Online appendix for the paper

GEM: a Distributed Goal Evaluation Algorithm for Trust Management

published in Theory and Practice of Logic Programming

DANIEL TRIVELLATO*, NICOLA ZANNONE*, SANDRO ETALLE*,+

*Eindhoven University of Technology, Eindhoven, The Netherlands

+University of Twente, Enschede, The Netherlands

(*e-mail:* {d.trivellato,n.zannone,s.etalle}@tue.nl)

submitted 30 May 2011; revised 13 March 2012; 17 July 2012; accepted 24 September 2012

Appendix A Proofs

As mentioned in Section 3.3 of the paper, we assume that given a request or response message M sent by a principal a to a principal b, one and only one instance of message M is received by b. In other words, we assume no message duplication, and that messages are always received.

We introduce one last definition.

Definition 1
Let S be the set of tables resulting from running GEM on a goal G w.r.t. $P = P_1 \cup \ldots \cup P_n$. Let G_1 be a goal whose table is in S. Let θ be a solution of G_1 using clause $H \leftarrow B_1, \ldots, B_n$. Then, by construction $\exists \theta_0, \ldots, \theta_n$ s.t. $\theta_0 = mgu(G_1, H)$ and θ_j is a solution of $B_j\theta_0 \cdots \theta_{j-1}$ (with $j \in \{1, \ldots, n\}$). The *ranking* of θ is defined inductively as follows:

- $rank(\theta) = 1$ if $n = 0$ (i.e., the clause is a fact),
- $rank(\theta) = 1 + max(rank(\theta_1), \ldots, rank(\theta_n))$ otherwise, where $rank(\theta_j)$ is the ranking of solution θ_j. □

We can now prove the soundness result of GEM.

Proof of Theorem 1. We proceed by contradiction and assume that there exists at least a "wrong" solution $\theta_{i,j}$ in Sol_i, i.e., a solution s.t. there is no corresponding SLD derivation of $P \cup \{G_i\}$ with c.a.s. σ where $G_i\theta_{i,j}$ is a renaming of $G_i\sigma$ (hypothesis).

Let us choose $\theta_{i,j}$ to be a "wrong" solution with minimal ranking $(*)$. Let $G_i = \leftarrow A_i$. Since $\theta_{i,j}$ is a solution of G_i, there exists an evaluation tree of G_i in S created by CREATE TABLE (lines 2-7) with root $\langle id, A_i \leftarrow A_i, new\rangle$, a subnode with clause $c = H \leftarrow B_1, \ldots, B_n$ and substitutions $\theta_0, \ldots, \theta_n$ s.t. $\theta_0 = mgu(A_i, H)$, and for each $l \in \{1, \ldots, n\}$ there exists:

- A node in the evaluation tree of G_i with selected atom $B_l\theta_0 \cdots \theta_{l-1}$ (ACTIVATE NODE, lines 8, 18-19).

- An evaluation tree of $\leftarrow B_l\theta_0 \cdots \theta_{l-1}$ created by CREATE TABLE (lines 2-7) at the location of $B_l\theta_0 \cdots \theta_{l-1}$.
- A solution θ_l of $\leftarrow B_l\theta_0 \cdots \theta_{l-1}$; the answer $B_l\theta_0 \cdots \theta_l$ is sent to the requester of $\leftarrow B_l\theta_0 \cdots \theta_{l-1}$ by GENERATE RESPONSE (lines 12-14 or 20-23) if $\leftarrow B_l\theta_0 \cdots \theta_{l-1}$ is involved in a loop, or by TERMINATE (lines 3-5) otherwise.
- A node with clause $(H \leftarrow B_{l+1}, \ldots, B_n)\theta_0 \cdots \theta_l$ added to the evaluation tree of G_i by PROCESS RESPONSE (lines 20-23).

Then, $\theta_{i,j} = \theta_0 \cdots \theta_n$. If the body of c is empty, then there is a trivial 1-step SLD derivation of $P \cup \{G_i\}$ with c.a.s. σ_i (namely the *mgu* of G_i and c), therefore contradicting the hypothesis. So, let us now assume that $n > 0$; by construction, for each $l \in \{1, \ldots, n\}$, $rank(\theta_l) < rank(\theta_{i,j})$. So, by the minimality argument ($*$), for each $l \in \{1, \ldots, n\}$ there exists an SLD derivation of $P \cup \{\leftarrow B_l\theta_0 \cdots \theta_{l-1}\}$ with c.a.s. σ_l s.t. $B_l\sigma_0 \cdots \sigma_{l-1}\sigma_l = B_l\theta_0 \cdots \theta_{l-1}\theta_l$. But then, by standard logic programming results (given the presence of clause c), there exists a successful SLD derivation of $P \cup \{G_i\}$ with c.a.s. σ s.t. $G_i\sigma = G_i\theta_{i,j}$, contradicting the hypothesis. $\square$

Since GEM employs a "wait" mechanism to determine when the answers of a goal should be sent to the requester, both the completeness and termination properties of the algorithm depend on the correctness of this mechanism. Therefore, before demonstrating that GEM is complete and always terminates, we prove that the "wait" mechanism is correctly implemented, i.e., that the answers of a goal are eventually sent to the requester. This is particularly challenging in the presence of loops.

In the implementation of GEM proposed in Section 3.3 of the paper, the "wait" mechanism for goals involved in a loop consists of loop counters: at each iteration of a loop id, the answers of a goal G are only sent when the counter of loop id in set $ActiveGoals$ is 0 (procedure PROCESS RESPONSE, line 24). Since the counter is set to the number k of subgoals of G which are involved in loop id (GENERATE RESPONSE, lines 7 and 19), at each iteration of loop id the principal evaluating G should thus receive k response messages. In order to prove this, we first show that GEM correctly keeps track of the loops in which the subgoals of G are involved.

Proposition 1
Let $G_1, \ldots, G_m$ be the goals involved in a loop id_1. Let $G_i, G_j \in \{G_1, \ldots, G_m\}$ be two goals s.t. G_j is a subgoal of G_i. Then, the node in the evaluation tree of G_i with selected atom G_j has status *loop(ID)*, where $id_1 \in ID$.

Proof of Proposition 1. Let G_1 be the coordinator of loop id_1. Let $G_1, \ldots, G_k$ be a subset of $G_1, \ldots, G_m$ s.t. for each $i \in \{2, \ldots, k\}$ goal G_i is a subgoal of G_{i-1}, and G_1 is a subgoal of G_k. The node in the evaluation tree of $G_1, \ldots, G_k$ with selected atom $G_2, \ldots, G_k, G_1$ respectively has status *loop(ID)*, where $id_1 \in ID$, because of the following observations:

- The identifiers $id_2, \ldots, id_k$ of the requests for goals $G_1, \ldots, G_k$ and the identifier id_{k+1} of the request for goal G_1 are constructed by procedures CREATE TABLE (lines 5-6) and PROCESS RESPONSE (lines 21-22) in such a way that $id_j \sqsubset id_1$, for each $j \in \{2, \ldots, k+1\}$, and thus the lower request id_{k+1} for G_1 can be identified.

