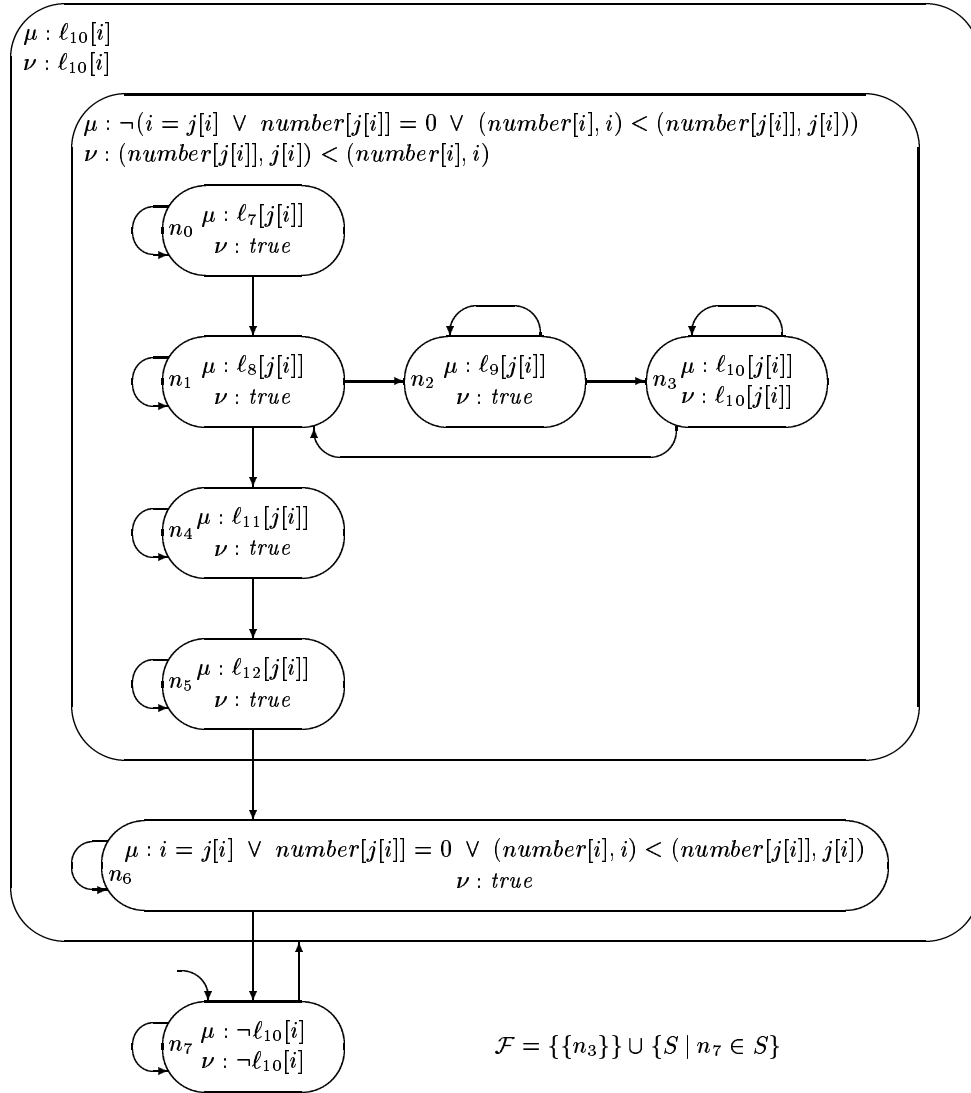
We thank Anca Browne, Michael Colón and Tomás Uribe for their comments and suggestions.

## References

- [AL90] M. Abadi and L. Lamport. Composing specifications. In *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, vol. 430 of *LNCS*, pages 1–41. Springer-Verlag, 1990.
- [BLM97] N.S. Bjørner, U. Lerner, and Z. Manna. Deductive verification of parameterized fault-tolerant systems: A case study. In *Intl. Conf. on Temporal Logic*. Kluwer, 1997. To appear.
- [BMS95] A. Browne, Z. Manna, and H.B. Sipma. Generalized temporal verification diagrams. In *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, vol. 1026 of *LNCS*, pages 484–498. Springer-Verlag, 1995.
- [BMS96] A. Browne, Z. Manna, and H.B. Sipma. Hierarchical verification using verification diagrams. In *2<sup>nd</sup> Asian Computing Science Conf.*, vol. 1179 of *LNCS*, pages 276–286. Springer-Verlag, December 1996.
- [Lam74] L. Lamport. A new solution of Dijkstra’s concurrent programming problem. *Communications of the ACM*, 17(8):435–455, 1974.
- [Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Engin.*, 3:125–143, 1977.
- [MBSU98] Z. Manna, A. Browne, H.B. Sipma, and T.E. Uribe. Visual abstractions for temporal verification. In A. Haeberer, editor, *AMAST’98*, vol. 1548 of *LNCS*, pages 28–41. Springer-Verlag, December 1998.
- [MP94] Z. Manna and A. Pnueli. Temporal verification diagrams. In M. Hagiya and J.C. Mitchell, editors, *Proc. International Symposium on Theoretical Aspects of Computer Software*, vol. 789 of *LNCS*, pages 726–765. Springer-Verlag, 1994.
- [MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
- [MP96] Z. Manna and A. Pnueli. Temporal verification of reactive systems: Progress. Draft Manuscript, 1996.
- [Pnu96] A. Pnueli. Lecture notes: the Bakery algorithm. Draft Manuscript, Weizmann Institute of Science, Israel, May 1996.
- [Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, pages 133–191. Elsevier Science Publishers (North-Holland), 1990.



**Fig. 7.** Diagram  $\mathcal{G}_4[i]$ , proving  $(\Box \Diamond \neg \ell_9[i] \wedge \Box \Diamond \neg \ell_{10}[i]) \rightarrow \Box(\ell_2[i] \rightarrow \Diamond \ell_{11}[i])$



**Fig. 8.** Verification diagram  $\mathcal{G}_7[i]$ , proving  $\varphi[i] : \square \diamond \neg \ell_{10}[i]$ , by diagram induction