Auxiliary Constructs for Proving Liveness in Compassion Discrete Systems*

Teng Long[1,2] and Wenhui Zhang[1]

[1] State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] School of Information Science and Engineering
Graduate University of China Academy of Sciences, Beijing, China
{longteng,zwh}@ios.ac.cn

Abstract. For proving response properties in systems with compassion requirements, a deductive rule is introduced in [1]. In order to use the rule, auxiliary constructs are needed. They include helpful assertions and ranking functions defined on a well-founded domain. The work in [2] computes ranking functions for response properties in systems with justice requirements. This paper presents an approach which extends the work in [2] with compassion requirements. The approach is illustrated on two examples of sequential and concurrent programs.

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. The concept of abstraction helps enhancing the applicability of model checking to infinite systems. Predicate abstraction [3,4,5], has been useful for the verification of safety properties in infinite systems. For the verification of liveness properties, ranking abstraction has been introduced in [6,7,8] recognizing that the usual state abstraction is often inadequate to capture liveness properties. Compassion requirements[1] are introduced into the abstract system so that the ranking 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 [9] is one such method for automatically generating helpful assertions and ranking functions for proving liveness

* Supported by the National Natural Science Foundation of China under Grant Nos. 60721061, 60833001, and the CAS Innovation Program.

[1] A compassion requirement is a pair of assertions requiring that in a computation, if the first assertion is satisfied infinitely often, then the second one must also be satisfied infinitely often.

A. Bouajjani and W.-N. Chin (Eds.): ATVA 2010, LNCS 6252, pp. 276–290, 2010.

properties in systems with justice requirements[2]. The method was then extended to handle a larger class of problems by relaxing restrictions requiring that the helpful assertions and ranking functions only depend on the local states of a process [10]. For 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.

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. 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 [1] 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 two examples of sequential and concurrent programs. 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 [1].

Computational Model. A fair discrete system (FDS) is a quintuple $D=\langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ 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 $s' \in \Sigma$.
- $\mathcal{J}$: 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.
- $\mathcal{C}$: A set of compassion requirements (strong fairness). The compassion requirement $\langle p, q\rangle \in \mathcal{C}$ is a pair of assertions, which guarantee that every computation should include either only finitely many p-states, or infinitely many q-states.

[2] A justice requirement is an assertion requiring that in a computation, this assertion must be satisfied infinitely often.

Computation. A computation of D is an infinite sequence of states $\sigma : s_0, s_1, s_2, \ldots$, 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 $\langle p, q \rangle \in \mathcal{C}$. A *compassion discrete system* (CDS) is an FDS with no justice requirements.

Proof Rule for Response Properties. For verifying response properties under the assumption of compassion (strong fairness) requirements over a CDS (since an FDS is equivalent to a CDS[3], it is sufficient to consider CDS only), the deductive rule RESPONSE which was presented and proved to be sound and complete in [1], was developed (this rule is hereafter referred to as C-RESPONSE for emphasizing that it involves compassion requirements). It is shown as follows.

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

R1	p	$\Rightarrow q \vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j)$
$\forall i \leq n$:		
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	$\Rightarrow \Diamond q$

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 $i \in \{1, \ldots, n\}$. It means that the initial states must be a goal state or in a rank. 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 a φ_i-state moves either directly to a q-state, or to a $(p_j \wedge \varphi_j)$-state with decreasing rank, or stay 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

[3] 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.

violates the compassion requirement. R3, R4 and the well-founded domain of the ranks together guarantee that a sequence of moves starting from a state cannot infinitely often decrease the rank or stay at some states with the same rank indefinitely, therefore it must go to the goal state.

The rule also implicitly requires a match among the number of compassion requirements, the number of assertions, and the number of ranking functions. To be more flexible, we extend the proof rule to allow a compassion requirement to be matched with more than one assertion and ranking function. The modified rule is presented as follows.