- Upon receiving the lower request id_{k+1}, the principal evaluating G_1 returns a response $\langle id_{k+1}, Ans_{k+1}, S_{k+1}, \{id_1\}\rangle$ to the principal evaluating G_k (procedure PROCESS REQUEST, lines 5-7).
- The status of the node in the evaluation tree of G_k with selected atom G_1 is set to $loop(\{id_1\})$ (PROCESS RESPONSE, lines 12-13).
- A counter for loop id_1 is added to set $ActiveGoals$ in the table of goal G_k (PROCESS RESPONSE, line 14).
- For each $i \in \{2, \ldots, k\}$, the principal evaluating goal G_i sends to the principal evaluating goal G_{i-1} a response of the form $\langle id_i, Ans_i, S_i, ID\rangle$, where ID is the set of all loops in $ActiveGoals$ whose identifier is higher than id_i (GENERATE RESPONSE, lines 18, 21, and 23), and thus $id_1 \in ID$.
- The status of the node in the evaluation tree of goal G_{i-1} with selected atom G_i is set to $loop(ID)$, where $id_1 \in ID$ (PROCESS RESPONSE, lines 12-13). □

Corollary 1
Let G be a goal involved in a loop id. Let k be the number of nodes in the evaluation tree of G with status $loop(ID)$ s.t. $id \in ID$. When a response is sent to the requester of the higher request for G (or lower request, if G is the loop coordinator), the counter of loop id in set $ActiveGoals$ in the table of G is set to k.

At each loop iteration, the counters of the loops in which a goal G is involved are set to the number of subgoals of G involved in those loops by procedure GENERATE RESPONSE, lines 7 and 19. Hence, we now need to show that at each iteration of a loop id the number of response messages with status $loop(id)$ received by the principal evaluating G is equal to the number of subgoals of G involved in loop id, i.e., that counters correctly keep track of the number of response messages received by the principal evaluating G at each iteration of loop id.

Informally, the correctness of counters stems from the fact that at each loop iteration step for a goal G there is *only one* choice of loop identifier to include in the response to the requester of a higher request for G. This is because of the following considerations:

1. Let G be a goal involved in one or more loops. In the loop processing phase, the loop identifier included by the principal evaluating G in the response sent to the requester of a higher request for G is taken from the status of the root node of the evaluation tree of G (procedure GENERATE RESPONSE, lines 20-21).
2. After a response for G is sent by GENERATE RESPONSE (lines 20-23), the status of the root node of the evaluation tree of G is set to *active* (line 24).
3. If G is a non-coordinator goal, then there can be *at most one* loop identifier per time in the status of the root node of its evaluation tree. Therefore, when sending a response for G, procedure GENERATE RESPONSE has only one choice of loop identifier to include in the response status. The reason why a non-coordinator goal can have at most one loop identifier in the status of the root of its evaluation tree is the following. The only point where the status of the root node of a non-coordinator goal G is modified to take into account the loop being processed is on line 18 of procedure PROCESS RESPONSE, and the check on line 17 updates the status only in case it is currently set to *active*. We point out that when the response for a subgoal of

G is processed by PROCESS RESPONSE the status of the root of the evaluation tree of G is always *active*, due to point (2) above and the fact that GEM only processes one goal at a time (which is due to condition on line 24 of PROCESS RESPONSE), and thus no response will be received by the principal evaluating G in the context of a loop unless a response for G was previously sent.

4. if G is the coordinator of a loop id_l, then there can be *at most two* loop identifiers per time in the status of the root node of its evaluation tree: one for loop id_l, and *at most one* for a higher loop id_h. Remember that as loop identifiers we use the identifier of the higher request for the coordinator; hence, in this case id_l is the identifier of the higher request for G. Given the condition on line 20 of GENERATE RESPONSE, only id_h can be included in the status of a response for G sent to the requester of a higher request. In fact, id_h (denoted id_4 in the procedure) is the only identifier in the status of the root node of the evaluation tree of G that is higher than id_l (denoted id_1 in the procedure), i.e., higher than the identifier of the higher request for G. Therefore, when sending a response for G, GENERATE RESPONSE has only one choice of loop identifier to include in the response status, namely id_h.
   Technically, a coordinator can have at most two loop identifiers in the status of the root node of its evaluation tree because of the following. Similarly to non-coordinators, due to condition on line 17 of PROCESS RESPONSE only *one* loop identifier can be added to the root's status on line 18 of PROCESS RESPONSE. This occurs when G receives a response from one of its subgoals in the context of a higher loop id_h. A second loop identifier (the identifier of loop id_l) can be added to the root status on lines 8-9 of GENERATE RESPONSE if the response received by the principal evaluating G in the context of loop id_h leads to new answers of G, which need to be sent to the goals involved in loop id_l. No more than two loops at a time will be processed by the principal evaluating G (i.e., id_l and at most one higher loop id_h) because of the following reasons. Upon receiving a response in the context of a higher loop id_h:

   - a response for G in the context of loop id_h will not be sent to a higher goal until a fixpoint for the loop id_l of which G is the coordinator is reached, during which time the status of the root node of the evaluation tree of G is $loop(\{id_h, id_l\})$, and
   - due to the condition on line 24 of PROCESS RESPONSE no responses for higher goals can be received by the principal evaluating G until a response for G in the context of loop id_h is sent upwards. In fact, the counter of loop id_h in the table of higher goals cannot be 0, because no response for G in the context of loop id_h was sent upwards yet. When a response for G is sent upwards, the status of the root node of its evaluation tree becomes *active* again (see point (2)).

Formally, the correctness of counters is demonstrated by the following Proposition.

Proposition 2
Let G be a goal and $G_1, \ldots, G_k$ be the subgoals of G s.t. $G, G_1, \ldots, G_k$ are involved in a loop id_l. At each iteration of loop id_l, the principal evaluating G receives k response messages, one for each subgoal $G_i \in \{G_1, \ldots, G_k\}$.

Proof of Proposition 2. Let $G, G_1, \ldots, G_k, \ldots, G_m$ be all the goals involved in loop id_l, where $m \geq k$. Let $id, id_1, \ldots, id_m$ be the identifiers of the requests for goals $G, G_1, \ldots, G_m$ respectively. The proof is by induction on the number ℓ of goals $G_j \in \{G_1, \ldots, G_m\}$ s.t. $id_j \sqsubset id$, that is, the number of goals whose request identifier is lower than the identifier id of the request for G.

Base case: $\ell = 1$. Then, also $k = 1$. Let $G_j \in \{G_1, \ldots, G_m\}$ be the only goal s.t. $id_j \sqsubset id$. It is straightforward to see that goal G_j is (a variant of) the coordinator of loop id_l, and id_j denotes the lower request for G_j. When there are no more nodes with status new in the evaluation tree of G_j, i.e., when all the branches of the evaluation tree of G_j have been evaluated, procedure GENERATE RESPONSE is invoked by ACTIVATE NODE (lines 2-3). By procedure GENERATE RESPONSE (lines 12-14), at each loop iteration one and only one response to the request for the coordinator G_j is sent by GEM to the principal evaluating G. Thus, at each iteration of loop id_l the principal evaluating G receives $k = 1$ response messages. Q.e.d.

