Auxiliary Constructs for Proving Liveness in Compassion Discrete Systems

Teng Long and Wenhui Zhang
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences, Beijing, China
{longteng,zwh}@ios.ac.cn

Abstract. For proving response properties for systems with compassion requirements, a deduction rule is introduced in [13]. In order to use the rule, auxiliary constructs are needed. They include helpful assertions and ranking functions defined on a well-founded domain. Along the line of the work [2] that computes ranking functions for response properties for systems with justice requirements, we develop an approach which extends that of [2] for computing ranking functions for systems with compassion requirements. We illustrate the use of the approach on three examples.

1 Introduction

Model checking is a main verification technique for finite state systems, and has been successfully applied to proving the correctness of hardware and software designs. However, in the general case, the real systems in the world may be (practically) infinite, such that model checking cannot be directly used to prove the correctness of such systems.

The concept of abstraction helps enhancing the applicability of model checking to infinite systems. Predicate abstraction [8, 4, 5], has been useful for verification of safety properties of infinite systems, and for verification of liveness properties. Ranking abstraction has been introduced in [9, 3, 10] recognizing that the usual state abstraction is often inadequate in order to capture liveness properties, in which compassion requirements are introduced into the abstract system such that the abstraction preserves the liveness properties under consideration.

One of the common features of these two methods is that we need to extract auxiliary constructs in order to make the methods successful in proving safety and liveness properties. In the former case, one needs to construct invariants and in the latter, one needs to construct ranking functions.

Our focus is on methods for computing ranking functions for proving liveness properties. Invisible ranking introduced in [6] is one such method for automatically generating helpful assertions and ranking functions for proving liveness properties in systems with justice requirements. The method was then extended to handle a large class of problems by relaxing restrictions requiring that the helpful assertions and ranking functions only depend on the local states of a process [7]. For dealing with ranking abstraction in proving liveness properties

in systems with justice requirements, an approach is presented in [2] based on graph manipulation for generating helpful assertions and ranking functions.

Along this line of research, our approach presented in this paper extends that of [2] in order to be able to compute ranking functions for proving liveness properties in sequential and concurrent programs with compassion requirements including those introduced with ranking abstraction and those stated explicitly in the compassion discrete system. Our approach may as well be used for proving liveness properties with the use of predicate abstraction (when ranking abstraction does not provide additional useful compassion requirements).

The rest of this paper is organized as follows. In Section 2 we introduce the basic concepts used in the approach. It includes the computational model FDS (fair discrete system) and CDS (compassion discrete system) with its related notions of fairness, the rule RESPONSE [13] for the deductive proof of response properties of CDS. Section 3 presents the approach for computing the auxiliary constructs, and Section 4 illustrates the application of the approach on three examples. Finally, concluding remarks are contained in Section 5.

2 Preliminaries

We introduce the computational model with fairness requirements [11], and the rule for proving response properties [13].

2.1 Computational Model

A fair discrete system (FDS) is a quintuple $D = \langle V, \Theta, \rho, J, C \rangle$ where the components are as follows.

- V : A finite set of typed *system variables*, containing data and control variables. A set of states (interpretation) over V is denoted by Σ. For a state s and a system variable $v \in V$, we denote by s[v] the value assigned to v by the state s.
- Θ : The *initial condition* - an assertion (state formula) characterizing the initial states.
- ρ : The *transition relation* - an assertion $\rho(V, V')$, relating the variables in state $s \in \Sigma$ to the V' in a D-successor state $s' \in \Sigma$.
- J : A set of justice requirements (weak fairness). The justice requirement $J \in \mathcal{J}$ is an assertion which guarantee that every computation should include infinitely many states satisfying J.
- C : A set of compassion requirements (strong fairness). The compassion requirement $\langle p, q \rangle \in C$ is a pair of assertions, which guarantee that every computation should include either only finitely many p-states, or infinitely many q-states.

Computation A computation of D is an infinite sequence of state $\sigma : s_0, s_1, s_2, \ldots$, satisfying the following requirements: (1) $s_0 \models \Theta$. (2) For each j = 0, 1, ..., the state s_{j+1} is in a D-successor of the state s_j. For each $v \in V$, we interpret v as $s_l[v]$ and v' as $s_{l+1}[v]$, that is $\langle s_l, s_{l+1} \rangle \models \rho(V, V')$.

Justice A computation σ is *just*, if σ contains infinitely many occurrences of J-states for every $J \in \mathcal{J}$. A *justice discrete system* (JDS) is an FDS with no compassion requirements.

Compassion A computation σ is *compassionate*, if σ contains only finitely many p-states, or σ contains infinitely many q-states, for every $\langle p, q\rangle \in C$. A *compassion discrete system* (CDS) is an FDS with no justice requirements.

2.2 Proof Rule for Response Properties

The problem is that given an FDS D and a response property of the form $p \Rightarrow \Diamond q$, where p and q are assertions, we want to prove the following statement

$$D \models (p \Rightarrow \Diamond q)$$

Since an FDS is equivalent to a CDS[1], it is sufficient to consider CDS only. For proving response properties over a CDS, the rule RESPONSE [13] was developed for systems with compassion requirements (this rule is hereafter referred to as C-RESPONSE for emphasizing that it involves compassion requirements). The rule is shown in Fig. 1.

Let p, q be assertions.
Let $\mathcal{A} : (W, \succ)$ be a well-founded domain.
Let $\{\varphi_i \mid i \in \{1, ..., n\}\}$ be a set of assertions.
Let $\{F_i = \langle p_i, q_i\rangle \mid i \in \{1, ..., n\}\}$ be a set of compassion requirements.
Let $\{\Delta_i : \Sigma \to W \mid i \in \{1, ..., n\}\}$ be a set of ranking functions.

R1	p	$\Rightarrow q \vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j)$
For each i:		
R2	$p_i \wedge \varphi_i \wedge \rho$	$\Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j)$
R3	$\varphi_i \wedge \rho$	$\Rightarrow q' \vee (\varphi'_i \wedge \Delta_i = \Delta'_i) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_i \succ \Delta'_j)$
R4	φ_i	$\Rightarrow \neg q_i$
	p	$\Rightarrow \Diamond q$

Fig. 1. Proof Rule: C-RESPONSE

The use of the rule requires: a well-founded domain $\mathcal{A}$, and for each compassion requirement $\langle p_i, q_i\rangle$, a helpful assertion φ_i and a ranking function $\Delta_i : \Sigma \mapsto W$ mapping states of D to elements of $\mathcal{A}$.

R1 requires that any p-state is either a goal state (i.e., a q-state), or a $(p_i \wedge \varphi_i)$-state for some $i \in \{1, \ldots, n\}$. R2 requires that any step from a $(p_i \wedge \varphi_i)$-state moves either directly to a q-state, or to another $(p_j \wedge \varphi_j)$-state, or stays at a state of the same type (i.e., a $(p_i \wedge \varphi_i)$-state). R3 requires that any step from

[1] The justice requirement can be expressed as the degenerate compassion requirement $\langle 1, J\rangle$, where 1 denotes the assertion *True* which holds at every state.

a φ_i-state moves either directly to a q-state, or to another $(p_j \wedge \varphi_j)$-state with decreasing rank, or stays at a state of the same type with the same rank. R4 together with the previous rules guarantees that if an execution does not satisfy $\Diamond q$, then it violates the compassion requirement.

3 Proving a Response Property

Given a CDS D and a response property $\psi : p \Rightarrow \Diamond q$. In order to be able to use the proof rule C-RESPONSE for proving the property, we have to define a well-founded domain $\mathcal{A}$, and for each compassion requirement $\langle p_i, q_i \rangle$, define a helpful assertion φ_i and a ranking function $\Delta_i : \Sigma \mapsto W$ mapping states of D to elements of $\mathcal{A}$.

The phases for proving $D \models \psi$ including those of computing the helpful assertions and ranking functions are as follows:

(1) use ranking abstraction [9, 3, 2] and construct D^α and ψ^α from D and ψ, and then construct a pending graph [2] based on D^α.

(2) construct an initial rank for each node (except the goal state) of the pending graph and a set of helpful compassion requirements associated to each of these nodes of the pending graph.

(3) construct an abstract graph from the pending graph, such that each node in the abstract graph represents a subset of the nodes of the pending graph, then construct $\mathcal{A}$, and for each node, construct φ_i and Δ_i, and make an association of some $F_i = (p_i, q_i) \in \mathcal{F}$ to the node. Note that according to the construction, one F_i may correspond to several abstract nodes.

(4) use the helpful constructs to prove the response property $D \models \psi$ by the proof rule.

3.1 Ranking Abstraction and Pending Graph

This step is carried out according to the technique of ranking abstraction [9, 3, 2] and pending graph [2].

Ranking abstraction, as explained in [2], is a method of augmenting the concrete program by a non-constraining progress monitor, which measures the progress of program execution, relative to a given ranking function. In order to distinguish this kind of ranking functions from the ranking functions in the proof rule C-RESPONSE, we call this kine of ranking functions ARFs (augmenting ranking functions) in the sequel. Once a program is augmented, a conventional state abstraction can be used. In such a way, the state abstraction can preserve the ability to monitor progress in the abstract system.