Let p, q be assertions.
Let $\mathcal{A} : (W, \succ)$ be a well-founded domain.
Let $\{F_i = \langle p_i, q_i \rangle \mid i \in \{1, ..., n\}\}$ be a set of compassion requirements.
Let $(\{F_i \mid i \in \{1, ..., n\}\}, \{(F_1, k_1), ..., (F_n, k_n)\})$ be a multiset, in which k_i is the number of instances of F_i in the multiset, such that $\Sigma_{i=1}^{n} k_i = m$.
Let $\{\varphi_i \mid i \in \{1, ..., m\}\}$ be a set of assertions.
Let $\{\Delta_i : \Sigma \to W \mid i \in \{1, ..., m\}\}$ be a set of ranking functions.

R1	p	$\Rightarrow q \vee \bigvee_{j=1}^{m}(p_{h(j)} \wedge \varphi_j)$
$\forall i \leq m$:		
R2	$p_{h(i)} \wedge \varphi_i \wedge \rho$	$\Rightarrow q' \vee \bigvee_{j=1}^{m}((p'_{h(j)} \wedge \varphi'_j)$
R3	$\varphi_i \wedge \rho$	$\Rightarrow q' \vee (\varphi'_i \wedge \Delta_i = \Delta'_i) \vee \bigvee_{j=1}^{m}(p'_{h(j)} \wedge \varphi'_j \wedge \Delta_i \succ \Delta'_j)$
R4	φ_i	$\Rightarrow \neg q_{h(i)}$
	p	$\Rightarrow \Diamond q$

In which $h(i) = a$, such that $a \in \{1, \ldots, n\} \wedge 0 < i - \Sigma_{l=1}^{a-1} k_l \leq k_a$ holds.

The correctness follows from the original rule by viewing one compassion requirement as multiple identical compassion requirements.

3 Proving a Response Property

In order to be able to use the proof rule C-RESPONSE for proving a response property $\psi : p \Rightarrow \Diamond q$, 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 CDS 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 [6,7,2] 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 of the pending graph and a set of compassion requirements associated to each of these nodes.
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 compassion requirement F_i to the node. Note that according to the construction, one F_i may correspond to several abstract nodes.

3.1 Ranking Abstraction and Pending Graph

This step is carried out according to the technique of ranking abstraction [6,7,2] 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 kind 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 $D = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ (in which $\mathcal{J}$ 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 D by δ is

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

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 have an abstract program D^α from D constructed by the above process with the abstraction map α, a pending graph 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 auxiliary constructs for the proof rule.

3.2 Compassion Requirements and Initial Ranks

The ranking functions in the abstract program are represented as a mapping $N \rightarrow TUPLES$, 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 Δ_l and H_l be respectively the rank and the list of compassion requirements for $S_l \in N$. For convenience, we write q for q^α, and similarly for other formulas and constructions. Let $\mathcal{F} = \{F_1, \ldots, F_n\}$ be a set of original compassion requirements and $\mathcal{F}_D = \{(dec_\delta > 0, dec_\delta < 0)\}$ be the set of