Inductive case: Now, assume that G has ℓ such goals $G_j \in \{G_1, \ldots, G_m\}$ s.t. $id_j \sqsubset id$, where $\ell > 1$. In this case, each subgoal $G_i \in \{G_1, \ldots, G_k\}$ of G is either the coordinator of loop id_l or a goal with at most $\ell - k$ subgoals $G_p \in \{G_1, \ldots, G_m\}$ s.t. $id_p \sqsubset id_i$. If G_i is the coordinator of loop id_l, by the same reasoning done in the base case, one and only one response to the request for G_i is sent by GEM to the principal evaluating G at each loop iteration.
On the other hand, if G_i is not the loop coordinator, there exist at most $\ell - k$ goals $G_p \in \{G_1, \ldots, G_m\}$ s.t. $id_p \sqsubset id_i$. Let t be the number of subgoals of G_i involved in loop id_l. Since $\ell - k \leq \ell - 1$, by the inductive hypothesis ($*$) the principal evaluating G_i receives t response messages at each iteration of loop id_l. By Corollary 1, at each loop iteration the counter of loop id_l in the table of goal G_i is set to t, and is decreased by 1 every time a response to the requests for its subgoals involved in the loop is received (procedure PROCESS RESPONSE, lines 15-16). Therefore, after t response messages, the counter of loop id_l in the table of G_i is 0, and procedure PROCESS RESPONSE (lines 24-25) resumes the evaluation of goal G_i. When there are no more nodes with status new in the evaluation tree of G_i, procedure ACTIVATE NODE (lines 2-3) invokes GENERATE RESPONSE. By procedure GENERATE RESPONSE (lines 20-21), one and only one response to the request for G_i is sent by GEM to the principal evaluating G at each iteration of loop id_l. Therefore, at each iteration of loop id_l the principal evaluating goal G receives k response messages. □

Finally, we show that procedure TERMINATE is eventually invoked for any goal in a computation.

Proposition 3
Let G_1 be a goal. Procedure TERMINATE is eventually called for G_1.

Proof of Proposition 3. The proof is divided into two parts. First, we show that TERMINATE is eventually called for a goal G_1 that is not involved in a loop. Then, we show that it is always invoked also if G_1 is involved in one or more loops.
The first part of the proof is straightforward, and is given by the fact that the number of answers of goal G_1 is finite. This is because of the following observations:

1. The global policy P is finite, and the terms in P that are not variables are constants defined in P; thus, the Herbrand model of P is finite.
2. Let $P_{G_1} \in P$ be the policy where goal G_1 is defined. The answers of G_1 are computed by GEM through the clauses in P_{G_1} applicable to G_1 (procedure CREATE TABLE, lines 3-7). Each clause can be either a fact or have the form $H \leftarrow B_1, \ldots, B_m$, such that $B_1, \ldots, B_m$ are defined in a policy in P. By (1), both the number of facts in P_{G_1} and the number of answers of subgoals $B_1, \ldots, B_n$ are finite.

Thus, the number of answers of goal G_1 is finite. When all the answers of G_1 have been computed and all the nodes in the partial tree of G_1 have been evaluated, procedure ACTIVATE NODE (lines 2-3) invokes GENERATE RESPONSE, which in turn (lines 2-3) invokes TERMINATE.
Consider now the case in which G_1 is part of an SCC consisting of loops $id_1, \ldots, id_k$, s.t. $id_k \sqsubset \ldots \sqsubset id_1$. Let $G_1, \ldots, G_m$ be all the goals involved in loops $id_1, \ldots, id_k$ (where $m \geq k$), and goal $G_{c_i} \in \{G_1, \ldots, G_m\}$ be the coordinator of loop $id_i \in \{id_1, \ldots, id_k\}$. Because the number of answers of each goal $G_1, \ldots, G_m$ is finite, we have that:

- At each iteration of loop id_i, if new answers of the loop coordinator G_{c_i} are derived, they are sent to the requesters of the lower requests for G_{c_i}, starting a new iteration of loop id_i (procedure GENERATE RESPONSE, lines 6-14). On the contrary, if no answer of G_{c_i} is computed, the answers of G_{c_i} are sent to the requester of the higher request for G_{c_i} (GENERATE RESPONSE, lines 20-23). The loops higher than id_i in the SCC are then processed.
- At each iteration of loop id_1, if new answers of the leader G_{c_1} are derived, they are sent to the requesters of the lower requests for G_{c_1}, starting a new iteration of loop id_1 (procedure GENERATE RESPONSE, lines 6-14). Notice that this might cause a fixpoint for the loops lower than id_1 in the SCC to be recomputed. On the contrary, if no answer of G_{c_1} is computed, the answers of G_{c_1} are sent to the requester of the higher request for G_{c_1}, and a response with status $disposed$ is sent to the requesters of the lower requests for G_{c_1} (GENERATE RESPONSE, lines 15-16 and TERMINATE, lines 3-5).
- For each goal $G_j \in \{G_1, \ldots, G_m\}$, all the nodes in the evaluation tree of G_j are disposed (PROCESS RESPONSE, lines 5-8); then, procedure ACTIVATE NODE is invoked, which immediately invokes GENERATE RESPONSE (lines 2-3).
- The principal evaluating goal G_j sends a response with status $disposed$ to the requester of the higher request for G_j. If G_j is a loop coordinator, the principal evaluating G_j also sends a response with status $disposed$ to the requesters of the lower requests for G_j (GENERATE RESPONSE, lines 2-3 and TERMINATE, lines 3-5).

Therefore, procedure TERMINATE is always invoked for goal G_1. □

Proposition 3 implies that the table of a goal involved in a computation is always disposed. In fact, the disposal of the table of a goal is carried out by procedure TERMINATE (lines 2, 6, and 7). Consider, for instance, the following variation of the global policy introduced in Section 3.1 of the paper, where the research insitute ri refers to goal $memberOfAlpha(c1,X)$ instead of $memberOfAlpha(c2,X)$:

$\mathsf{memberOfAlpha}(c1{,}X) \leftarrow \mathsf{memberOfAlpha}(c2{,}X).$
$\mathsf{memberOfAlpha}(c2{,}X) \leftarrow \mathsf{memberOfAlpha}(ri{,}X).$
$\mathsf{memberOfAlpha}(ri{,}X) \leftarrow \mathsf{memberOfAlpha}(c1{,}X).$

First of all, let us recall that the termination of the evaluation of the goals involved in a loop is commanded by the leader of the SCC (goal $memberOfAlpha(c1{,}X)$ in the example policy). When no new answer of the leader is computed by $c1$ during a loop iteration, procedure TERMINATE is invoked (lines 15-16 of GENERATE RESPONSE), which disposes the table of the goal and sends a response with status *disposed* both to the requesters of the higher and lower requests for $memberOfAlpha(c1{,}X)$ (lines 3-5). When ri receives the response, it disposes all the nodes in the evaluation tree of $memberOfAlpha(ri{,}X)$ involved in a loop (lines 5-8 of PROCESS RESPONSE), which in this case corresponds to disposing all the non-root nodes. At this point, the status of the root node of the evaluation tree of $memberOfAlpha(ri{,}X)$ is *active* (see point 2 of the discussion preceding Proposition 2). Therefore, the condition on line 24 of PROCESS RESPONSE is satisfied, and procedure ACTIVATE NODE is invoked for $memberOfAlpha(c1{,}X)$. Since all the non-root nodes in the evaluation tree of $memberOfAlpha(c1{,}X)$ have status *disposed*, GENERATE RESPONSE is invoked (lines 2-3 of ACTIVATE NODE), which in turn (lines 2-3) invokes procedure TERMINATE. TERMINATE disposes the table of goal $memberOfAlpha(ri{,}X)$ and sends a response with status *disposed* to $c2$. Similarly to $memberOfAlpha(ri{,}X)$, PROCESS RESPONSE disposes all the nodes in the evaluation tree of goal $memberOfAlpha(c2{,}X)$, and a response with status *disposed* is sent by $c2$ to $c1$ by procedure TERMINATE. Since the root of the evaluation tree of $memberOfAlpha(c1{,}X)$ had already been disposed, in this case the response message is ignored by $c1$ (line 4 of PROCESS RESPONSE).

Next, we prove the completeness and termination results.