For a system $S = \langle V, \Theta, \rho, J, C \rangle$ (in which J is empty when CDS is considered) and a well-founded domain $(W, \prec)$, let δ be an ARF over W, let dec_δ be a fresh variable, the augmentation of S by δ is

$$S + \delta : \langle V \cup \{dec_\delta\}, \Theta, \rho \wedge \rho_\delta, J, C \cup \{(dec_\delta > 0, dec_\delta < 0)\} \rangle$$

where ρ_δ is defined by

$$dec'_\delta = \begin{cases} 1 & \delta \succ \delta' \\ 0 & \delta = \delta' \\ -1 & otherwise \end{cases}$$

A system may be augmented with a set of ARFs $\{\delta_1, ..., \delta_k\}$. Then predicate abstraction may be applied. In the predicate abstraction, it is not necessary to abstract variables of the form dec_δ since it ranges over the finite domain $\{-1, 0, 1\}$, and the abstraction preserves the compassion requirement ($dec_\delta > 0, dec_\delta < 0$).

Assuming that we already have an abstract program D^α from D constructed by the above process with the abstraction map α, a pending graph [2] is then constructed from D^α. Let us denote the graph by $G = \langle N, E \rangle$. The set of nodes N are those satisfying $pend \vee g$ where $pend$ characterizes the states reachable from a p-state by a q-free path, and g is a q^α-state reachable from a pending state in one step. The set of edges E consists of all transitions connecting two pending states and the edges connecting $pend$ nodes to the goal node g.

The set of nodes of G may be written as $\{S_0, S_1, ..., S_m\}$ where $S_0 = g$ is the goal state and $S_1, ..., S_m$ are pending states. This is the starting point of our algorithm for computing the helpful constructs for the proof rule.

3.2 Helpful Compassion Requirements and Initial Ranks

The ranking functions in the abstract program are represented as a mapping $N \to TUPLES$, where TUPLES is the type of lexicographic tuples whose elements are either natural numbers or ARFs. For simplicity, we call such a tuple as a "rank". Let R_l and H_l be respectively the rank and the list of helpful compassion requirements for $S_l \in N$. For convenience, we write q for q^α, and similarly for other formulas and constructions. Let $F_1, ..., F_n$ be the original compassion requirements, and $F_{n+1}, ..., F_{n'}$ be the dec-requirements (the compassion requirements introduced by the ranking abstraction). The procedure for computing R_l and H_l (which are initially empty) is described in Algorithm 1. The main functions are explained as follows.

decompose(G) The graph G is decomposed into a set of MSCCs (maximal strongly connected components), denoted $C_0, ..., C_k$. They are ordered such that if C_i is reachable from C_j, then $i < j$. For MSCCs not connected to each other, their indices may be in an arbitrary order.

violate(C_i, F_j) The MSCC (may be the trivial one) C_i violates the compassion requirement $F_j = (p_j, q_j)$, if p_j is satisfied by some node of the MSCC and q_j is not satisfied by any node of the MSCC.

term(F_j) For a dec-requirement F_j of the form $\langle dec_\delta > 0, dec_\delta < 0 \rangle$, $term(F_j) = \delta$.

Algorithm 1 C-RANK(G)

```
1: decompose(G) into a set of MSCCs [C_0, ..., C_k];
2: for i=0, i ≤ k, i++ do
3:    for each S_l of C_i, append i to R_l;
4: end for
5: for i=0, i ≤ k, i++ do
6:    if the goal state g is in C_i then
7:       continue;
8:    end if
9:    for j=1; j ≤ n'; j++ do
10:      if not violate(C_i,F_j) then
11:         continue;
12:      end if
13:      for each S_l of C_i do
14:         append F_j to H_l;
15:      end for
16:      if j > n then
17:         for each S_l of C_i do
18:            append term(F_j) to R_l;
19:         end for
20:      end if
21:      if C_i is a trivial MSCC then
22:         break;
23:      else
24:         removeedge(C_i,F_j); C-RANK(C_i); break;
25:      end if
26:   end for
27: end for
```

removeedge(C_i, F_j) Given an MSCC C_i and a compassion requirement $F_j = (p_j, q_j)$, this procedure modifies the MSCC in such a way that one node satisfying p_j is identified and all incoming edges of such a node in this MSCC are removed.

3.3 Abstract Nodes, Helpful Assertions and Ranks

According to H_l, we construct the abstract nodes as an assertion Φ by grouping together certain nodes that need satisfying the same helpful compassion requirements. Let G' be the abstract graph, initially empty, i.e., $G' = (\{\}, \{\})$. The procedure for constructing G' is described in Algorithm 2. The main functions are explained as follows.

subgraph(G, F_i) In the assignment $W_i = subgraph(G, F_i)$, the variable W_i is a local variable used to hold a subset of nodes (a subgraph). The nodes of the subgraph is constructed according to F_i (non-dec-requirements) as follows: $S_l \in W_i$ iff $F_i \in H_l$. Then W_i is considered as a derived subgraph of G with the nodes as specified.

Algorithm 2 C-GRAPH

```
1: call C-RANK(G);
2: for each F_i ∈ {F_1, ..., F_n} do
3:    W_i=subgraph(G, F_i);
4: end for
5: for each W_i ∈ {W_1, ..., W_n} do
6:    createandmergenodes(W_i, G');
7: end for
8: createedges(G');
```

createandmergenodes(W_i, G') (1) W_i may contain several different MSCCs violating F_i. For each such MSCC in W_i, a node representing this MSCC is created and added to G'. The rank Δ_l of the abstract node Φ_l is assigned the rank obtained before the MSCC is split into smaller MSCCs. (2) For the nodes created previously, they are merges according to the following condition: the states that the nodes represent differ only in the dec-variables introduced in the ranking abstraction. Then the rank of the abstract node is assigned the lowest rank of the nodes represented by the abstract node. (3) For each node Φ_l in the final abstract graph G', the concrete helpful assertion $\varphi_l = \alpha^{-1}(\Phi_l)$ is obtained by concretizing the abstract nodes (viewed as abstract assertions, by making correspondence between formulas and sets of states).

createedges(G') For each pair of nodes Φ and Φ' such that $\Phi' \not\subset \Phi$, if some node of Φ is connected to some of Φ', an edge from Φ to Φ' is created.

3.4 Claim of the Helpful Constructions

For each compassion requirement (p_i, q_i), several abstract states may be associated. We consider the set of compassion requirements as a multiset, such that one compassion requirement (p_i, q_i) has the number of occurrences matching the number of associated abstract states. Then each (occurrence of a) compassion requirement corresponds to one abstract state (with a rank and a helpful assertion) associated with it. These helpful constructs are then to be used in the proof rule for proving the response property $D \models \Psi$.

Ranking Core For the correctness, we assume that every ranking function in the ranking abstraction is chosen to be a variable. Such a set of variables (representing a set of ranking functions) are called ranking core $\mathcal{R}$ [9]. It is easily seen that the proof of the correctness of the above algorithms with this assumption can be extended to ranking functions that are arithmetic terms. The abstraction of D according to the abstraction map α and the ranking core $\mathcal{R}$ is denoted $\mathcal{D}^{\mathcal{R},\alpha}$.

Main Claim *Let CDS $\mathcal{D}$, a ranking core $\mathcal{R}$, an abstraction mapping α and the property Ψ be given. Let the assertions φ_i and ranking functions Δ_i be those extracted by C-GRAPH. If $\mathcal{D}^{\mathcal{R},\alpha} \models \Psi^\alpha$ then R_1-R_4 of the rule C-RESPONSE*

are provable with the extracted helpful constructs (with the set of compassion requirements reorganized as a suitable multiset).

Correctness The correctness is then established by analyzing the different steps in the construction of the helpful constructs.

Let $\Delta > \Delta'$ (similarly $R > R'$), where Δ, Δ' represent ranks and R, R' represent ranks, denote that Δ appear after Δ' according to the lexicographic order. Let $\Delta \succ_E \Delta'$ denote that Δ appear after Δ' according to the lexicographic order with the following conditions: (1) each variable x of the ranking core $\mathcal{R}$ appearing in Δ' are replaced by x', (2) the lexicographic order is augmented by an environment E that specifies whether $x > x'$ or $x = x'$. The environment E may be replaced by a state S that reflects whether the value of a variable is decreased when the program moves to the state S.

Claim 1 *Let S_i and S_j be states in the pending graph. The following properties hold.*

P_1. If $\Delta_i \succ_{S_j} \Delta_j$ and $\Delta_j > \Delta_k$, then $\Delta_i \succ_{S_j} \Delta_k$.
P_2. If the states S_i and S_j agree on the values of their non-dec variables, then they have the same set of successors.

P_1 is true according to the definition. P_2 is true according to the construction of the pending graph. These properties are the same as those stated in [2].

Claim 2 *Let R_i and R_j be the associated ranks of S_i and S_j. Then on successful termination of C-RANK, the following properties hold.*

P_3. For every two states S_i, S_j belonging to different MSCCs such that S_i is connected to S_j, there is a rank decrease $R_i \succ_{S_j} R_j$.
P_4. For every two states S_i, S_j belonging to one MSCC such that S_i is connected to S_j, there is no rank decrease only if the MSCC violates some compassion requirement (p_k, q_k) (non-dec-requirements) and there is at least a state S_k which $S_k \models p_k$, or the MSCC does not violate any compassion requirement.

P_3 follows from the decomposition of the pending graph into MSCCs. P_4 follows from the way MSCCs being modified when they do not satisfy some compassion requirements.

Claim 3 *Let Δ_i and Δ_j be the associated ranks of Φ_i and Φ_j. Then on termination of C-GRAPH, the following properties hold.*

P_5. If Φ_i is connected to Φ_j in the abstract graph and $S \in \Phi_j$, then there is a Φ_k such that $S \in \Phi_k$, $\Delta_i \succ_S \Delta_k$ and $S \models p_k$ where (p_k, q_k) is the compassion requirement violated by the MSCC represented by Φ_k.
P_6. Let $s[\Delta]$ be Δ with the variables replaced by their value in the state s. If concrete states s, s' satisfy $s \models \varphi_i$, and $s' \models \varphi_j$, $i \neq j$ and s' is a D-successor of s, then there is a φ_k such that $s' \models \varphi_k \wedge p_k$ and $s[\Delta_i] \succ s'[\Delta_k]$.

