Automatic Verification of Parameterized Networks of Processes by Abstraction

David Lesens and Hassen Saïdi

VERIMAG*
Centre Equation, 2, rue de Vignate, 38610 Gières, France
Email: {David.Lesens,Hassan.Saidi}@imag.fr
Phone: (+33) 4.76.63.48.44, Fax: (+33) 4.76.63.48.50

June 11, 1997

Abstract

In this paper we are interested in the verification of safety properties of parameterized networks. A network is defined as a parallel composition of an arbitrary but finite number of identical sequential processes, where we consider parallel composition by interleaving and synchronization by shared variables. Using abstraction techniques, a process, called an *abstract network*, encoding the behavior of the entire network is constructed. The property is then checked on this process. Our verification method has the following advantages: the construction of the abstract network is fully automatic; the obtained process is generally a simple process on which the property can be easily verified. Of course, if the property cannot be verified on the abstract network, another more precise abstraction has to be computed. The construction requires to discharge a set of first order verification conditions (VCs). The PVS theorem prover is used to discharge the generated VCs. This allows us to consider processes with arbitrary data types. The effectiveness of our verification method is illustrated on two examples including a parameterized version of the Fischer's protocol.

keywords: *computer-aided verification, parameterized networks, abstraction techniques, theorem proving.*

1 Introduction

After the first success of automatic verification of finite state systems [QS82, CES86], there has been much interest in verification of infinite state systems. Parameterized networks of processes are an interesting class of infinite state systems where both the control part and the data part can be infinite. In the case of networks of boolean processes, many verification techniques have been proposed (for example [LHR97]). These methods can be combined with abstraction techniques to verify networks of infinite state processes as suggested in [LS97].

In this paper we investigate the problem of the verification of safety properties of parameterized networks of infinite state processes, where we consider parallel composition by interleaving and synchronization by shared variables. We propose an abstraction technique allowing us to encode the behavior of the parallel composition of an arbitrary number of identical sequential processes. In [GS97] an algorithm for the automatic construction of an abstract state graph is presented. It allows to construct this graph for a fixed number of parallel components by discharging a set of first order formulas called verification conditions. Each abstract state is a valuation of a set of predicates $\varphi_1, ..., \varphi_\ell$ defining the abstract state space and a valuation of the concrete global control configuration. In the case of parameterized networks of processes, it is not possible to use the same abstraction algorithm. The obvious reason is

*Verimag is a joint laboratory of CNRS, Institut National Polytechnique de Grenoble, Université J. Fourier and Verilog SA associated with IMAG. http://www.imag.fr/VERIMAG/

that the control configuration is a tuple of arbitrary size. Thus, we propose to construct an abstraction of a network by applying successively two different abstractions: first, we construct an abstraction for a single process, and finally we construct an *abstract network* using the obtained abstract process. The constructed abstract network is a process with a finite set of transitions and a finite set of global control configurations. It has as variables the global variables of the concrete network, and a set of integers variables. These variables are added to count the number of processes of the concrete network verifying some predicates. In this abstract network only linear operations are performed on these integer variables. Thus if all the global variables of the concrete network are linear variables, we obtain a linear system on which the property we want to verify can be easily checked using classical verification methods such as deductive methods as in [Saï97], or using combination of abstraction techniques and deductive methods as in [GS97], or using symbolic model checking using polyhedra as in [HPR94]. The advantage of our method is that the abstract network can be constructed fully automatically.

This paper is organized as follows: we first give some preliminary definition to define our computational model that is transition systems. We describe the abstraction techniques used to abstract one process, and then our abstraction algorithm to abstract a network. We finally illustrate the method on two examples.

2 Preliminary definitions

As computational model for processes we use transition systems, labeled by guarded commands.

Definition 2.1 (Transition system)
A transition systems S is a tuple $< \mathcal{V},\ \mathcal{T} = \{\tau_1, \cdots, \tau_n\},\ L,\ Init >$, where $\mathcal{V}$ is a set of variables, $\mathcal{T}$ a set of transitions or actions, L a set of control locations and $Init$ the initial location.