Proof of Theorem 2. We proceed by contradiction, and assume that S is missing a solution of G_1. That is, there exists a successful SLD derivation of $P \cup \{G_1\}$ with c.a.s. θ and there is no solution σ of G_1 generated by the algorithm s.t. $G_1\theta = G_1\sigma$ (hypothesis).
This implies that there exist a (maximal) set of goals $G_1, \ldots, G_k$ in S s.t. for each $i \in \{1, \ldots, k\}$ there is a non-empty maximal set of substitutions $\{\theta_{i,1}, \ldots, \theta_{i,m_i}\}$ s.t.:

(a) G_i is a goal in S.
(b) $\theta_{i,1}, \ldots, \theta_{i,m_i}$ are correct solutions of G_i according to SLD resolution: for each $\theta_{i,j}$ there exists a successful SLD derivation of $P \cup \{G_i\}$ with c.a.s. $\theta_{i,j}$ (up to renaming).
(c) The algorithm does not generate the answers $G_i\theta_{i,1}, \ldots, G_i\theta_{i,m_i}$ (up to renaming).

The set $G_1, \ldots, G_k$ is not empty as it contains at least G_1 (the finiteness of the construction is demonstrated in the proof of Proposition 3).

For each i, j, let $der_{i,j}$ be the SLD derivation of $P \cup \{G_i\}$ with c.a.s. $\theta_{i,j}$ of minimal length. Let us choose integers p, q in such a way that $der_{p,q}$ has minimal length among the derivations in the set $\{der_{i,j}\}$. The fact that $der_{p,q}$ has minimal length implies that for any goal G' in S, the following holds: if there exists an SLD derivation of $P \cup \{G'\}$ of length smaller than *len*$(der_{p,q})$ with c.a.s. θ', then the algorithm generates a solution ϑ' for which $G'\theta'$ is a renaming of $G'\vartheta'$ $(*)$.

Let c be the clause used in the first step of the derivation $der_{p,q}$. If c is a fact, we immediately have a contradiction: since G_p is a goal in S, this means that there exists an evaluation tree of $G_p = \leftarrow A_p$ created by CREATE TABLE (lines 2-7) with root node $\langle id, A_p \leftarrow A_p, new\rangle$ and a node with clause c as subnode of the root node. Therefore, the algorithm will compute a c.a.s. equivalent to $\theta_{p,q}$ (ACTIVATE NODE), contradicting the hypothesis.

If c is a rule $H \leftarrow B_1, \ldots, B_n$, and $\sigma_0 = mgu(G_p, H)$, then by hypothesis there exist SLD derivations $der_{B_1}, \ldots, der_{B_n}$, and substitutions $\sigma_1, \ldots, \sigma_n$ s.t. $H\sigma_0 \cdots \sigma_n = G_p\theta_{p,q}$, and for each $i \in \{1, \ldots, n\}$:

- der_{B_i} is an SLD derivation of $P \cup \{\leftarrow B_i\sigma_0 \cdots \sigma_{i-1}\}$.
- The c.a.s. of der_{B_i} is σ_i, and $len(der_{B_i}) < len(der_{p,q})$. (**)

Since G_p is a goal in S, there exists an evaluation tree of G_p created by CREATE TABLE (lines 2-7) with root node $\langle id, A_p \leftarrow A_p, new\rangle$ and a node with clause c as subnode of the root node. Then, it is easy to see that for each $i \in \{1, \ldots, n\}$:

- There exists a node in the evaluation tree of G_p with selected atom $B_i\sigma_0 \cdots \sigma_{i-1}$ (ACTIVATE NODE, lines 8, 18-19).
- There exists an evaluation tree of $\leftarrow B_i\sigma_0 \cdots \sigma_{i-1}$ created by CREATE TABLE (lines 2-7) at the location of $B_i\sigma_0 \cdots \sigma_{i-1}$.
- Since $len(der_{B_i}) < len(der_{p,q})$, by (*) and (**) the algorithm computes a solution equivalent to σ_i of the goal $\leftarrow B_i\sigma_0 \cdots \sigma_{i-1}$.
- By Propositions 2 and 3, the answer $B_i\sigma_0 \cdots \sigma_i$ is sent to the requester of $\leftarrow B_i\sigma_0 \cdots \sigma_{i-1}$ by GENERATE RESPONSE (lines 12-14 or 20-23) if $\leftarrow B_i\sigma_0 \cdots \sigma_{i-1}$ is involved in a loop, or by TERMINATE (lines 3-5) otherwise.
- There exists a node with clause $(H \leftarrow B_{i+1}, \ldots, B_n)\sigma_0 \cdots \sigma_i$ added to the evaluation tree of G_p by PROCESS RESPONSE (lines 20-23).

Therefore, $\sigma_1 \cdots \sigma_n$ is (equivalent to) a solution of the evaluation tree of G_p, contradicting (a), (b), and (c). □

Proof of Theorem 3. We assume that nodes (i.e., goals) in the call graph of P inherit the identifier (and the associated ordering) of the request for which they are created. Termination follows from two observations: (i) the call graph of P is finite, and (ii) the number of response messages exchanged by the principals involved in the evaluation of G is finite.
The call graph of P is finite (i) for the following reasons:

1. The set of goals over predicates in P (up to renaming) is finite. This is because terms that are not variables are constants in P.
2. There is no infinite path in the call graph of P composed of nodes $id_1, \ldots, id_n$ s.t. $id_n \sqsubset \ldots \sqsubset id_1$. This is because of (1) and because the algorithm never creates a new node with identifier id_i for a goal if a node with identifier id_j already exists for a variant of that goal and $id_i \sqsubset id_j$.
3. The outdegree of each node in the call graph of P is finite. This is because the number of atoms in the body of each clause in P is finite.

The number of response messages is finite (ii) because:

1. The number of answers of each goal defined in P is finite (see the proof of Proposition 3).

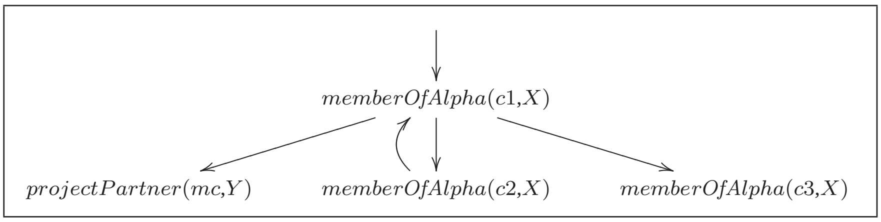


Fig. B 1. Call Graph of the Evaluation of *memberOfAlpha(c*1*,X)* with Respect to the Example Global Policy

2. The (possibly empty) set of answers of a goal are transmitted only when a table for the goal is first created (and a node representing the goal is added to the call graph of P) or new answers of its subgoals are received.
3. For any nodes id_1 and id_2, a set of answers that flows from id_2 to id_1 in response to a request id_2 never contains answers previously communicated in response to request id_2 (SEND RESPONSE, lines 3-4).
4. An empty set of answers may flow from id_2 to id_1 only if $id_2 \sqsubset id_1$ (GENERATE RESPONSE, lines 20-23, and TERMINATE, lines 3-5), or id_1 identifies a lower request and a loop id_2 has just been identified (PROCESS REQUEST, lines 5-7).
5. There is no infinite path composed of nodes $id_n, \ldots, id_1$ in the call graph of P through which the answers flow s.t. $id_n \sqsubset \ldots \sqsubset id_1$.
6. By Proposition 3, procedure TERMINATE is eventually invoked for any goal. $\square$

Appendix B Example

In this section we show how GEM computes the answers of a goal using the procedures presented in Section 3.3 of the paper. As an example global policy, we use a fragment of the policy introduced in Section 1. In particular, we consider the following policy statements:

1. memberOfAlpha$(c1,X) \leftarrow$ projectPartner(mc,Y), memberOfAlpha(Y,X).
2. projectPartner$(mc,c2)$.
3. projectPartner$(mc,c3)$.
4. memberOfAlpha$(c2,X) \leftarrow$ memberOfAlpha$(c1,X)$.
5. memberOfAlpha$(c2,alice)$.
6. memberOfAlpha$(c3,bob)$.