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 Δ_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:    flag = 0;
10:   for each F ∈ F ∪ F_D do
11:      if not violate(C_i,F) then
12:         continue;
13:      end if
14:      flag = 1 ;
15:      for each S_l of C_i do
16:         append F to H_l;
17:      end for
18:      for each F ∈ F_D do
19:         for each S_l of C_i do
20:            append term(F) to Δ_l;
21:         end for
22:      end for
23:      if C_i is a trivial MSCC then
24:         break;
25:      else
26:         remove_edge(C_i,F); C-RANK(C_i); break;
27:      end if
28:   end for
29:   if flag = 0 then
30:      terminate unsuccessfully;
31:   end if
32: end for
```

dec-requirements (the compassion requirements introduced by the ranking abstraction). The procedure for computing Δ_l and H_l (which are initially empty) is described in Algorithm 1. The way of dealing with MSCCs (maximal strongly connected components) and fairness follows the idea of [12] by Emerson and Lei. The main functions are explained as follows.

decompose(G). The graph G is decomposed into a set of MSCCs, denoted as $C_0, ..., C_k$. They are ordered so 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). The MSCC (may be the trivial one) C_i violates the compassion requirement $F = (p, q)$, if p is satisfied by some node of the MSCC and q is not satisfied by any node of the MSCC.

Algorithm 2. C-GRAPH

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

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

remove_edge(C_i, F). Given an MSCC C_i and a compassion requirement $F = (p, q)$, this procedure modifies the MSCC in such a way that one node satisfying p 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 to satisfy the same 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 $F_i \in \mathcal{F}$ (the original compassion 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.

create_merge_nodes(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).

create_edges(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 Correctness of Auxiliary Constructs

For each compassion requirement (p_i, q_i), several abstract states may be associated. This has been made explicit in the modified C-RESPONSE rule, where we consider the set of original 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.

Ranking Core. For the correctness, we assume that every ranking function in ranking abstraction is chosen to be a variable. Such a set of variables (representing a set of ranking functions) are called ranking core $\mathcal{R}$ [6]. 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 $\mathcal{D}^{\mathcal{R},\alpha}$.

Theorem. *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 that successfully 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 auxiliary constructs.*

The correctness is established by analyzing the different steps in the construction of the auxiliary construct. Firstly, two ranks are compared as follows [2]. Let $\Delta_i = (a_1, \ldots, a_r)$, $\Delta_j = (b_1, \ldots, b_r)$. Let $gt(\Delta_i, \Delta_j)$ be defined by:

$$gt(\Delta_i, \Delta_j) \equiv \bigvee_{k=1}^{r} (a_1 = b'_1) \wedge \cdots \wedge (a_{k-1} = b'_{k-1}) \wedge (a_k \succ b'_k)$$

The formula $gt(\Delta_i, \Delta_j)$ formalizes the condition for $\Delta_i \succ \Delta_j$ in the lexicographic order. We may not be able to decide whether $gt(\Delta_i, \Delta_j)$ is true or false immediately, because a_k, b_k may be functions such as $\delta_k = x$ or $\delta_k = y$. Let $\Delta \succ_E \Delta'$ denote that Δ appear after Δ' according to the lexicographic order with the following conditions: the lexicographic order is augmented by an environment E that specifies whether $\delta_k \succ \delta'_k$ or $\delta_k = \delta'_k$. 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. Let $\Delta > \Delta'$, where Δ, Δ' represent ranks, denote that Δ appear after Δ' according to the lexicographic order in the initial state.

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 Δ_i and Δ_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 $\Delta_i \succ_{S_j} \Delta_j$.