P_5 follows from the construction of the abstract graph. Φ_k in P_5 is necessarily a superset of Φ_j (i.e., $\Phi_j \subseteq \Phi_k$), when Φ_k is considered as a set of nodes of the pending graph. P_6 follows from P_5 by the soundness of the abstraction.

Proof of the main claim. Let $\Phi_0 = g, \Phi_1, ..., \Phi_n$ be the nodes in the abstract graph, and $\varphi_i, \Delta_i, (p_i, q_i)$ be the helpful assertion, rank, and compassion requirement (which has been reorganized into a multiset that matches the number of Φ_i) associated to Φ_i for $i = 1, ..., n$. Assume $D^\alpha \models \Psi^\alpha$. (1) Since Ψ^α is true, the disjunction of the abstract states $g, \Phi_1, ..., \Phi_n$ covers the states in the pending graph. By the correctness of the abstraction, $g \vee \bigvee_{i=1}^{l} \varphi_i$ covers the state space represented by the pending graph. The a p-state is either the goal state g or a state with a progress requirement, i.e. $p_i \wedge \varphi_i$ for some $i \in \{1, ..., n\}$. (2) Similarly, successor states of such a state (excluding g) also satisfy the same condition. (3) The correctness of R_3 follows from property P_6. (4) R_4 is guaranteed by the construction of φ_i, since each φ_i represents an MSCC (or a collection of MSCCs when they are merged) violating the compassion requirement (p_i, q_i).

3.5 Discussion

Previous works in this direction of research include using deduction rules with weak fairness (justice) requirements to prove liveness properties of sequential or simple concurrent programs. They depend on Dec-requirements to decide the ranks of states in just MSCC. Our approach concerns deductive rule with strong fairness requirements to prove liveness properties of more complex concurrent programs. It depends on compassion requirements to decide the ranks of states in MSCC. A comparison of the algorithm RANK-JUST-GRAPH [2] and C-GRAPH is given in Fig.2.

Algorithm	RANK-JUST-GRAPH	C-GRAPH
Deductive Rule	JUST-RESPONSE	C-RESPONSE
Fairness of Systems	Weak fairness	Fairness
Control of Unfair Loops	Depend on Dec-requirements to decide the ranks of states in MSCC	Depend on compassion requirements to decide the ranks of states in MSCC

Fig. 2. Algorithms for Computing Auxiliary Constructs

4 Examples

We illustrate the application of the approach on three programs:

– MUX-SEM, a concurrent program for mutual exclusion [1].
This example is supposed to show the approach applied on a verification problem with predicate abstraction.

- COND-TERM, a sequential program with a non-deterministic choice of the values of a variable [13].
  This example is supposed to show the approach applied on a verification problem with predicate abstraction in which some Dec-requirement is introduced in the abstraction phase. The pending graph includes two kinds of uncompassionate loops, one violating the given compassion and the other violating the compassion introduced in the abstraction phase.
- UP-DOWN, a concurrent program with only justice requirements [12].
  This example is supposed to show some differences and similarities between this approach and the approach presented in [2] when compassion requirements in the system are restricted to justice requirements.

4.1 Example 1: MUX-SEM

Fig. 3 shows the program MUX-SEM. Let $at\_l_i[j]$ denotes that process j is at l_i (of process j). The response property we wish to establish is $at\_l_2[1] \Rightarrow \Diamond at\_l_3[1]$. The just requirements are $\neg at\_l_4[j]$ and $\neg at\_l_3[j]$. The compassion requirement is $F_1 = \langle at\_l_2[1] \wedge y = 1, at\_l_3[1] \rangle$. The just requirements are special compassion requirements formulated as $\langle 1, \neg at\_l_4[i] \rangle$ for $i \in \{1, ..., n\}$ and $\langle 1, \neg at\_l_3[i] \rangle$ for $i \in \{1, ..., n\}$.

$$\textbf{local}\, y : \textbf{boolean} \quad \textbf{init}\, y = 1;$$
$$\|_{i=1}^{n} P[i] :: \left[\begin{array}{l} \textbf{loop forever do} \\ \quad l_1 : \textbf{Noncritical} \\ \quad l_2 : \textbf{request}\, y \\ \quad l_3 : \textbf{Critical} \\ \quad l_4 : \textbf{release}\, y \end{array}\right]$$

Fig. 3. Program MUX-SEM

The abstraction mapping α is defined by:

$$\alpha : \Pi = \pi, \Pi_3 = \pi_3, \Pi_4 = \pi_4, Y = (y > 0)$$

where Π is a function with range $\{1, 2, 3, 4\}$ (and the domain being the system states). $\Pi = i$ denotes that $at\_l_i[1]$ is true, for $i \in \{1, ..., 4\}$. Π_k is a function with range $\{0, 1\}$ and it is 1 iff the following is true:

$$\bigvee_{j=2}^{n} (at\_l_k[j] \wedge \bigwedge_{i=2}^{n} ((i \neq j) \Rightarrow \neg at\_l_k[i]))$$

The set of compassion requirements $\{\langle 1, \neg at\_l_4[i] \rangle \mid i = 1, ..., n\}$ and the set $\{\langle 1, \neg at\_l_3[i] \rangle \mid i = 1, ..., n\}$ induce two new compassion requirements $F_2 = \langle 1, \neg \pi_4 = 1 \rangle$ and $F_3 = \langle 1, \neg \pi_3 = 1 \rangle$. Then the set of compassion requirements of the abstract program is $\mathcal{F} = \{F_1, F_2, F_3\}$.

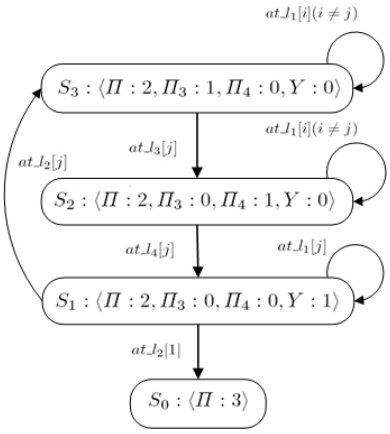

Fig. 4. The Pending Graph

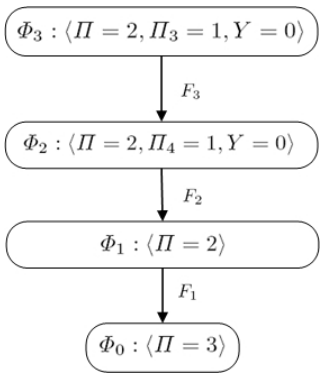

Fig. 5. The Abstract Graph

Let $-$ denote any value in the range of the respective position of the abstract states. For instance, $(1, -, -, -)$ denote the abstract states where the value of (Π, Π_3, Π_4, Y) satisfying $\Pi = 1$ and the rest of the positions could be any value. Then the abstract states represented by the following tuples covers the reachable concrete states

$$(1, -, -, -)$$
$$(2, 0, 0, 1)$$
$$(2, 1, 0, 0)$$
$$(2, 0, 1, 0)$$
$$(3, -, -, -)$$
$$(4, -, -, -)$$

Let S_0, S_1, S_2, S_3 be the set of states represented by respectively

$$(3, -, -, -), (2, 0, 0, 1), (2, 0, 1, 0), (2, 1, 0, 0).$$

Then we construct the pending graph (shown in shown in Fig. 4) with these four states with $S_0 = g$. Let R_i and H_i be the rank of S_i and the set of helpful compassion requirements associated to S_i, respectively, initially with $R_i = []$ and $H_i = []$.

We have two MSCCs: $\{S_3, S_2, S_1\}$ and $\{S_0\}$. Then $R_0 = [0], R_1 = R_2 = R_3 = [1]$.

By checking the MSCCs $\{S_3, S_2, S_1\}$ and $\{S_0\}$, we find that the first one violates the compassion requirement $F_1 = \langle at\_l_2[1] \wedge y = 1, at\_l_3[1] \rangle$. Then $H_1 = H_2 = H_3 = [F_1]$.

We remove the edges (S_1, S_3) and (S_1, S_1), then apply C-RANK again. We obtain 3 MSCCs $\{S_1\}, \{S_2\}, \{S_3\}$. Then $R_3 = [1, 2], R_2 = [1, 1], R_1 = [1, 0]$.

By checking the 3 MSCCs, we find that $\{S_2\}$ violates $F_2 = \langle 1, \neg \pi_4 = 1 \rangle$ and $\{S_3\}$ violates $F_3 = \langle 1, \neg \pi_3 = 1 \rangle$. The compassion requirements are then added

to the respective nodes, such that $H_3 = [F_1, F_3], H_2 = [F_1, F_3]$. The final value of R_i and H_i are as shown in Table 1.

Index i	S_i	R_i	H_i
3	S_3	$[1, 2]$	$[F_1, F_3]$
2	S_2	$[1, 1]$	$[F_1, F_2]$
1	S_1	$[1, 0]$	$[F_1]$
0	S_0	$[0]$	

Table 1. The Rank Table of Program MUX-SEM

According to H_i, we construct the abstract nodes by grouping together nodes that need satisfying the same compassion requirement. The abstract nodes of the program MUX-SEM with their respective ranks[2] are listed in Table 2 and the abstract graph is shown in Fig. 5. In the abstract graph, we mark "F_i" as the label of edges to illuminate that the program must follow the transitions because of the compassion requirements. The transitions which do not relate to the compassion requirements are not shown in the abstract diagram.