Each transition τ_i is a guarded command

$$l_i : \quad guard_i \ \longrightarrow\ v_1 := e_1, \cdots, v_k := e_k \ \text{ goto } l_{i'}$$

where $\{v_1, \cdots, v_k\} \subseteq \mathcal{V}$ and $\{l_i, l_{i'}\} \subseteq L$. $guard_i$ is the guard of the transition τ_i, defining the set of states from which τ_i is enabled. Each variable v_i is assigned with an expression e_i of a compatible type. Locations l_i and $l_{i'}$ are respectively the source and target location of the transition τ_i.

Definition 2.2 (Predicate transformer)
$\widetilde{pre}\{\tau_i\}(\varphi)$ is the weakest precondition of the predicate φ with respect to the transition τ_i and is defined as the set of states from which it is not possible to reach via a transition τ_i a state satisfying $\neg\varphi$. $\widetilde{pre}\{\tau_i\}(\varphi)$ is equivalent to the predicate: $guard_i \Rightarrow \varphi[v_i/e_i]$. We also define the set $pre\{\tau_i\}(\varphi)$ as the set of predecessor of a set satisfying the predicate φ.

Definition 2.3 (Network)
A network is the parallel composition $\|_{i=1}^{N} S_i$ of N processes S_i. The set of global variable is $\mathcal{G}$. Each process S_i is a transition system $< \mathcal{V}_i,\ \mathcal{T}_i = \{\tau_1, \cdots, \tau_n\},\ L_i,\ Init_i >$ where the set of variable $\mathcal{V}_i$ is the union of the set $\mathcal{L}_i$ of local variables of S_i and the global variables $\mathcal{G}$.

Each process is allowed to read and to write its own local variables. The global variables of $\mathcal{G}$ are shared, and therefore can be read and written by any process S_i.

3 Abstraction of one process

In [GS97] we define an abstract space state using a set of predicates $\varphi_1, ..., \varphi_\ell$ on process variables. Each abstract state is a valuation[1] ψ of these predicates and the valuation of the concrete control variables. The process construction starts by defining the initial abstract state. This is done by checking the possible valuations of each predicate φ_i in the concrete initial state. The successors of an abstract state are computed as follows:

[1]A valuation ψ is a complete monomial defined on the predicates $\varphi_1, ..., \varphi_\ell$.

1. Select all the possible transitions τ_i from the abstract state.
2. For each such transition compute the postconditions of each predicate φ_i, that is whether φ_i or $\neg\varphi_i$ is a postcondition. The successor ψ' of a valuation ψ is then computed as follow:

$$\psi' = \begin{cases} false & \text{if } \psi \Rightarrow \neg guard_i \\ \forall i.\varphi_i = \left\{ \begin{array}{ll} true & \text{if } \psi \Rightarrow \widetilde{pre}\{\tau_i\}(\varphi_i) \\ false & \text{if } \psi \Rightarrow \widetilde{pre}\{\tau_i\}(\neg\varphi_i) \\ \perp & \text{otherwise} \end{array} \right. \Bigg\} & \text{otherwise} \end{cases}$$

In the case where $\perp$ is a postcondition of a predicate φ_i, two successors are created corresponding to the case where φ_i or $\neg\varphi_i$ is a postcondition. The successor $false$ is interpreted as the empty set, that is no successor exists for the considered abstract state via the transition τ_i. Thus, the abstract process $\tilde{S}_i$ of a process S_i is a transition system $< \mathcal{A}_\mathcal{V}, \mathcal{A}_\mathcal{T} = \{a\tau_1, \cdots, a\tau_m\}, \mathcal{A}_L, \mathcal{A}_{Init} >$, where $\mathcal{A}_\mathcal{V}$ is the set of abstract variables, $\mathcal{A}_\mathcal{T}$ a set of abstract transitions, $\mathcal{A}_L$ the set of new control locations and $\mathcal{A}_{Init}$ the abstract initial state. The set of transitions $\mathcal{A}_\mathcal{T}$ is obtained from the set $\mathcal{T}$ where the guards and the assignments involving the variables of the concrete process are replaced by expressions involving the predicates $\varphi_1, ..., \varphi_\ell$.