- 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 one 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 theorem. 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 directive 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. We concern deductive rule with strong fairness (compassion) requirements to prove liveness properties of more complex concurrent programs. It depends on compassion requirements to decide the ranks of states in MSCC.

4 Application Examples

We illustrate the application of the approach on two programs:

- COND-TERM, a sequential program with a non-deterministic choice of the values of a variable [1].
  This example is supposed to show the approach applied on a verification problem with ranking abstraction in which some dec-requirement is introduced in the abstraction phase.
- MUX-SEM, a concurrent program for mutual exclusion [13].
  This example is supposed to show the approach applied on a concurrent program.

4.1 Example 1: COND-TERM

The following is the program COND-TERM (conditional termination). 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:

Phase 1 (ranking abstraction and pending graph). The ranking core in this case is chosen to be $\{y\}$ and the ARF (augmenting ranking function) y is associated with the natural numbers as the well-founded set. 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. 1. 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. We have $\mathcal{F}_D = \{F_D\}$ in this example. The pending graph includes two kinds of uncompassionate loops, one violating the given compassion and the other violating the compassion introduced in the abstraction process.

There are 8 states $\{S_0, S_1, ..., S_7\}$ with $S_0 = g$.

Phase 2 (compassion requirements and initial ranks). Let Δ_i and H_i be the rank of S_i and the set of compassion requirements associated to S_i, respectively, initially with $\Delta_i = []$ and $H_i = []$.

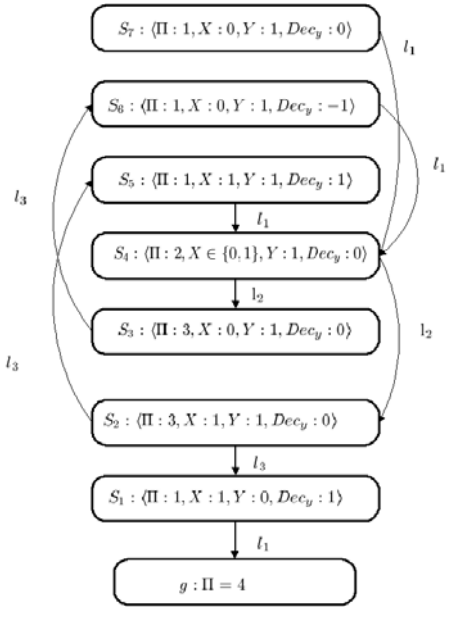

Fig. 1. The Pending Graph

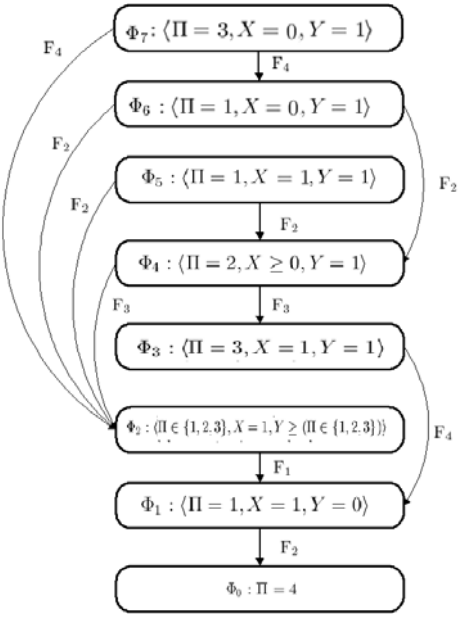

Fig. 2. The Abstract Graph

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 $\Delta_0 = [0], \Delta_1 = [1], \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = \Delta_6 = [2], \Delta_7 = [3]$.

Let C_0, C_1, C_2, C_3 denote the 4 MSCCs. Since C_0 is the set of the goal state, we check C_1, C_2, C_3 against the compassion requirements, and obtain Table 1.

Table 1. The MSCCs C_1, C_2, C_3

Component	Violation
C_1	F_2
C_2	F_1
C_3	F_2

Table 2. The MSCCs C_{21}, C_{22}, C_{23}

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

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

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 $\Delta_3 = [2, 2], \Delta_6 = [2, 1], \Delta_2 = \Delta_4 = \Delta_5 = [2, 0]$.

Let C_{21}, C_{22}, C_{23} denote the 3 MSCCs. By checking the MSCCs against the compassion requirements, we obtain Table 2.

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 $\Delta_2, \Delta_4, \Delta_5$, and obtain $\Delta_2 = \Delta_4 = \Delta_5 = [2, 0, y]$.

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 $2, 1, 0$, and we obtain $\Delta_5 = [2, 1, y, 2], \Delta_4 = [2, 1, y, 1], \Delta_2 = [2, 1, y, 0]$.

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 Δ_i and H_i are as shown in Table 3.

Table 3. The Rank Table of Program COND-TERM

$Index\ i$	S_i	Δ_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]	

Phase 3 (abstract nodes, helpful assertions and ranks). According to H_i, we construct the abstract nodes [4] 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 the dec-variable: merging S_7 and S_5 as another abstract state Φ_6. The abstract nodes with their respective ranks are listed in Table 4 and the abstract graph is shown in Fig. 2.

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. 2. The helpful assertions and the ranks are shown in Table 5.

The validity of the premises of the rule C-RESPONSE for this example may be verified by using the constructed auxiliary constructs $\varphi_1, ..., \varphi_7$ and $\Delta_1, ..., \Delta_7$. The reader is referred to the technical report [5] for the details.

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

Table 4. The Abstract Table of Program COND-TERM

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 5. The Concrete Table of Program COND-TERM

Index	*Helpful Assertion* 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	$at\_l_1 \wedge y = 0 \wedge x = 1$	$[1]$

4.2 Example 2: MUX-SEM