Abstract Node	*Nodes*	*Rank*	*Compassion Req.*
Φ_3	S_3	$[1, 2]$	F_3
Φ_2	S_2	$[1, 1]$	F_2
Φ_1	S_1, S_2, S_3	$[1]$	F_1
Φ_0	S_0	$[0]$	

Table 2. The Abstract Table of Program MUX-SEM

Finally, we obtain the concrete helpful assertions $\varphi_1, \varphi_2, \varphi_3$ by concretizing the abstract assertions Φ_1, Φ_2, Φ_3, and obtain the ranks $\Delta_1, \Delta_2, \Delta_3$ by renumbering the respective ranks in Fig. 5. The concrete assertions $\varphi_1, \varphi_2, \varphi_3$ and the ranks $\Delta_1, \Delta_2, \Delta_3$ are shown in the table in Table 3.

Relating to the original program with n processes, let $(3, j)$ denote the states where process j is at label l_3 and $(4, j)$ denote the states where process j is at label l_4 for $j = 1, ..., n$, we obtain Table 4.

The validity of the premises of rule C-RESPONSE for example 1 has been verified with the constructed auxiliary constructs $\varphi_1, \varphi_2, \varphi_3$ and $\Delta_1, \Delta_2, \Delta_3$. The reader is referred to Appendix A.1 for the details.

[2] Note that F_1 is associated to the nodes in the first level of computation of ranks, therefore the rank of Φ_1 is only concerned with its first level projection.

Index i	φ_i	Δ_i
3	$at\_l_2[1] \wedge \bigvee_{j=2}^{n}(at\_l_3[j] \wedge \bigwedge_{i=2}^{n}((i \neq j) \Rightarrow \neg at\_l_3[i])) \wedge y = 0$	[2]
2	$at\_l_2[1] \wedge \bigvee_{j=2}^{n}(at\_l_4[j] \wedge \bigwedge_{i=2}^{n}((i \neq j) \Rightarrow \neg at\_l_4[i])) \wedge y = 0$	[1]
1	$at\_l_2[1]$	[0]

Table 3. The Concrete Table of MUX-SEM Program

index i	φ_i	Δ_i
$(3, j), j > 1$	$at\_l_2[1] \wedge at\_l_3[j] \wedge y = 0$	[2]
$(4, j), j > 1$	$at\_l_2[1] \wedge at\_l_4[j] \wedge y = 0$	[1]
$(-, 1)$	$at\_l_2[1]$	[0]

Table 4. The Concrete Table of MUX-SEM Program

4.2 Example 2: COND-TERM

Fig. 6 shows the program COND-TERM. The response property we wish to establish is $\Psi : at\_l_1 \Rightarrow \Diamond at\_l_4$. The just requirements are $\neg at\_l_i$ for $i = 1, 2, 3, 4$, and the compassion requirements is $F_1 = \langle at\_l_3 \wedge x = 0, 0\rangle$. Let F_{i+1} be $\langle 1, \neg at\_l_i\rangle$ representing the just requirements for $i = 1, 2, 3$.

x,y: natural init x = 0
l_1: **while** $y > 0$ **do**
l_2: x:= {0,1}
l_3: $y := y + 1 - 2x$
l_4:

Fig. 6. Program COND-TERM

Let $dec_y = sign(y - y')$ in which y denotes the value of y in the previous state and y' denotes the value of y in the current state. The abstraction mapping α is defined by :

$$\alpha : \Pi = \pi, X = (x > 0), Y = (y > 0), Dec_y = dec_y$$

where $\Pi = i$ denotes at_l_i is true. We construct the pending graph as showing in Fig. 7. The constraints of the graph include the additional compassion requirement $F_D = \langle Dec_y > 0, Dec_y < 0\rangle$ which is deduced from the condition of the while loop according to the rank abstraction process.

There are 8 states $\{S_0, S_1, ..., S_7\}$ with $S_0 = g$. Let R_i and H_i be the rank of S_i and the set of helpful compassion requirements associated to S_i, respectively, initially with $R_i = []$ and $H_i = []$.

In the first level of computation, we have 4 MSCCs: $\{g\}$, $\{S_1\}$, $\{S_2, S_3, S_4, S_5, S_6\}$, $\{S_7\}$. Then $R_0 = [0], R_1 = [1], R_2 = R_3 = R_4 = R_5 = R_6 = [2], R_7 = [3]$.

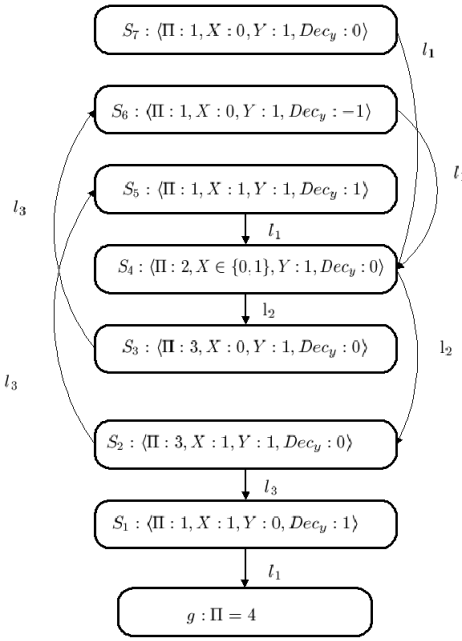

Fig. 7. The Pending Graph

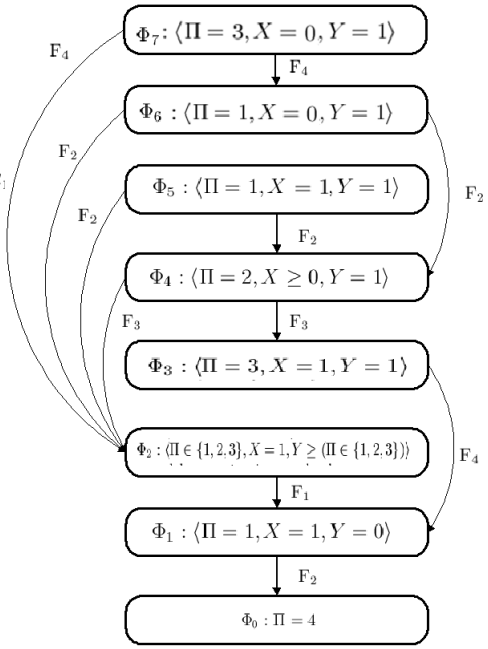

Fig. 8. The Abstract Graph

Let C_0, C_1, C_2, C_3 denote the 4 MSCCs. By checking the MSCCs (excluding C_0) against the compassion requirements, we obtain the table as shown in Table 5.

Component	Violation
C_1	F_2
C_2	F_1
C_3	F_2

Table 5. The MSCCs C_1, C_2, C_3

Component	Violation
C_{21}	F_4
C_{22}	F_2
C_{23}	F_D

Table 6. The MSCCs C_{21}, C_{22}, C_{23}

Then we add the respective compassion requirement to $H_1, ..., H_7$ such that $H_1 = H_7 = [F_2], H_2 = H_3 = H_4 = H_5 = H_6 = [F_1]$.

Since C_2 is not a non-trivial subgraph, we remove the edge $(S_4 \rightarrow S_3)$, which leads into the state satisfying $at\_l_3 \wedge x = 0$, from C_2, and compute again with the modified subgraph.

In the second level of computation, we have 3 MSCCs: $\{S_3\}$, $\{S_6\}$, $\{S_4, S_2, S_5\}$. Then $R_3 = [2, 2], R_6 = [2, 1], R_2 = R_4 = R_5 = [2, 0]$.

Let C_{21}, C_{22}, C_{23} denote the 3 MSCCs. By checking the MSCCs against the compassion requirements, we obtain the table as shown in Table 6.

Then we add the respective compassion requirement to $H_2, ..., H_6$. In addition, since C_{23}=$\{S_4,\ S_2,\ S_5\}$ violates $F_D = \langle Dec_y > 0, Dec_y < 0\rangle$, we add y to R_2, R_4, R_5. Then since C_{23} is not a non-trivial subgraph, we remove the edge $(S_2 \rightarrow S_5)$, which leads into the state satisfying $Dec_y > 0$, from C_{23}, and compute again with the modified subgraph.

In the third level of computation, we have 3 MSCCs: $\{S_5\}$, $\{S_4\}$, $\{S_2\}$. The rank to be assigned to the nodes in this level is $2, 1, 0$.

Then $R_5 = [2, 1, y, 2], R_4 = [2, 1, y, 1], R_2 = [2, 1, y, 0]$. Since $\{S_5\}$, $\{S_4\}$ and $\{S_2\}$ are trivial MSCCs, we add F_2, F_3, F_4 to H_5, H_4, H_2 respectively. The final value of R_i and H_i are as shown in Table 7.

Index i	S_i	R_i	H_i
7	S_7	$[3]$	$[F_2]$
6	S_6	$[2, 1]$	$[F_1, F_2]$
5	S_5	$[2, 0, y, 2]$	$[F_1, F_D, F_2]$
4	S_4	$[2, 0, y, 1]$	$[F_1, F_D, F_3]$
3	S_3	$[2, 2]$	$[F_1, F_4]$
2	S_2	$[2, 0, y, 0]$	$[F_1, F_D, F_4]$
1	S_1	$[1]$	$[F_2]$
0	S_0	$[0]$	

Table 7. The Rank Table of Program COND-TERM

According to H_i, we construct the abstract nodes [3] by grouping together nodes that need satisfying the same compassion requirement merging S_6 - S_2 as an abstract state Φ_2 and by grouping together nodes that agree with the value of all variables except Dec: merging S_7 and S_5 as another abstract state Φ_6.