Choice of predicates $\varphi_1, ..., \varphi_\ell$: In [GS97] it is proposed to choose as predicates $\varphi_1, ..., \varphi_\ell$ the literals appearing in the guards of the process transitions. In general, we can chose any predicate on the process variables. However if the property is not verified on the obtained process, we have to construct a more precise abstraction by providing a relevant set of predicates.

4 Abstraction of a network

Let us first partition the set $\mathcal{G}$ of global variables into 2 sets $\mathcal{G}_1$ and $\mathcal{G}_2$:

- $\mathcal{G}_1$ is the set of global variables depending on some local variables. These variables will be abstracted in the construction of the abstract network.
- $\mathcal{G}_2 = \mathcal{G} - \mathcal{G}_1$, that is the set of global variables which do not depend on local variables.

Let $\mathcal{P}$ be the set of predicates used in the process S_i. Let us partition $\mathcal{P}$ on 3 sets of predicates Φ_1, Φ_2 and Φ_3, depending on the previous partition of $\mathcal{G}$.

- Φ_1 is the set of predicates depending *only* on local variables. They will be used to define the abstract process $\tilde{S}_i$ (and will be ignored in the abstract network)
- Φ_3 is the set of all predicates depending only on global variables of $\mathcal{G}_2$. There are the only predicates which will appear in the abstract network.
- Φ_2 is the set of all predicates which are not in Φ_1 or Φ_3. These predicates depend either on global variables of $\mathcal{G}_1$, or to both local variables and global variables of $\mathcal{G}_2$. They will be used to define the abstract network. Indeed, since they depend on local variables (which are abstracted), their values will be unknown in the abstract network.

Thus, we define $\tilde{S}_i$ as the abstraction of S_i using the predicates of Φ_1.

We propose then to construct an abstraction of the network using $\tilde{S}_i$ and Φ_2. The idea is to construct a process encoding the behavior of the network, where only global variables of $\mathcal{G}_2$ appear: we propose to associate to each abstract state l_i, a set of new variables (X_i^j), which counts the number of processes which are in this abstract state, and which satisfy a predicate φ_j. φ_j is a valuation of the predicates of Φ_2.

Definition 4.1 (abstract network)
An abstract network $\mathcal{AG}$ is a transition system $< \mathcal{AG}_\mathcal{V},\ \mathcal{AG}_\mathcal{T} = \{a\tau_1, \cdots, a\tau_k\},\ \{1\},\ \mathcal{AG}_{Init} >$.

$\mathcal{AG}_\mathcal{V}$ is the set of new system variables: it is defined as the union of the global variables of $\mathcal{G}_2$ (which are not abstracted) and the new variables (X_i^j). This transition system has only one location (numbered 1): this is due to the fact that the control configuration of the network are encoded by the set of variables (X_i^j). $\mathcal{A}_\mathcal{T}$ is the set of abstract transitions: this set construction is quite technical, only the main ideas of the abstraction algorithm is given here. For each transition τ_i

$$l_i : \quad guard_i \quad \longrightarrow \quad v_1 := e_1, \cdots, v_k := e_k \quad \text{goto } l_{i'}$$

where l_i and $l_{i'}$ are abstract states of $\tilde{S}_i$, we define a transition in the abstract network, for each value of predicates associated with l_i and $l_{i'}$. For each tuple $(\varphi_j, \varphi_{j'})$ in $\Phi_2 \times \Phi_2$, the transition is enabled only if