The call graph of the global policy is shown in Figure B 1. We illustrate the computation for an initial request *(*h_1*,h,memberOfAlpha(c*1*,X))* from hospital h to company $c1$. Table B 1 shows the list of all procedure calls made by GEM to produce the response to the initial request. The first column of the table indicates the order in which the calls are made; the second column denotes the principal and location where each procedure is evaluated. GEM computes the answers of goal *memberOfAlpha(c*1*,X)* by making 53 procedure calls; the number of messages exchanged between different principals, however, is only 14, consisting of 5 request messages and 9 response messages (including the initial request and its response). Next, we present and discuss some “screenshots” showing the status of the computation at various stages.

Call	Principal	Procedure
1	c1	PROCESS REQUEST((h_1,h,memberOfAlpha(c1,X)))
2	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
3	mc	PROCESS REQUEST((h_1c1_1,c1,projectPartner(mc,Y)))
4	mc	ACTIVATE NODE(projectPartner(mc,Y))
5	mc	ACTIVATE NODE(projectPartner(mc,Y))
6	mc	ACTIVATE NODE(projectPartner(mc,Y))
7	mc	GENERATE RESPONSE(projectPartner(mc,Y))
8	mc	TERMINATE(projectPartner(mc,Y))
9	mc	SEND RESPONSE((h_1c1_1,c1,projectPartner(mc,Y)),disposed,{})
10	c1	PROCESS RESPONSE(h_1c1_1,{projectPartner(mc,c2),projectPartner(mc,c3)},disposed,{})
11	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
12	c2	PROCESS REQUEST((h_1c1_2,c1,memberOfAlpha(c2,X)))
13	c2	ACTIVATE NODE(memberOfAlpha(c2,X))
14	c1	PROCESS REQUEST(($h_1c1_2c2_1$,c2,memberOfAlpha(c1,X)))
15	c1	SEND RESPONSE(($h_1c1_2c2_1$,c2,memberOfAlpha(c1,X)),active,{h_1})
16	c2	PROCESS RESPONSE($h_1c1_2c2_1$,{},active,{h_1})
17	c2	ACTIVATE NODE(memberOfAlpha(c2,X))
18	c2	ACTIVATE NODE(memberOfAlpha(c2,X))
19	c2	GENERATE RESPONSE(memberOfAlpha(c2,X))
20	c2	SEND RESPONSE((h_1c1_2,c1,memberOfAlpha(c2,X)),active,{h_1})
21	c1	PROCESS RESPONSE(h_1c1_2,{memberOfAlpha(c2,alice)},active,{h_1})
22	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
23	c3	PROCESS REQUEST((h_1c1_3,c1,memberOfAlpha(c3,X)))
24	c3	ACTIVATE NODE(memberOfAlpha(c3,X))
25	c3	ACTIVATE NODE(memberOfAlpha(c3,X))
26	c3	GENERATE RESPONSE(memberOfAlpha(c3,X))
27	c3	TERMINATE(memberOfAlpha(c3,X))
28	c3	SEND RESPONSE((h_1c1_3,c1,memberOfAlpha(c3,X)),disposed,{})
29	c1	PROCESS RESPONSE(h_1c1_3,{memberOfAlpha(c3,bob)},disposed,{})
30	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
31	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
32	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
33	c1	GENERATE RESPONSE(memberOfAlpha(c1,X))
34	c1	SEND RESPONSE(($h_1c1_2c2_1$,c2,memberOfAlpha(c1,X)),loop(h_1),{})
35	c2	PROCESS RESPONSE($h_1c1_2c2_1$,{memberOfAlpha(c1,alice),memberOfAlpha(c1,bob)},loop(h_1),{})
36	c2	ACTIVATE NODE(memberOfAlpha(c2,X))
37	c2	ACTIVATE NODE(memberOfAlpha(c2,X))
38	c2	ACTIVATE NODE(memberOfAlpha(c2,X))
39	c2	GENERATE RESPONSE(memberOfAlpha(c2,X))
40	c2	SEND RESPONSE((h_1c1_2,c1,memberOfAlpha(c2,X)),loop(h_1),{h_1})
41	c1	PROCESS RESPONSE(h_1c1_2,{memberOfAlpha(c2,bob)},loop(h_1),{h_1})
42	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
43	c1	ACTIVATE NODE(memberOfAlpha(c1,X))
44	c1	GENERATE RESPONSE(memberOfAlpha(c1,X))
45	c1	TERMINATE(memberOfAlpha(c1,X))
46	c1	SEND RESPONSE((h_1,h,memberOfAlpha(c1,X)),disposed,{})
47	c1	SEND RESPONSE(($h_1c1_2c2_1$,c2,memberOfAlpha(c1,X)),disposed,{})
48	c2	PROCESS RESPONSE($h_1c1_2c2_1$,{},disposed,{})
49	c2	ACTIVATE NODE(memberOfAlpha(c2,X))
50	c2	GENERATE RESPONSE(memberOfAlpha(c2,X))
51	c2	TERMINATE(memberOfAlpha(c2,X))
52	c2	SEND RESPONSE((h_1c1_2,c1,memberOfAlpha(c2,X)),disposed,{})
53	c1	PROCESS RESPONSE(h_1c1_2,{},disposed,{})

Table B 1. *Procedure Call Stack For the Example Global Policy*

When principal $c1$ receives the request for goal *memberOfAlpha($c1$,X)* from h, it calls procedure PROCESS REQUEST (Algorithm 1 in Section 3.3 of the paper) that initializes

Principal c1	
HR	(h_1,h,←memberOfAlpha(c1,X))
LR	{}
ActiveGoals	{}
AnsSet	{}
Tree	(h_1,memberOfAlpha(c1,X)← memberOfAlpha(c1,X),new)
	(h_1c1_1,memberOfAlpha(c1,X)← projectPartner(mc,Y), memberOfAlpha(Y,X),new)

Table B 2. *Status of the Computation After Procedure Call 1 in Table B 1*

Principal c1	
HR	(h_1,h,←memberOfAlpha(c1,X))
LR	{}
ActiveGoals	{}
AnsSet	{}
Tree	(h_1,memberOfAlpha(c1,X)← memberOfAlpha(c1,X),active)
	(h_1c1_1,memberOfAlpha(c1,X)← projectPartner(mc,Y), memberOfAlpha(Y,X),active)
Principal mc	
HR	(h_1c1_1,c1,←projectPartner(mc,Y))
LR	{}
ActiveGoals	{}
AnsSet	{(projectPartner(mc,c2),{}),(projectPartner(mc,c3),{})}
Tree	(h_1c1_1,projectPartner(mc,Y)← projectPartner(mc,Y),new)
	($h_1c1_1mc_1$,projectPartner(mc,c2),answer)
	($h_1c1_1mc_2$,projectPartner(mc,c3),answer)

Table B 3. *Status of the Computation After Procedure Call 7 in Table B 1*

the table of the goal. Table B 2 shows the table of *memberOfAlpha(c1,X)* resulting from the execution of PROCESS REQUEST on the initial request. The table field HR (higher request) is set to the initial request, and the evaluation tree of the goal, $Tree$, is initialized by adding to the root node a subnode representing the only clause in $c1$'s local policy applicable to the goal, i.e., clause 1. The node status is set to new, and the node identifier is obtained by concatenating the request identifier h_1 with string $c1_1$. To keep the representation more compact, in Table B 2 and in the other tables presented in this section the evaluation tree of a goal is represented as a list of nodes rather than as the structure defined in Section 3.3 of the paper.