The abstract nodes with their respective ranks are listed in Table 8 and the abstract graph is shown in Fig. 8.

Finally, we obtain the concrete helpful assertions $\varphi_1, \ldots, \varphi_7$ by concretizing the abstract assertions $\Phi_1, \ldots, \Phi_7$, and obtain the ranks $\Delta_1, \ldots, \Delta_7$ by renumbering the respective ranks in Fig. 8. The helpful assertions and the ranks are shown in the table in Table 9.

The validity of the premises of rule C-RESPONSE for example 2 has been verified with the constructed auxiliary constructs $\varphi_1, ..., \varphi_7$ and $\Delta_1, ..., \Delta_7$. The reader is referred to Appendix A.2 for the details.

[3] Note that F_D is not involved in the construction of abstract nodes, since it is not one of the original system constraints.

Abstract Node	*Nodes*	*Rank*	*Compassion Req.*
Φ_7	S_3	$[2,2]$	F_4
Φ_6	S_6, S_7	$[2,1]$	F_2
Φ_5	S_5	$[2,0,y,2]$	F_2
Φ_4	S_4	$[2,0,y,1]$	F_3
Φ_3	S_2	$[2,0,y,0]$	F_4
Φ_2	$S_2, ..., S_6$	$[2]$	F_1
Φ_1	S_1	$[1]$	F_2
Φ_0	S_0	$[0]$	

Table 8. The Abstract Table of Program COND-TERM

Index	φ_i	Δ_i
7	$at\_l_3 \wedge y > 0 \wedge x = 0$	$[2,2]$
6	$at\_l_1 \wedge y > 0 \wedge x = 0$	$[2,1]$
5	$at\_l_1 \wedge y > 0 \wedge x = 1$	$[2,0,y,2]$
4	$at\_l_2 \wedge y > 0 \wedge x \in \{0,1\}$	$[2,0,y,1]$
3	$at\_l_3 \wedge y > 0 \wedge x = 1$	$[2,0,y,0]$
2	$at\_l_{1..3} \wedge y \geqslant at\_l_{1,2,3} \wedge x \in \{0,1\}$	$[2]$
1	$at\_l_1 \wedge y = 0 \wedge x = 1$	$[1]$

Table 9. The Concrete Table of Program COND-TERM

4.3 Example 3: UP-DOWN

Fig. 9 shows the program UP-DOWN. This program can be handled by RANK-JUST-GRAPH [2]. Here we illustrate how it is handled by C-GRAPH, in order to get an impression of the similarity and differences of these two approaches.The abstract program is shown in Fig.10.

The response property is $(\pi_1 = 0 \wedge \pi_2 = 0) \Rightarrow \Diamond(\pi_1 = 4)$, where π_1 and π_2 are the location counters for P_1 and P_2. They are denoted by l_i and m_i, respectively.

x,y: natural init x =0, y=1

l_0: **while** x =0 **do**	
l_1: y := y+1	
l_2: **while** $y > 0$ **do**	m_0: x:=1
l_3: y := y-1	m_1:
l_4:	

Fig. 9. Program UP-DOWN

X,Y: natural init X =0, Y=1

l_0: **while** X =0 **do**	
l_1: Y := 1	
l_2: **while** $Y = 1$ **do**	m_0: X:=1
l_3: Y := max(Y-1,0)	m_1:
l_4:	

Fig. 10. Abstract Program UP-DOWN

We also use the notation at_l_i to denote $\pi_1 = i$, and at_m_j to denote $\pi_2 = j$. The justice requirements of the program include $F_0 : \langle 1, \neg at\_l_0 \rangle$, $F_1 : \langle 1, \neg at\_l_1 \rangle$, $F_2 : \langle 1, \neg at\_l_2 \rangle$, $F_3 : \langle 1, \neg at\_l_3 \rangle$, $F_4 : \langle 1, \neg at\_m_1 \rangle$. Employing the predicate base

$\mathcal{P} : x > 0, y > 0$, we obtain the abstraction

$$\alpha : \Pi_1 = \pi_1, \Pi_2 = \pi_2, X = (x > 0), Y = (y > 0)$$

The abstract property is $\Psi_\alpha : (\Pi_1 = 0 \wedge \Pi_2 = 0) \Rightarrow \Diamond(\Pi_1 = 4)$.

The pending graph is shown in Fig.11 and the abstract diagram is shown in Fig.12.

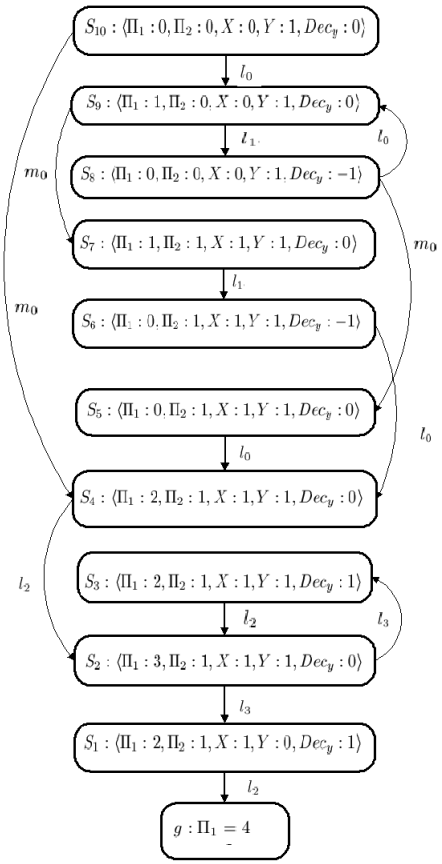

Fig. 11. The Pending Graph

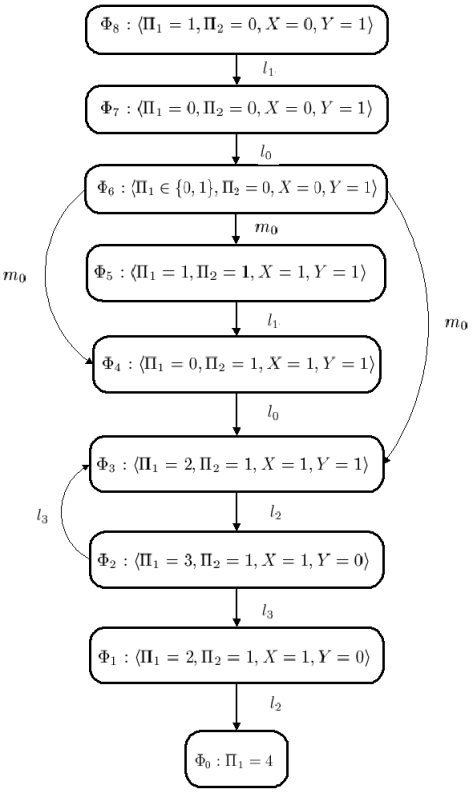

Fig. 12. The Abstract Graph

In the first level of computation, the MSCC decomposition yields the following sorted list: g ,S_1,{S_2,S_3},S_4,S_5,S_6,S_7,{ S_8,S_9},S_{10}. Consequently, we assign to nodes g, S_1,...,S_7 the sequence of ranks: 0 ,1 ,2 ,2 ,3 ,4 ,5 ,6, 7, 7, 8.

In the second level of computation, we consider the MSCCs {S_2,S_3} and {S_8,S_9}. Table 10 shows the final values of R_i and H_i.

$Index\ i$	S_i	R_i	H_i
10	S_{10}	[8]	$[F_0, F_5]$
9	S_9	[7, 1]	$[F_5, F_1]$
8	S_8	[7, 0]	$[F_5, F_0]$
7	S_7	[6]	$[F_1]$
6	S_6	[5]	$[F_0]$
5	S_5	[4]	$[F_0]$
4	S_4	[3]	$[F_2]$
3	S_3	$[2, y, 1]$	$[F_2, F_D]$
2	S_2	$[2, y, 0)]$	$[F_3, F_D]$
1	S_1	[1]	$[F_2]$
0	S_0	[0]	

Table 10. The Rank Table of Program UP-DOWN

According to R_i and H_i for $i = 1, ..., 8$, we construct abstract states with their associated ranks as shown in Table 11. The the abstract graph is shown in Fig.12.

$Abstract\ Node$	$Nodes$	$Rank$	$Compassion\ Req.$
Φ_8	S_9	[7]	F_1
Φ_7	S_8, S_{10}	[6]	F_0
Φ_6	S_9, S_8, S_{10}	[5]	F_5
Φ_5	S_7	[4]	F_1
Φ_4	S_5, S_6	[3]	F_0
Φ_3	S_3, S_4	$[2, y, 1]$	F_2
Φ_2	S_2	$[2, y, 0]$	F_3
Φ_1	S_1	[1]	F_2
Φ_0	S_0	[0]	

Table 11. The Abstract Table of Program UP-DOWN

At the last step, we compute the concrete helpful assertions $\varphi_1, \ldots, \varphi_8$ by concretization of the abstract assertions $\Phi_1, \ldots, \Phi_8$. The concrete helpful assertions are shown in the table in Table 12.

The validity of the premises of rule C-RESPONSE for example 3 has been verified with the constructed auxiliary constructs $\varphi_1, ..., \varphi_8$ and $\Delta_1, ..., \Delta_8$. The reader is referred to Appendix A.3 for the details.