- some abstract processes are in the abstract state l_i, where the predicate φ_j holds, that is $X_i^j \geq 1$.
- the guard $guard_i$ is satisfied. Of course, only predicates which do not contain abstract variables must appear, that is only $guard_i \cap \Phi_3$.
- the predicates (of Φ_1) associated with the target abstract state $l_{i'}$ are compatible with the current values of non abstract global variables, that is, $\exists \mathcal{L}_i, \mathcal{G}_1.\, pre\{\tau_i\}(\varphi_{j'} \wedge \varphi(l_{i'}))$ (where $\varphi(l_{i'})$ denotes the valuation of the predicates of Φ_1 in the abstract state $l_{i'}$).

The new transition of the abstract network is then guarded by

$$guard' \quad := \quad (X_i^j \geq 1) \wedge (guard_i \cap \Phi_3) \wedge (\exists \mathcal{L}_i, \mathcal{G}_1.\, pre\{\tau_i\}(\varphi_{j'} \wedge \varphi(l_{i'})))$$

Of course, only some of these guards are satisfaisable.

Each assignment $v_k := e_k$ is either abstracted if $(v_k = e_k) \in \Phi_2$, or is maintained (in this case, only non abstract global variables appear in this assignment, that is $(v_k = e_k) \in \Phi_3$).

The new state is defined by updating the (X_i^j) in the following way:

- first, we state $X_i^j := X_i^j - 1$ to express the fact that a process leaves the ith abstract state where φ_j holds.
- second, we have to express the assignment of global variables in considering the change in predicates involving these global variables. In this way, all the processes of the network can change synchronously their abstract state by changing some of their predicates. Of course, concrete states are not modified.

  Thus, we consider each predicates $\varphi_k \in \Phi_2$. This predicate is associated with the two predicates $\varphi_{k'} = \neg\varphi_k$ (the negation of φ_k) and $\varphi_{k''} = \widetilde{\varphi_k}$ (the φ_k value is unknown).

  - If $\varphi_{j'} \wedge \varphi_k \wedge \ldots$ and $\varphi_{j'} \wedge \neg\varphi_k \wedge \ldots$ are both satisfaisable, the value of predicate φ_k will be unknown in the new state. Thus, all states verifying this predicate or this negation have to move. We state
  $$\forall m,\ X_m^{k''} := X_m^k + X_m^{k'} + X_m^{k''};\ X_m^k := 0;\ X_m^{k'} := 0$$
  i.e., all processes change synchronously one of their predicate.
  - If only $\varphi_{j'} \wedge \varphi_k \wedge \ldots$ is satisfaisable, we state
  $$\forall m,\ X_m^k := X_m^k + X_m^{k'} + X_m^{k''};\ X_m^{k'} := 0;\ X_m^{k''} := 0$$
  - If only $\varphi_{j'} \wedge \neg\varphi_k \wedge \ldots$ is satisfaisable, we state
  $$\forall m,\ X_m^{k'} := X_m^k + X_m^{k'} + X_m^{k''};\ X_m^k := 0;\ X_m^{k''} := 0$$
- finally, we state $X_{i'}^{j'} := X_{i'}^{j'} + 1$ to express the fact that a process moves to the i'th abstract state, which satisfies the predicate $\varphi_{j'}$.

5 Examples

This section presents two examples of mutual exclusion. Transition systems describing a process are shown respectively fig. 1 and 2. The tool described in [Saï97] computes an abstraction of these two processes by spliting each state depending on the predicates which appear in guards or assignments.

5.1 The mux_ast system

(see http://www.imag.fr/VERIMAG/PEOPLE/David.Lesens/NA/EXAMPLES/mux.html).

Consider the following example taken from [MP95], and implementing a mutual exclusion algorithm for n processes. Each process S_i has a local variable t_i and can access to the shared variable y (see fig. 1).