The following is the concurrent 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 $\langle 1, \neg at\_l_4[i] \rangle$ and $\langle 1, \neg at\_l_3[i] \rangle$ for $i \in \{1, ..., n\}$.

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

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 if and only if 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 $\mathcal{F} = \{F_1, F_2, F_3\}$.

Let $-$ denote any value in the range of the respective position of the abstract states. For instance, $(1, -, -, -)$ denotes 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. And (2, 1, 0, 0) denotes that process 1 is at l_2 and there is another process j at l_3 meanwhile $y = 1$. 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 with these four states with $S_0 = g$, and proceed with computing the temporary Δ_i and H_i for each of the states S_1, S_2, S_3 using algorithm 1, and obtain $(\Delta_1, \Delta_2, \Delta_3) = ([1, 2], [1, 0], [1, 1])$. Then we compute the abstract states and their associated ranks using algorithm 2, and obtain three abstract nodes (not counting the node representing the goal state) $\Phi_1 = \{S_1, S_2, S_3\}, \Phi_2 = \{S_2\}, \Phi_3 = \{S_3\}$ with their respective ranks $[1], [1, 0], [1, 1]$ and associated compassion requirement F_1, F_2, F_3.

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. The concrete assertions $\varphi_1, \varphi_2, \varphi_3$ and the ranks $\Delta_1, \Delta_2, \Delta_3$ are shown as follows.

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]

5 Concluding Remarks

For proving a response property in systems with fairness based on the rule presented in [1], we need auxiliary constructs. We have presented a method for extracting such constructs. The method consists of phase 2 and phase 3 described in Section 3, while phase 1 is as same as that of [2]. The method extends that presented in [2] which aimed at proving a response property in systems with justice. The use of the method has been illustrated by examples of concurrent and sequential programs. When the system is restricted to only allowing justice

requirements, the auxiliary constructs we obtained may be different from those obtained by using the method presented in [2]. For illustrating this, we have also tried our method on the example of [2] for proving the response property in a system with justice, the details can be found in the technical report [5].

References

1. Pnueli, A., Sa'ar, Y.: All you need is compassion. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 233–247. Springer, Heidelberg (2008)
2. Balaban, I., Pnueli, A., Zuck, L.D.: Modular ranking abstraction. Int. J. Found. Comput. Sci. 18(1), 5–44 (2007)
3. Graf, S., Saïdi, H.: Construction of abstract state graphs with pvs. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
4. Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of c programs. In: PLDI, pp. 203–213 (2001)
5. Long, T., Zhang, W.: Auxiliary constructs for proving liveness in compassion discrete systems. Technical Report, ISCAS–LCS–09–03, Institute of Sofware, Chinese Academy of Sciences (2009), `http://lcs.ios.ac.cn/~zwh/tr/`
6. Kesten, Y., Pnueli, A.: Verification by augmented finitary abstraction. Inf. Comput. 163(1), 203–243 (2000)
7. Balaban, I., Pnueli, A., Zuck, L.D.: Ranking abstraction as companion to predicate abstraction. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 1–12. Springer, Heidelberg (2005)
8. Kesten, Y., Pnueli, A., Vardi, M.Y.: Verification by augmented abstraction: The automata-theoretic view. J. Comput. Syst. Sci. 62(4), 668–690 (2001)
9. Fang, Y., Piterman, N., Pnueli, A., Zuck, L.D.: Liveness with invisible ranking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 223–238. Springer, Heidelberg (2004)
10. Fang, Y., Piterman, N., Pnueli, A., Zuck, L.D.: Liveness with incomprehensible ranking. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 482–496. Springer, Heidelberg (2004)
11. Manna, Z., Pnueli, A.: Completing the temporal picture. Theor. Comput. Sci. 83(1), 91–130 (1991)
12. Emerson, E.A., Lei, C.L.: Modalities for model checking: Branching time logic strikes back. Sci. Comput. Program. 8(3), 275–306 (1987)
13. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.D.: Parameterized verification with automatically computed inductive assertions. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 221–234. Springer, Heidelberg (2001)