4.4 Discussion

On different compassion requirements Suppose that in the first example, we replace the original compassion requirement $\langle at\_l_2[1] \wedge y, at\_l_3[1] \rangle$ by $\langle at\_l_2[1] \wedge$

$Index$	φ_i	Δ_i
8	$at\_l_1 \wedge at\_m_0 \wedge x = 0 \wedge y > 0$	[7]
7	$at\_l_0 \wedge at\_m_0 \wedge x = 0 \wedge y > 0$	[6]
6	$at\_l_{0,1} \wedge at\_m_0 \wedge x = 0 \wedge y > 0$	[5]
5	$at\_l_1 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	[4]
4	$at\_l_0 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	[3]
3	$at\_l_2 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	$[2, y, 1]$
2	$at\_l_3 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	$[2, y, 0]$
1	$at\_l_2 at\_m_1 \wedge x > 0 \wedge y = 0$	[1]

Table 12. The Concrete Table of Program UP-DOWN

$y \wedge at\_l_2[j], at\_l_3[1]\rangle$. According to the approach, we can also obtain a set of auxiliary assertions and ranks, however these are too weak for the deductive proof of the liveness property. Since the modified system does not satisfy the liveness property, because the fairness requirement is not strong enough, the result is as expected. Details are in Appendix A.1 (Example 1a).

On the construction of pending graph We may construct a pending graph by grouping states together to obtain a smaller graph. However this may cause problems, if it is not done carefully. Suppose that in the second example, we construct the pending graph as shown in Fig. 13, which is more compact than that used in the second example. The problem with this pending graph is explained

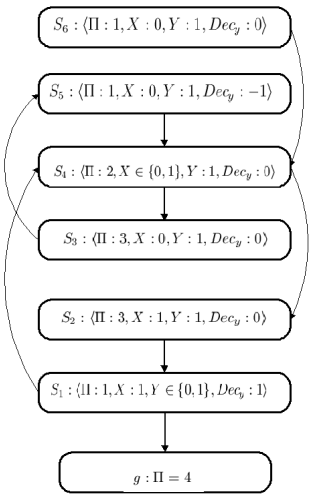

Fig. 13. Another Pending Graph of COND-TERM

as follows. Since the MSCC $\{S_4, S_2, S_1\}$ violates $F_D : \langle Dec_y > 0, Dec_y < 0\rangle$, we have to remove some edges of the MSCC in order to continue the further analysis. If we remove the edge $(S_2 \rightarrow S_1)$, then the rank of S_2 is the minimum. However it does not necessarily make progress to the goal state, because the value of variable X in S_1 includes 1 and 0. Therefore we have to separate the state $\langle \Pi : 1, X \in \{0,1\}, Y : 0, Dec_y : 1\rangle$ into two different states: $\langle \Pi : 1, X : 0, Y : 0, Dec_y : 1\rangle$ and $\langle \Pi : 1, X : 1, Y : 0, Dec_y : 1\rangle$. This modification makes the pending graph of COND-TERM that we used in Section 4.2.

5 Concluding Remarks

For proving the response property in systems with fairness based on the rule presented in [13], we need auxiliary constructs. We have presented a method for extracting such constructs. The method extends that presented in [2] which aimed at proving the response property in systems with justice. When the system is restricted to only allow justice requirements, the auxiliary constructs we obtained may be different from that obtained by using the method presented in [2].

References

1. Arons T, Pnueli A, Ruah S, Xu Y, Zuck L D. Parameterized Verification with Automatically Computed Inductive Assertions. *Proc. Computer Aided Verification*, volume 2102 of LNCS, pp.221–234. Springer,2001.
2. Balaban I, Pnueli A, Zuck L D. Modular ranking abstraction. *International Journal of Foundations of Computer Science*, 2007, 18(1): 5–44.
3. Balaban I, Pnueli A, Zuck L D. Ranking abstraction as companion to predicate abstraction. In *Proc. FORTE*, Taipei, Taiwan, October 2–5, 2005, pp.1-12.
4. Ball T, Majumdar R, Millstein T, Rajamani S. Automatic predicate abstraction if C programs. *Proc. PLDI*, volume 36 of ACM SIGPLAN Notices, pp.203–213. ACM Press, 2001.
5. Das S. Predicate Abstraction. Stanford University, 2003.
6. Fang Y, Piterman N, Pnueli A, Zuck L D. Liveness by invisible ranking. *Proc. VMCAI2004*, LNCS 2937, pp.223–238, 2004.
7. Fang Y, Piterman N, Pnueli A, Zuck L D. Liveness with incomprehensible ranking. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems*, Barcelona, Spain, Mar. 29–Apr. 2, 2004, pp.482-496.
8. Graf S, Saidi H. Construction of abstract state graph with PVS. *Proc. Computer Aided Verification*, volume 1254 of LNCS, pp.72–83. Springer,1997.
9. Kesten Y, Pnueli A. Verification by augmented finitary abstraction. *Information and Computation*, 2000, 163(1): 203-243.
10. Kesten Y, Pnueli A, Vardi M. Verification by augmented abstraction: The automata theoretic view. *J.Comp¿systems Sci*, 2001, 62:668–690.
11. Manna Z, Pnueli A. Completing the temporal picture. *TCS*, 1991, 83(1):91–130.
12. Manna z, Pnueli A. Temporal verification diagrams. *Theoretical Aspects of Computer Software*, Volume 789:726-765,1994.
13. Pnueli A, Sa'ar Y. All you need is compassion. In *Proc. VMCAI*, San Francisco, USA, January 7–9, 2008, pp.33–247.

A Appendix

This appendix contains the steps of checking whether the premisses of the rule C-RESPONSE is valid by utilizing the auxiliary constructs produced by C-GRAPH.

A.1 Example 1: MUX-SEM

The goal represented by $p \rightarrow \Diamond q$, the compassion requirements represented by (p_i, q_i), and the auxiliary constructs for example 1 are listed as follows.

p: $at\_l_2[1]$	q: $at\_l_3[1]$		
$\varphi_1 : at\_l_2[1]$	$p_1 : at\_l_2[1] \wedge y = 1$	$q_1 : at\_l_3[1]$	Δ_1: [0]
$\varphi_2 : at\_l_2[1] \wedge at\_l_4[j] \wedge y = 0$	p_2: 1	$q_2 : \neg at\_l_4[j]$	Δ_2: [1]
$\varphi_3 : at\_l_2[1] \wedge at\_l_3[j] \wedge y = 0$	p_3: 1	$q_3 : \neg at\_l_3[j]$	Δ_3: [2]

We prove that the premisses of the rule C-RESPONSE hold when using the above auxiliary constructs as follows. In the proof, S_i which represents a subset of the concrete states is the corresponding state in Fig. 4.

$$\begin{array}{lll} \text{R1 p} & \Rightarrow \text{q} \vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j) \\ S_3, S_2, S_1 & p_1 \wedge \varphi_1 : S_1 \\ & p_2 \wedge \varphi_2 : S_2 \\ & p_3 \wedge \varphi_3 : S_3 \end{array}$$

This means that the initial states S_3, S_2, S_1 which satisfy $p : at\_l_2[1]$ can be covered by the set of states specified on the right hand side. This proof uses information provided by the pend graph. Formally, the proof of $p \Rightarrow q \vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j)$ may be carried out (without using the pend graph) as follows.

$$\begin{array}{|ll|} \hline p & \Rightarrow q \vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j) \\ \text{iff} & \\ at\_l_2[1] & \Rightarrow at\_l_3[1] \vee (at\_l_2[1] \wedge y = 1 \wedge at\_l_2[1]) \\ & \quad \vee \exists j \neq 1.(at\_l_2[1] \wedge at\_l_4[j] \wedge y = 0) \\ & \quad \vee \exists j \neq 1.(at\_l_2[1] \wedge at\_l_3[j] \wedge y = 0) \\ \text{iff} & \\ at\_l_2[1] & \Rightarrow at\_l_3[1] \vee (y = 1) \\ & \quad \vee \exists j \neq 1.(at\_l_4[j] \wedge y = 0) \\ & \quad \vee \exists j \neq 1.(at\_l_3[j] \wedge y = 0) \\ \text{iff} & \\ at\_l_2[1] & \Rightarrow (y = 1) \\ & \quad \vee \exists j \neq 1.(at\_l_4[j] \wedge y = 0) \\ & \quad \vee \exists j \neq 1.(at\_l_3[j] \wedge y = 0) \\ \text{only if} & \\ y \neq 1 & \Rightarrow \exists j.(at\_l_4[j]) \vee \exists j.(at\_l_3[j]) \\ \hline \end{array}$$

The last implication holds, because it is an invariant of the program. In the following, for simplicity, we use the pend graph as the basis of our proofs of the validity of R2-R4, and omit this kind of details.

i=1

$$\begin{array}{lll} \text{1R2}\ p_1 \wedge \varphi_1 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) & & \\ \quad S_3, S_1 & \text{g} & p_1 \wedge \varphi_1 : S_1 \\ & & p_3 \wedge \varphi_3 : S_3 \\ \text{1R3}\ \varphi_1 \wedge \rho \quad \Rightarrow q' \vee (\varphi'_1 \wedge \Delta_1 = \Delta'_1) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_1 \succ \Delta'_j) & & \\ \quad S_3, S_2, S_1 \quad \text{g} & \varphi_1 : S_1, S_2, S_3 & \\ \text{1R4}\ \varphi_1 \quad \Rightarrow \neg q_1 & & \\ \quad S_3, S_2, S_1 \quad S_3, S_2, S_1 & & \end{array}$$

i=2

$$\begin{array}{lll} \text{2R2}\ p_2 \wedge \varphi_2 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) & & \\ \quad S_1, S_2 & \text{g} & p_1 \wedge \varphi_1 : S_1 \\ & & p_2 \wedge \varphi_2 : S_2 \\ \text{2R3}\ \varphi_2 \wedge \rho \Rightarrow q' \vee (\varphi'_2 \wedge \Delta_2 = \Delta'_2) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_2 \succ \Delta'_j) & & \\ \quad S_1, S_2 \quad \text{g} & \varphi_2 : S_2 & p_1 \wedge \varphi_1 : S_1 \\ \text{2R4}\ \varphi_2 \Rightarrow \neg q_2 & & \\ \quad S_2 \quad S_2 & & \end{array}$$

i=3

$$\begin{array}{lll} \text{3R2}\ p_3 \wedge \varphi_3 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) & & \\ \quad S_2, S_3 & & p_2 \wedge \varphi_2 : S_2 \\ & & p_3 \wedge \varphi_3 : S_3 \\ \text{3R3}\ \varphi_3 \wedge \rho \Rightarrow q' \vee (\varphi'_3 \wedge \Delta_3 = \Delta'_3) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_3 \succ \Delta'_j) & & \\ \quad S_2, S_3 & \varphi_3 : S_3 & p_2 \wedge \varphi_2 : S_2 \\ \text{3R4}\ \varphi_3 \Rightarrow \neg q_3 & & \\ \quad S_3 \quad S_3 & & \end{array}$$

This means that the transitions from φ_i and $p_i \wedge \varphi_i$ satisfy $R_2 - R_4$. Therefore we can reach the goal state: $at\_l_3[1]$ with these transitions, and the liveness property holds.