The abstract process $\tilde{S}_i$ is obtained using the predicates appearing in the guards and depending on the variable t_i, that is the predicates: $\Phi_1 = \{(t_i < 0), (t_i = -1), (t_i = 0)\}$. The obtained abstract process has 11 states (see fig. 3). Since y does not depend on local variables, the set Φ_2 is empty and the global variable y is not abstracted (that is $\mathcal{G}_1 = \emptyset$ and $\mathcal{G}_2 = \mathcal{G} = \{y\}$). Thus the abstract network can be described by 12 integer variables (11 states of the abstract process and y). Equations describing the abstract network can be found on the previous URL. An invariant can easily be found to prove mutual exclusion.

5.2 The Fischer's protocol

(see http://www.imag.fr/VERIMAG/PEOPLE/David.Lesens/NA/EXAMPLES/fischer.html).

Consider n processes like the one shown in fig. 2, each with an associated timer c_i. All of them have access to the shared integer variable $lock$. The time passing is modeled by an integer variable t which can be incremented. The ith process may attempt to enter its critical section only if $lock = 0$. Before two units of time, it has to lock it, by setting the variable $lock$ to i. Then, the process waits at least two other units of time, to allow any other processes that may have passed the check $lock = 0$ to perform their assignment $lock := i$. It can then enter its critical section if and only if $lock = i$. Once the process exits the critical section, it unlocks the $lock$.

Since $\Phi_1 = \emptyset$, S_i is equal to its abstraction. Since the shared variable $lock$ can be assign to i, it must be abstracted. The following predicates on $lock$ and t are used to construct the abstract network: $\Phi_2 = \{\varphi_1 = (lock = 0), \varphi_2 = (lock = i), \varphi_3 = (t - c_i = 0), \varphi_4 = (t - c_i = 1), \varphi_5 = (t - c_i \geq 2)\}$, $\Phi_3 = \emptyset$. We obtain 9 valuations: $\varphi_1.\overline{\varphi_2}.\varphi_3.\overline{\varphi_4}.\overline{\varphi_5}$, $\varphi_1.\overline{\varphi_2}.\overline{\varphi_3}.\varphi_4.\overline{\varphi_5}$, $\varphi_1.\overline{\varphi_2}.\overline{\varphi_3}.\overline{\varphi_4}.\varphi_5$, $\overline{\varphi_1}.\varphi_2.\varphi_3.\overline{\varphi_4}.\overline{\varphi_5}$, $\overline{\varphi_1}.\varphi_2.\overline{\varphi_3}.\varphi_4.\overline{\varphi_5}$, $\overline{\varphi_1}.\varphi_2.\overline{\varphi_3}.\overline{\varphi_4}.\varphi_5$, $\overline{\varphi_1}.\overline{\varphi_2}.\varphi_3.\overline{\varphi_4}.\overline{\varphi_5}$, $\overline{\varphi_1}.\overline{\varphi_2}.\overline{\varphi_3}.\varphi_4.\overline{\varphi_5}$, $\overline{\varphi_1}.\overline{\varphi_2}.\overline{\varphi_3}.\overline{\varphi_4}.\varphi_5$. Thus, our abstract network will be described by $4 * 9 = 36$ integer variables. Equations describing the abstract network can be found on the previous URL. An invariant can easily be found to prove mutual exclusion.

6 Conclusion

In this paper we have presented an algorithm for the automatic construction of abstraction of parameterized networks. The method is based on the proof of first order verification conditions. We have applied successfully our method to verify two example including a parameterized version of the Fischer's protocol.

References

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2), 1986.

[GS97] S. Graf and H. Saïdi. Construction of abstract state graphs with pvs. In *Proceedings of 9th Conference on Computer-Aided Verification*, 1997.

[HPR94] N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In B. LeCharlier, editor, *International Symposium on Static Analysis, SAS'94*, Namur (Belgique), September 1994. LNCS 864, Springer Verlag.

[LHR97] D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *24th ACM Symposium on Principles of Programming Languages, POPL'97*, Paris, January 1997.