In order to compute the list of project members, $c1$ needs to first retrieve from *mc* the list of partner companies in the project, and then for each of these companies the list of its project members. Table B 3 shows the status of the computation after goal *projectPartner(mc,Y)* has been completely evaluated by *mc* (procedure calls 2 to 7 in Table B 1), i.e., after the set of project partners has been computed. The request for goal *projectPart-*

Principal c1	
HR	(h_1,h,←memberOfAlpha(c1,X))
LR	{($h_1c1_2c2_1$,c2,←memberOfAlpha(c1,X))}
ActiveGoals	{}
AnsSet	{}
Tree	(h_1,memberOfAlpha(c1,X)← memberOfAlpha(c1,X),active)
	(h_1c1_1,memberOfAlpha(c1,X)← projectPartner(mc,Y), memberOfAlpha(Y,X),disposed)
	(h_1c1_2,memberOfAlpha(c1,X)← memberOfAlpha(c2,X),active)
	(h_1c1_3,memberOfAlpha(c1,X)← memberOfAlpha(c3,X),new)

Principal mc	
HR	null
LR	{}
ActiveGoals	{}
AnsSet	{(projectPartner(mc,c2),{h_1c1_1}),(projectPartner(mc,c3),{h_1c1_1})}
Tree	(h_1c1_1,projectPartner(mc,Y)← projectPartner(mc,Y),disposed)
	($h_1c1_1mc_1$,projectPartner(mc,c2),answer)
	($h_1c1_1mc_2$,projectPartner(mc,c3),answer)

Principal c2	
HR	(h_1c1_2,c1,←memberOfAlpha(c2,X))
LR	{}
ActiveGoals	{}
AnsSet	{}
Tree	(h_1c1_2,memberOfAlpha(c2,X)← memberOfAlpha(c2,X),active)
	($h_1c1_2c2_1$,memberOfAlpha(c2,X)← memberOfAlpha(c1,X),active)
	($h_1c1_2c2_2$,memberOfAlpha(c2,alice),new)

Table B 4. *Status of the Computation After Procedure Call 15 in Table B 1*

ner(mc,Y) from $c1$ to *mc* is generated by the activation of node h_1c1_1 in $c1$'s table (procedure call 2, ACTIVATE NODE), which results in a change of status from new to $active$ of both the root node and the node itself. Similarly to $c1$, when *mc* receives the request it creates the table of the goal (call 3 in Table B 1), setting HR to the higher request and initializing $Tree$ with clauses 2 and 3 of the global policy presented above. Two calls to procedure ACTIVATE NODE (calls 4 and 5) lead to the identification of two answers of the goal, namely *projectPartner(mc,c2)* and *projectPartner(mc,c3)*, which are added to $AnsSet$ with an empty list of request identifiers. At the next call to ACTIVATE NODE (call 6), the evaluation tree of goal *projectPartner(mc,Y)* has no more nodes to activate (i.e., all the branches of the evaluation tree have been inspected) and procedure GENERATE RESPONSE is invoked (call 7). Since the goal is not involved in any loop, its evaluation is completed and procedure TERMINATE is executed next (line 3 of Algorithm 6 in Section 3.3 of the paper).

As a result of the execution of procedure TERMINATE, the root node of the evaluation tree of goal *projectPartner(mc,Y)* is disposed and the answers identified are sent to $c1$ through procedure SEND RESPONSE (procedure call 9, results shown in Table B 4). The response message received by $c1$ is processed by procedure PROCESS RESPONSE; the message contains the two answers (*projectPartner(mc,*$c2$*)* and *projectPartner(mc,*$c3$*)*) and an empty set of loop identifiers, and has status $disposed$, indicating that no more answers of goal *projectPartner(mc,Y)* will be received. The evaluation tree of goal *memberOfAlpha(*$c1$*,X)* is updated by adding two subnodes to node h_1c1_1, one for each project partner (see $c1$'s table in Table B 4).

The activation of node h_1c1_2 by $c1$ leads to the request for goal *memberOfAlpha(*$c2$*,X)* to $c2$. Accordingly, $c2$ creates a table for the goal; the evaluation tree of the goal consists of three nodes: the root node and two subnodes, representing clauses 4 and 5 of the global policy, with identifiers $h_1c1_2c2_1$ and $h_1c1_2c2_2$ respectively. The activation of node $h_1c1_2c2_1$ by $c2$, in turn, leads to a request for goal *memberOfAlpha(*$c1$*,X)* to $c1$, forming a loop. The loop is identified by $c1$ in procedure PROCESS REQUEST (call 14 in Table B 1): in fact, the identifier of the higher request for *memberOfAlpha(*$c1$*,X)* (h_1) is a prefix of the identifier of $c2$'s request ($h_1c1_2c2_1$). Therefore, the lower request is added by $c1$ to set LR, and a response is sent from $c1$ to $c2$ with a notification of loop h_1 (call 15).

The loop notification sent from $c1$ to $c2$ starts the loop processing phase, which involves procedure calls from 16 to 34 in Table B 1. The results of the loop processing phase are shown in Table B 5. Upon receiving the loop notification, $c2$ sets the status of the node whose evaluation formed the loop to *loop(*$\{h_1\}$*)* and "freezes" its evaluation; then, it proceeds with the evaluation of the other nodes of the evaluation tree. The activation of node $h_1c1_2c2_2$ (procedure call 17), in particular, leads to the first answer of the goal, i.e., *memberOfAlpha(*$c2$*,alice)*. Since at this point there are no more nodes to be activated, the computed answer can be sent to $c1$ with a notification about the loop. Before sending the answer, $c2$ sets the counter in $ActiveGoals$ to 1 (procedure GENERATE RESPONSE, call 19) and adds the identifier of HR to the set of recipients of answer *memberOfAlpha(*$c2$*,alice)* in $AnsSet$ (procedure SEND RESPONSE, call 20).

The loop is now processed at $c1$. After adding a subnode to the evaluation tree of goal *memberOfAlpha(*$c1$*,X)* for the answer received from $c2$, $c1$ freezes node h_1c1_2 and starts the evaluation of node h_1c1_3 (procedure call 22). This results in a request from $c1$ to $c3$ for the evaluation of goal *memberOfAlpha(*$c3$*,X)*. The only clause applicable to *memberOfAlpha(*$c3$*,X)* (clause 6 of the global policy) is a fact; therefore, the goal is completely evaluated after one call to procedure ACTIVATE NODE (call 24). The answer of the goal, *memberOfAlpha(*$c3$*,bob)*, is returned to $c1$ (procedure call 28). Since the status of the response message is $disposed$, $c1$ disposes node h_1c1_3 and adds subnode h_1c1_5 to it reflecting the answer received from $c3$ (procedure PROCESS RESPONSE, call 29). The next two executions of procedure ACTIVATE NODE at $c1$ lead to the identification of two answers of goal *memberOfAlpha(*$c1$*,X)*, namely *memberOfAlpha(*$c1$*,alice)* and *memberOfAlpha(*$c1$*,bob)*. Before returning these answers to the requester of HR (i.e., h), however, all the loops need to be fully processed. For this reason, $c1$ sends the two answers to $c2$ in response to LR first; the status of the response message is *loop(*h_1*)*, and the status of the root node of the evaluation tree in $c1$'s table is changed accordingly (procedure calls 33 and 34 in Table B 1).