For illustration, the transitions of the program with 3 processes are shown in Fig. 14- Fig. 17 . The labels on the transitions indicate how states are changed according to the premisses of the rule, such that if we start from some state satisfying the initial assertion p, then we have transitions satisfy the premisses of the rule, and we can reach the goal state with these transitions.

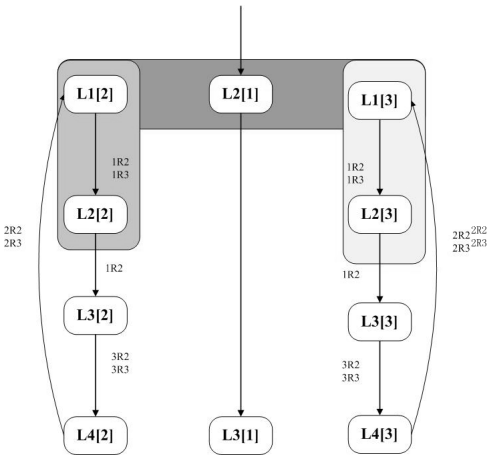

Fig. 14. S_1 in 3-Processes MUX

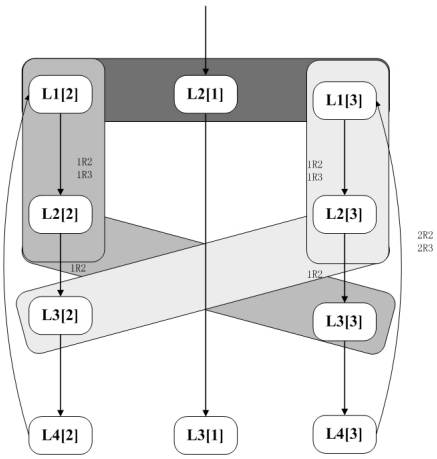

Fig. 15. S_2 in 3-Processes MUX

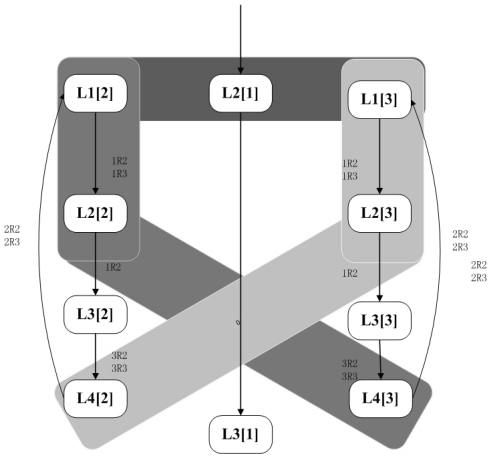

Fig. 16. S_3 in 3-Processes MUX

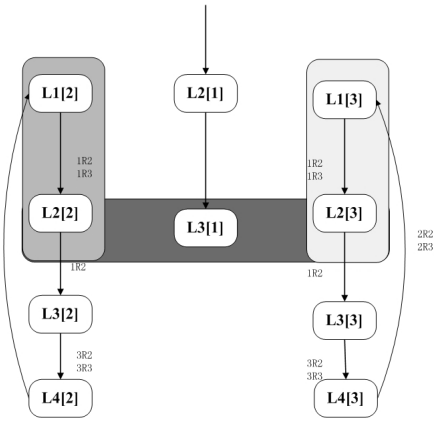

Fig. 17. S_0 in 3-Processes MUX

A.2 Example 2: COND-TERM

The goal, the compassion requirements and the auxiliary constructs for example 2 are listed as follows.

p: at_l_1	q: at_l_3		
$\varphi_1 : at\_l_1 \wedge y = 0$	p_1: 1	$q_1 : \neg at\_l_1$	Δ_1: [1]
$\varphi_2 : at\_l_{1..3} \wedge y \geqslant at\_l_{1,2,3} \wedge x \in \{0,1\}$	$p_2 : at\_l_3 \wedge x = 0$	q_2: 0	Δ_2: [2]
$\varphi_3 : at\_l_3 \wedge y > 0 \wedge x = 1$	p_3: 1	$q_3 : \neg at\_l_3$	Δ_3: [2,0,y,0]
$\varphi_4 : at\_l_2 \wedge y > 0 \wedge x \in \{0,1\}$	p_4: 1	$q_4 : \neg at\_l_2$	Δ_4: [2,0,y,1]
$\varphi_5 : at\_l_1 \wedge y > 0 \wedge x = 1$	p_5: 1	$q_5 : \neg at\_l_1$	Δ_5: [2,0,y,2]
$\varphi_6 : at\_l_1 \wedge y > 0 \wedge x = 0$	p_6: 1	$q_6 : \neg at\_l_1$	Δ_6: [2,1]
$\varphi_7 : at\_l_3 \wedge y > 0 \wedge x = 0$	p_7: 1	$q_7 : \neg at\_l_3$	Δ_7: [2,2]

We prove that the premisses of the rule C-RESPONSE hold when using the above auxiliary constructs as follows.

$$\begin{array}{lll} \text{R1 p} & \Rightarrow \text{q} \vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j) \\ S_7, S_6, S_5, S_1 & p_1 \wedge \varphi_1 : S_1 \\ & p_5 \wedge \varphi_5 : S_5 \\ & p_6 \wedge \varphi_6 : S_6, S_7 \end{array}$$

This means that the initial states S_7, S_6, S_5, S_1 which satisfy $p : at\_l_1$ can be covered by the set of states specified on the right hand side.

i=1

$$\begin{array}{lll} \text{1R2} & p_1 \wedge \varphi_1 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) \\ & \text{g} \qquad\qquad \text{g} \\ \text{1R3} & \varphi_1 \wedge \rho \Rightarrow q' \vee (\varphi_1 \wedge \Delta_1 = \Delta'_1) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_1 \succ \Delta'_j) \\ & \text{g} \qquad\quad \text{g} \\ \text{1R4} & \varphi_1 \Rightarrow \neg q_1 \\ & S_1 \quad S_1 \end{array}$$

i=2

$$\begin{array}{lll} \text{2R2} & p_2 \wedge \varphi_2 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) \\ & S_6 \qquad\qquad p_6 \wedge \varphi_6 : S_6 \\ \text{2R3} & \varphi_2 \wedge \rho \qquad\qquad \Rightarrow q' \vee (\varphi'_2 \wedge \Delta_2 = \Delta'_2) \vee \quad \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_2 \succ \Delta'_j) \\ & S_6, S_5, S_4, S_3, S_2, S_1 \quad \text{g} \quad \varphi_2 : S_6, S_5, S_4, S_3, S_2 \; p_1 \wedge \varphi_1 : S_1 \\ \text{2R4} & \varphi_2 \qquad\qquad \Rightarrow \neg q_2 \\ & S_6, S_5, S_4, S_3, S_2 \quad S_6, S_5, S_4, S_3, S_2 \end{array}$$

i=3

$$\begin{array}{lll} \text{3R2} & p_3 \wedge \varphi_3 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) \\ & S_5, S_1 \qquad\quad \text{g} \quad p_1 \wedge \varphi_1 : S_1 \\ & \qquad\qquad\quad\;\; p_5 \wedge \varphi_5 : S_5 \end{array}$$

3R3 $\varphi_3 \wedge \rho \Rightarrow q' \vee (\varphi'_3 \wedge \Delta_3 = \Delta'_3) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_3 \succ \Delta'_j)$
S_5, S_1 g $p_1 \wedge \varphi_1 : S_1$
$p_5 \wedge \varphi_5 : S_5$

In $3R_3$, we have $\Delta_3 \succ_{S_1} \Delta_1$ and $\Delta_3 \succ_{S_5} \Delta_5$. The validity of the former is trivial. The validity of latter depends on Dec_y. In S_5, we have $Dec_y = 1$ meaning $y > y'$. The values of Δ_3 and Δ_5 depend on the values of y. In this case, we have $\Delta_3 \succ_{S_5} \Delta_5$ and the condition $\bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_3 \succ \Delta'_j)$ holds in $3R_3$.