[LS97] D. Lesens and H. Saïdi. Experiments in finding invariants of abstract networks. Technical Report, VERIMAG, 1997.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming.* LNCS 137, Springer Verlag, April 1982.

[Saï97] H. Saïdi. The invariant-checker: Deductive verification of reactive systems. In *Proceedings of 9th Conference on Computer-Aided Verification*, 1997.

Appendix

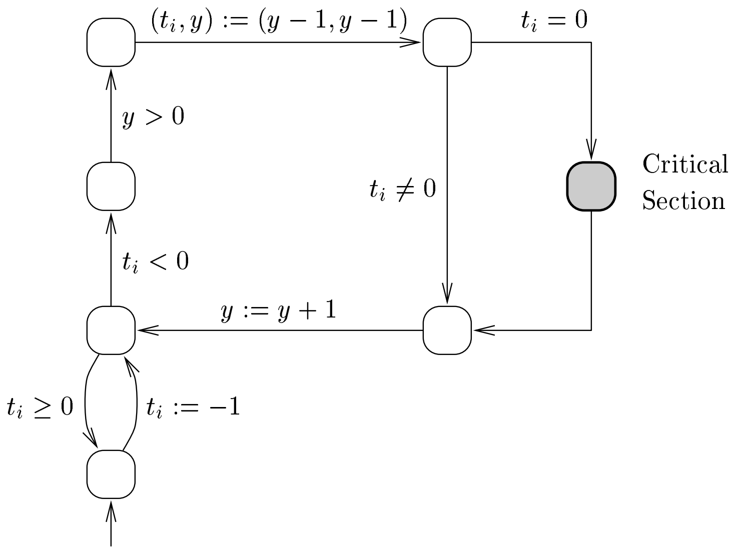

Figure 1: A process of the mux_ast system.

t_i is a local variable, y a shared one. As soon as a process attempts to enter its critical section, the predicate $y \leq 0$ is true.

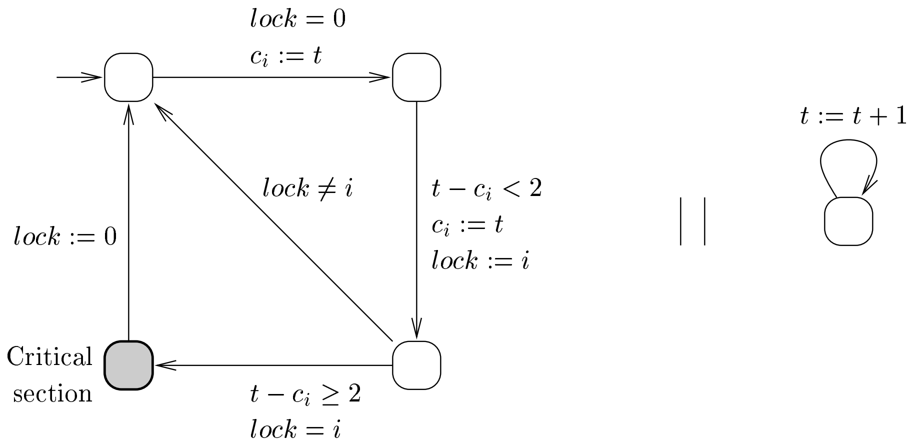

Figure 2: A process of the Fischer's protocol

lock is a shared variable. c_i is a local timer, which can be reset by the process. Before entering its critical section, a process must wait that all processes which have passed the lock have performed their assignment.