Now, the second iteration of the loop processing phase starts (procedure calls 35-44). In

Principal c1

HR	(h_1,h,←memberOfAlpha(c1,X))
LR	{($h_1c1_2c2_1$,c2,←memberOfAlpha(c1,X))}
ActiveGoals	{(h_1,1)}
AnsSet	{(memberOfAlpha(c1,alice),{$h_1c1_2c2_1$}),(memberOfAlpha(c1,bob),{$h_1c1_2c2_1$})}
Tree	(h_1,memberOfAlpha(c1,X)← memberOfAlpha(c1,X),loop({h_1}))
	(h_1c1_1,memberOfAlpha(c1,X)← projectPartner(mc,Y), memberOfAlpha(Y,X),disposed)
	(h_1c1_2,memberOfAlpha(c1,X)← memberOfAlpha(c2,X),loop({h_1}))
	(h_1c1_3,memberOfAlpha(c1,X)← memberOfAlpha(c3,X),disposed)
	(h_1c1_4,memberOfAlpha(c1,alice),answer)
	(h_1c1_5,memberOfAlpha(c1,bob),answer)

Principal mc

HR	null
LR	{}
ActiveGoals	{}
AnsSet	{(projectPartner(mc,c2),{h_1c1_1}),(projectPartner(mc,c3),{h_1c1_1})}
Tree	(h_1c1_1,projectPartner(mc,Y)← projectPartner(mc,Y),disposed)
	($h_1c1_1mc_1$,projectPartner(mc,c2),answer)
	($h_1c1_1mc_2$,projectPartner(mc,c3),answer)

Principal c2

HR	(h_1c1_2,c1,←memberOfAlpha(c2,X))
LR	{}
ActiveGoals	{(h_1,1)}
AnsSet	{(memberOfAlpha(c2,alice),{h_1c1_2})}
Tree	(h_1c1_2,memberOfAlpha(c2,X)← memberOfAlpha(c2,X),active)
	($h_1c1_2c2_1$,memberOfAlpha(c2,X)← memberOfAlpha(c1,X),loop({h_1}))
	($h_1c1_2c2_2$,memberOfAlpha(c2,alice),answer)

Principal c3

HR	null
LR	{}
ActiveGoals	{}
AnsSet	{(memberOfAlpha(c3,bob),{h_1c1_3})}
Tree	(h_1c1_3,memberOfAlpha(c3,X)← memberOfAlpha(c3,X),disposed)
	($h_1c1_3c3_1$,memberOfAlpha(c3,bob),answer)

Table B 5. *Status of the Computation After Procedure Call 34 in Table B 1*

this second iteration, $c2$ identifies a new answer of its goal, i.e., *memberOfAlpha($c2$,bob)*, which is sent back to $c1$. This answer, however, does not lead to new answers at $c1$. Since h_1 is the only loop in the SCC (and hence *memberOfAlpha($c1$,X)* is the leader of the SCC), and no new answers of *memberOfAlpha($c1$,X)* have been computed, the loop termination phase can start (line 15 of Algorithm 6 in Section 3.3 of the paper). In this phase, $c1$

Principal c1

HR	null
LR	$\{\}$
ActiveGoals	$\{\}$
AnsSet	$\{$(memberOfAlpha(c1,alice),$\{h_1c1_2c2_1,h_1\}$),(memberOfAlpha(c1,bob),$\{h_1c1_2c2_1,h_1\}$)$\}$
Tree	(h_1,memberOfAlpha(c1,X)$\leftarrow$ memberOfAlpha(c1,X),disposed)
	(h_1c1_1,memberOfAlpha(c1,X)$\leftarrow$ projectPartner(mc,Y), memberOfAlpha(Y,X),disposed)
	(h_1c1_2,memberOfAlpha(c1,X)$\leftarrow$ memberOfAlpha(c2,X),disposed)
	(h_1c1_3,memberOfAlpha(c1,X)$\leftarrow$ memberOfAlpha(c3,X),disposed)
	(h_1c1_4,memberOfAlpha(c1,alice),answer)
	(h_1c1_5,memberOfAlpha(c1,bob),answer)
	(h_1c1_6,memberOfAlpha(c1,bob),answer)

Principal mc

HR	null
LR	$\{\}$
ActiveGoals	$\{\}$
AnsSet	$\{$(projectPartner(mc,c2),$\{h_1c1_1\}$),(projectPartner(mc,c3),$\{h_1c1_1\}$)$\}$
Tree	(h_1c1_1,projectPartner(mc,Y)$\leftarrow$ projectPartner(mc,Y),disposed)
	($h_1c1_1mc_1$,projectPartner(mc,c2),answer)
	($h_1c1_1mc_2$,projectPartner(mc,c3),answer)

Principal c2

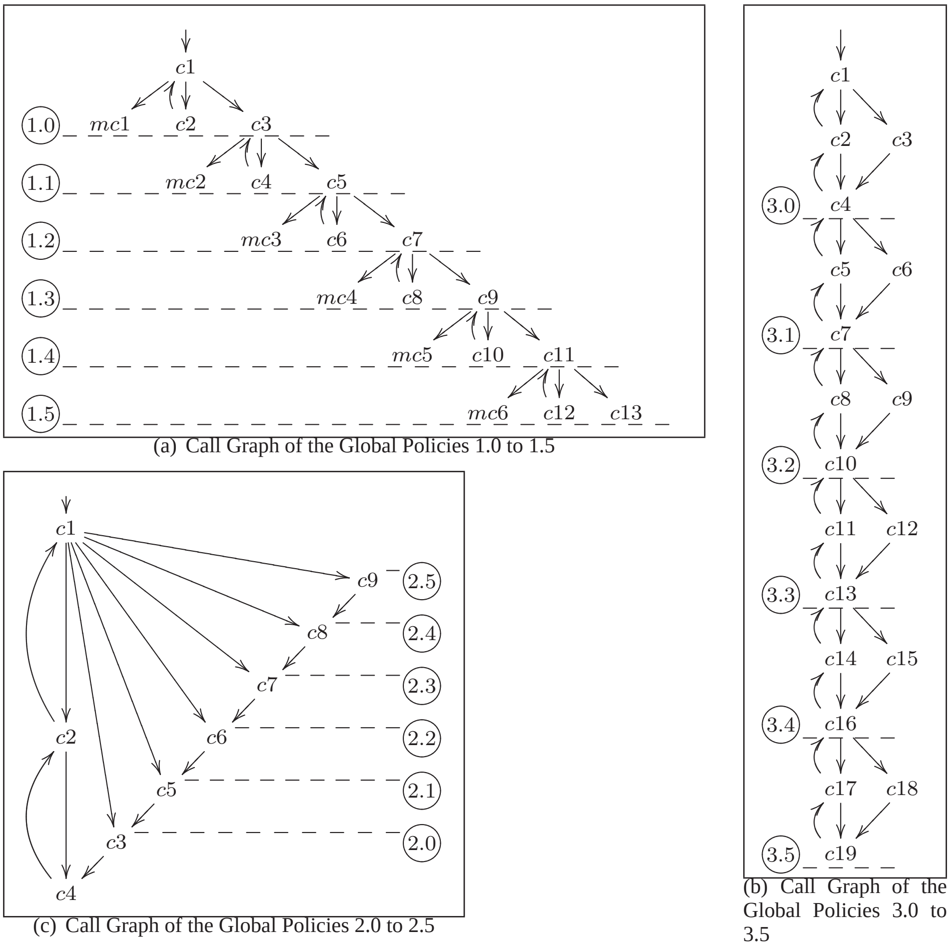
HR	null
LR	$\{\}$
ActiveGoals	$\{\}$
AnsSet	$\{$(memberOfAlpha(c2,alice),$\{h_1c1_2\}$),(memberOfAlpha(c2,bob),$\{h_1c1_2\}$)$\}$
Tree	(h_1c1_2,memberOfAlpha(c2,X)$\leftarrow$ memberOfAlpha(c2,X),disposed)
	($h_1c1_2c2_1$,memberOfAlpha(c2,X)$\leftarrow$ memberOfAlpha(c1,X),disposed)
	($h_1c1_2c2_2$,memberOfAlpha(c2,alice),answer)
	($h_1c1_2c2_3$,memberOfAlpha(c2,alice),answer)
	($h_1c1_2c2_4$,memberOfAlpha(c2,bob),answer)