3R4 $\varphi_3 \Rightarrow \neg q_3$
S_2 S_2

i=4

4R2 $p_4 \wedge \varphi_4 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j)$
S_3, S_2 g $p_3 \wedge \varphi_3 : S_2$
$p_1 \wedge \varphi_1 : S_3$

4R3 $\varphi_4 \wedge \rho \Rightarrow q' \vee (\varphi'_4 \wedge \Delta_4 = \Delta'_4) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_4 \succ \Delta'_j)$
S_3, S_2 g $p_3 \wedge \varphi_3 : S_2$
$p_2 \wedge \varphi_2 : S_3$

4R4 $\varphi_4 \Rightarrow \neg q_4$
S_4 S_4

i=5

5R2 $p_5 \wedge \varphi_5 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j)$
S_4 g $p_4 \wedge \varphi_4 : S_4$

5R3 $\varphi_5 \wedge \rho \Rightarrow q' \vee (\varphi'_5 \wedge \Delta_5 = \Delta'_5) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_5 \succ \Delta'_j)$
S_4 g $p_4 \wedge \varphi_4 : S_4$

5R4 $\varphi_5 \Rightarrow \neg q_5$
S_5 S_5

i=6

6R2 $p_7 \wedge \varphi_6 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j)$
S_4 $p_4 \wedge \varphi_4 : S_4$

6R3 $\varphi_6 \wedge \rho \Rightarrow q' \vee (\varphi'_6 \wedge \Delta_6 = \Delta'_6) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_6 \succ \Delta'_j)$
S_4 $p_4 \wedge \varphi_4 : S_4$

6R4 $\varphi_6 \Rightarrow \neg q_6$
S_7, S_6 S_7, S_6

i=7

7R2 $p_7 \wedge \varphi_7 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j)$
S_6 $p_6 \wedge \varphi_6 : S_6$

7R3 $\varphi_7 \wedge \rho \Rightarrow q' \vee (\varphi'_7 \wedge \Delta_7 = \Delta'_7) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_7 \succ \Delta'_j)$
S_6 $p_6 \wedge \varphi_6 : S_6$

7R4 $\varphi_7 \Rightarrow \neg q_7$
$S_3 \quad S_3$

This means that every state in the transitions satisfies the premisses of the rule, therefore we can reach the goal state: at_l_3 with these transitions, and the liveness property holds.

A.3 Example 3: UP-DOWN

The goal, the compassion requirements and the auxiliary constructs for example 3 are listed as follows.

p: $at\_l_0 \wedge at\_m_0$	q: at_l_4		
$\varphi_1 : at\_l_2 \wedge at\_m_1 \wedge x > 0 \wedge y = 0$	p_1: 1	$q_1 : \neg at\_l_2$	Δ_1: [1]
$\varphi_2 : at\_l_3 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	$p_2 : 1$	$q_2 : \neg at\_l_3$	Δ_2: [2,y,0]
$\varphi_3 : at\_l_2 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	p_3: 1	$q_3 : \neg at\_l_2$	Δ_3: [2,y,1]
$\varphi_4 : at\_l_0 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	p_4: 1	$q_4 : \neg at\_l_0$	Δ_4: [3]
$\varphi_5 : at\_l_1 \wedge at\_m_1 \wedge x > 0 \wedge y > 0$	p_5: 1	$q_5 : \neg at\_l_1$	Δ_5: [4]
$\varphi_6 : at\_l_{0,1} \wedge at\_m_0 \wedge x = 0 \wedge y > 0$	p_6: 1	$q_6 : \neg at\_m_0$	Δ_6: [5]
$\varphi_7 : at\_l_0 \wedge at\_m_0 \wedge x = 0 \wedge y > 0$	p_7: 1	$q_7 : \neg at\_l_0$	Δ_7: [6]
$\varphi_8 : at\_l_1 \wedge at\_m_0 \wedge x = 0 \wedge y > 0$	p_8: 1	$q_8 : \neg at\_l_1$	Δ_8: [7]

We prove that the premisses of the rule C-RESPONSE hold when using the above auxiliary constructs as follows.

R1 p $\Rightarrow$ q $\vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j)$
$S_{10}, S_8 \quad p_6 \wedge \varphi_6 : S_{10}, S_8$
$p_7 \wedge \varphi_7 : S_{10}, S_8$

This means that the initial states S_{10}, S_8 which satisfy $p : at\_l_0 \wedge at\_m_0$ can be covered by the set of states specified on the right hand side.

i=1

1R2 $p_1 \wedge \varphi_1 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j)$
g g
1R3 $\varphi_1 \wedge \rho \Rightarrow q' \vee (\varphi_1 \wedge \Delta_1 = \Delta'_1) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_1 \succ \Delta'_j)$
g g
1R4 $\varphi_1 \Rightarrow \neg q_1$
$S_1 \quad S_1$

i=2

2R2 $p_2 \wedge \varphi_2 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j)$
$S_3, S_1 \quad p_3 \wedge \varphi_3 : S_3$
$p_1 \wedge \varphi_1 : S_1$

2R3 $\varphi_2 \wedge \rho \Rightarrow q' \vee (\varphi'_2 \wedge \Delta_2 = \Delta'_2) \vee \bigvee_{j=1}^{n} (p'_j \wedge \varphi'_j \wedge \Delta_2 \succ \Delta'_j)$
S_3, S_1 g $p_1 \wedge \varphi_1 : S_1$
$p_3 \wedge \varphi_3 : S_3$

2R4 $\varphi_2 \Rightarrow \neg q_2$
S_3, S_1 S_3, S_1

i=3

3R2 $p_3 \wedge \varphi_3 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n} ((p'_j \wedge \varphi'_j)$
S_2 g $p_2 \wedge \varphi_2 : S_2$

3R3 $\varphi_3 \wedge \rho \Rightarrow q' \vee (\varphi'_3 \wedge \Delta_3 = \Delta'_3) \vee \bigvee_{j=1}^{n} (p'_j \wedge \varphi'_j \wedge \Delta_3 \succ \Delta'_j)$
S_2 g $p_2 \wedge \varphi_2 : S_2$

3R4 $\varphi_3 \Rightarrow \neg q_3$
S_3, S_4 S_3, S_4

i=4

4R2 $p_4 \wedge \varphi_4 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n} ((p'_j \wedge \varphi'_j)$
S_4 g $p_3 \wedge \varphi_3 : S_4$

4R3 $\varphi_4 \wedge \rho \Rightarrow q' \vee (\varphi'_4 \wedge \Delta_4 = \Delta'_4) \vee \bigvee_{j=1}^{n} (p'_j \wedge \varphi'_j \wedge \Delta_4 \succ \Delta'_j)$
S_4 g $p_3 \wedge \varphi_3 : S_4$

4R4 $\varphi_4 \Rightarrow \neg q_4$
$S_5 S_6$ $S_5 S_6$

i=5

5R2 $p_5 \wedge \varphi_5 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n} ((p'_j \wedge \varphi'_j)$
S_6 g $p_4 \wedge \varphi_4 : S_6$

5R3 $\varphi_5 \wedge \rho \Rightarrow q' \vee (\varphi'_5 \wedge \Delta_5 = \Delta'_5) \vee \bigvee_{j=1}^{n} (p'_j \wedge \varphi'_j \wedge \Delta_5 \succ \Delta'_j)$
S_6 g $p_4 \wedge \varphi_4 : S_6$

5R4 $\varphi_5 \Rightarrow \neg q_5$
S_7 S_7

i=6

6R2 $p_7 \wedge \varphi_6 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n} ((p'_j \wedge \varphi'_j)$
S_4, S_7, S_5 $p_3 \wedge \varphi_3 : S_5$
$p_4 \wedge \varphi_4 : S_5$
$p_5 \wedge \varphi_5 : S_7$

6R3 $\varphi_6 \wedge \rho \Rightarrow q' \vee (\varphi'_6 \wedge \Delta_6 = \Delta'_6) \vee \bigvee_{j=1}^{n} (p'_j \wedge \varphi'_j \wedge \Delta_6 \succ \Delta'_j)$
S_4, S_7, S_5 $p_3 \wedge \varphi_3 : S_5$
$p_4 \wedge \varphi_4 : S_5$
$p_5 \wedge \varphi_5 : S_7$

6R4 $\varphi_6 \Rightarrow \neg q_6$
S_8, S_9, S_{10} S_8, S_9, S_{10}

i=7

$$\begin{array}{ll} \text{7R2} & p_7 \wedge \varphi_7 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) \\ & S_9 \qquad\qquad\qquad\quad p_6 \wedge \varphi_6 : S_9 \\ \text{7R3} & \varphi_7 \wedge \rho \Rightarrow q' \vee (\varphi'_7 \wedge \Delta_7 = \Delta'_7) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_7 \succ \Delta'_j) \\ & S_9 \qquad\qquad\qquad\qquad\qquad\quad p_6 \wedge \varphi_6 : S_9 \\ \text{7R4} & \varphi_7 \qquad \Rightarrow \neg q_7 \\ & S_8, S_{10} \qquad S_8, S_{10} \end{array}$$

i=8

$$\begin{array}{ll} \text{8R2} & p_7 \wedge \varphi_7 \wedge \rho \Rightarrow q' \vee \bigvee_{j=1}^{n}((p'_j \wedge \varphi'_j) \\ & S_8 \qquad\qquad\qquad\quad p_7 \wedge \varphi_7 : S_8 \\ \text{8R3} & \varphi_7 \wedge \rho \Rightarrow q' \vee (\varphi'_7 \wedge \Delta_7 = \Delta'_7) \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j \wedge \Delta_7 \succ \Delta'_j) \\ & S_8 \qquad\qquad\qquad\qquad\qquad\quad p_7 \wedge \varphi_7 : S_8 \\ \text{8R4} & \varphi_7 \Rightarrow \neg q_7 \\ & S_9 \qquad S_9 \end{array}$$

This means that every state in the transitions satisfies the premisses of the rule, therefore we can reach the goal state: at_l_4 with these transitions, and the liveness property holds.