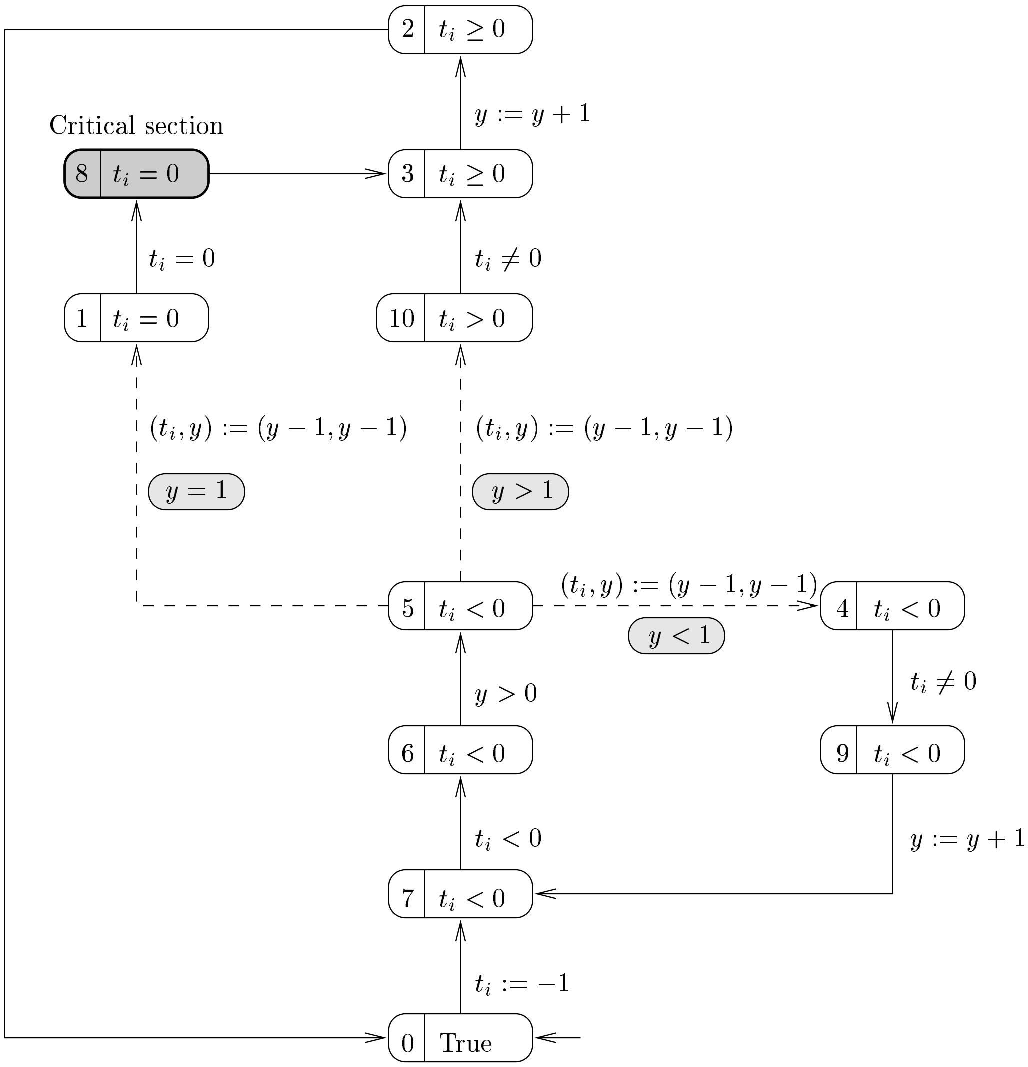

Figure 3: An abstract process of the mux_ast system

This transition system is constructed from the one showed fig. 1. Variable t_i has been abstracted, it is showed on transitions only for convenience. On each state, a predicate on t_i must be satisfied. In the abstract process, dashed transitions are undeterministic. The predicate associated with the target state allows us to compute a precondition on y: to reach predicates $t = 0$, $t > 0$ and $t < 0$ with the assignment $t := y - 1$, y must satisfy respectively preconditions $y = 1$, $y > 1$ and $y < 1$ (these preconditions on y are showed in gray). For example,

$$(y = 1) \;\equiv\; \exists \{t_i\}.pre\{\tau_i\}(t = 0)$$