Principal c3

HR	null
LR	$\{\}$
ActiveGoals	$\{\}$
AnsSet	$\{$(memberOfAlpha(c3,bob),$\{h_1c1_3\}$)$\}$
Tree	(h_1c1_3,memberOfAlpha(c3,X)$\leftarrow$ memberOfAlpha(c3,X),disposed)
	($h_1c1_3c3_1$,memberOfAlpha(c3,bob),answer)

Table B 6. *Final Status of the Computation for the Example Global Policy*

sends a response message with status $disposed$ to both $c2$ (the other principal in the loop) and *h* (to which also the answers are sent). Upon receiving this message, $c2$ disposes all



(a) Call Graph of the Global Policies 1.0 to 1.5

(b) Call Graph of the Global Policies 3.0 to 3.5

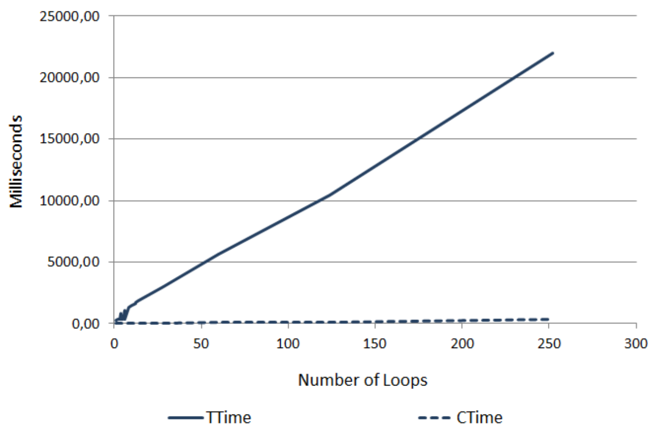
(c) Call Graph of the Global Policies 2.0 to 2.5

Fig. C 1. Call Graph of the Global Policies Used in the Experiments Set 1

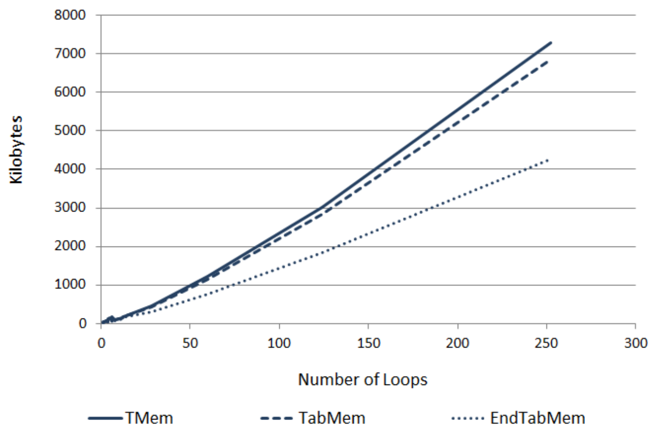
the nodes in the evaluation tree of *memberOfAlpha(c*2*,X)* that are involved in some loop (procedure PROCESS RESPONSE), and forwards the message back to *c*1 (calls 51 and 52). *c*1 simply ignores the message, as the status of the root node of the evaluation tree of *memberOfAlpha(c*1*,X)* is already $disposed$ (line 4 of Algorithm 5 in Section 3.3 of the paper), and the computation terminates. Table B 6 shows the status of the tables of all the goals at the end of the computation.

Appendix C Practical Evaluation

Figure C 1 shows how the global policies defined in Appendix B and in Section 3.1 of the paper have been modified to evaluate the performance of GEM in response to an increase in: (1) the number of principals and clauses (Figure 1(a)), (2) the number of loops (Figure 1(c)), and (3) both the number of principals, clauses and loops (Figure 1(b)) in a global policy. For each global policy, six variants have been created; in the figures, we use identifiers from x.0 to x.5 (where x is either 1, 2, or 3) to denote the variants, where variant



(a) Total and Computation Time for an Increasing Number of Loops in the Computation

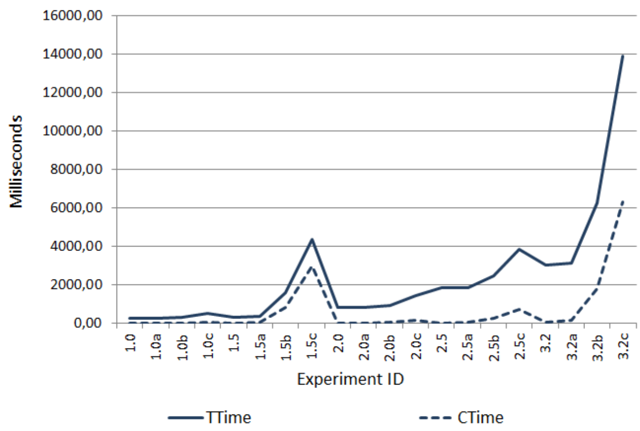


(b) Total and Tables Memory for an Increasing Number of Loops in the Computation

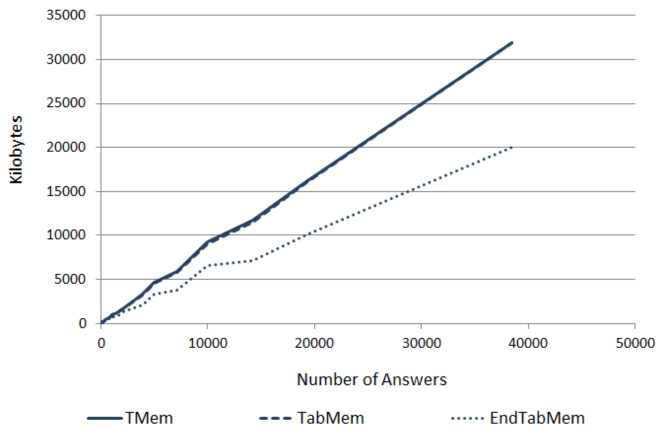
Fig. C 2. Time and Memory Results for Experiments Set 1

x.0 represents the original policy. To keep the figures as simple yet informative as possible, we label the nodes in the graph with the identifier of the principal evaluating the goal they represent rather than with the goal itself, as for the purpose of the experiments the number of principals involved in a computation is more relevant than the goals they evaluate.

Figures C 2 and C 3 provide a graphical overview of the main evaluation results of GEM, based on the values presented in Tables 1 and 2 in Section 5 of the paper.



(a) Time Results with Respect to the Number of Messages Exchanged in the Computation



(b) Memory Results with Respect to the Number of Answers Derived in the Computation

Fig. C 3. Time and Memory Results for Experiments Set 2