Goal-oriented coordination with cumulative goals

Aditya Ghose[1], Hoa Dam[1], Shunichiro Tomura[1], Dean Philp[2], and Angela Consoli[2]

[1] Decision Systems Lab, University of Wollongong, Wollongong, NSW, 2500, Australia
{aditya, hoa}@uow.edu.au; st655@uowmail.edu.au
[2] Defence Science and Technology Group, Edinburgh, SA, 5111, Australia
{dean.philp, angela.consoli}@dst.defence.gov.au

Abstract. Multi-actor coordination is in general a complex problem, with inefficient and brittle solutions. This paper addresses the problem of abstract coordination of multiple actors at the goal-level, which often enables us to avoid these problems. Goals in multi-actor settings often rely on the cumulative efforts of a group of distinct actors whose individual, measurable contributions accumulate to lead to the fulfillment of the goal. We formalize these as *cumulative goals* in a mathematical framework that permits us to measure individual actor contributions on a variety of scales (including qualitative ones). We view these as *softgoals*, a conception general enough to incorporate the more traditional *hardgoals*. Within this mathematical setting, we address the coordination problem as one of deciding which actor is allocated to fulfilling each goal of interest (and the associated problem of goal model maintenance). We also report some empirical evaluation results.

1 Introduction

Coordination is often a complex problem, with inefficient and brittle solutions. Goals afford a convenient means for managing complex underlying systems by enabling the use of high-level abstractions specifying the intent underpinning the behaviour being described. While the importance of multi-actor goal modelling is well-recognized in the literature [1], little attention has been paid to the problem of multi-actor *coordination* at the goal level ([2], [3] and [4] address a version of the problem indirectly. This paper aims to address this gap. Our work views all goals as *softgoals*, which include *hard goals* as a special case. In our conception, goals may be relaxed (although we define specific conditions under which goal decompositions are valid) and provide directions for improvement (in a similar sense to non-functional requirements, such as the one for *usability*, which requires us to improve usability as much as possible within the applicable constraints).

Goals are organized in the form of (informally defined) AND-OR goal trees where goals higher up in the tree are decomposed (via AND-decomposition or

OR-decomposition) to lower level sub-goals. AND-decomposition involves situations where the parent goal is deemed to be fulfilled only if all of the sub-goals have been fulfilled. The correctness of an AND-decomposition is checked via three tests (originally articulated by Dardenne et al [5]): (1) the conjunction of the goal conditions of the sub-goals must entail the parent goal condition (the *entailment* requirement; (2) the conjunction of the sub-goal conditions together with the parent goal condition must be consistent (the *consistency* requirement); (3) no subset of the sub-goals should satisfy the above two requirements (the *minimality* requirement). OR-decomposition involves situations where the parent goal is fulfilled if any one of the sub-goals are fulfilled. The correctness of an OR-decomposition is ensured by checking that each that the goal condition of each sub-goal entails the goal condition of the parent goal.

We introduce the notion of a *cumulative goal*. Informally, a cumulative goal is one where the goal state specification includes a measurable parameter whose target value is achieved by the *cumulative fulfillment* of its sub-goals. Thus a goal to make a cup of tea, AND-decomposed into three sub-goals (boil water, pour it into a cup and dunk a teabag in the cup) is not a cumulative goal since there is no measurable parameter with a target value. On the other hand, a goal to deliver 30 tons of produce, AND-decomposed into 3 sub-goals (each involving the delivery of 10 tons of produce, by a different trucking service provider in each case) is an example of a cumulative goal (the measurable parameter with a target value is the amount of produce delivered). We use the abstract algebraic framework of *c-semirings* [6] to formalize cumulative goals. This provides a general framework where the parameters of interest can be measured not just in terms of numeric weights as in the example of above (the *weighted* instance of a c-semiring) but also using fuzzy, probabilistic and even qualitative scales. Bistarelli et al [6] also show that new c-semirings can be obtained from composing existing c-semirings.

We show that cumulative goals are *preferential* in the sense that they lead to *preference orderings* on states of affairs. This feature is particularly useful is viewing cumulative goals as softgoals (as an informal example, we prefer to achieve states s_1, failing which we prefer to achieve state s_2 and so on). We then outline the use of cumulative goals in multi-actor coordination. In the same way that goal-oriented modeling of behaviour is sequence-agnostic (it abstracts away detailed action sequencing), our approach also abstracts away action sequencing, task scheduling and so on. We define an algorithm for goal model maintenance in settings where new actors become available (leading to new goals becoming feasible) or existing actors retire (rendering existing goals infeasible). We finally provide an algorithm that addresses the fundamental coordination problem: which actor is allocated to fulfilling each goal. We provide experimental results which suggest that the approach can be computationally realized and discuss related work.

The approach to goal-oriented coordination can find application in a wide variety of settings. In addition to the logistics and bushfire-fighting applications that the examples in this paper briefly touch upon, this approach can be used in any setting where UXV (Unmanned Aerial/Maritime/land Vehicle) coordi-

nation is necessary. It can also be used in managing teams of humans (hence in organizational management in general), or managing mixed teams of human and machine actors, amongst many others.

2 Cumulative goals

We leverage an approach to modelling *degrees of preference* using a subclass of semirings, formally defined below. A c-semiring is a tuple $\mathcal{V} = \langle V, \oplus, \otimes, \bot, \top \rangle$ satisfying (for all $\alpha \in V$) [6]:

- V is a set of abstract values with $\bot, \top \in V$.
- $\oplus$ is a commutative, associative, idempotent and closed binary operator on V with $\bot$ as unit element ($\alpha \oplus \bot = \alpha$) and $\top$ as absorbing element ($\alpha \oplus \top = \top$).
- $\otimes$ is a commutative, associative and closed binary operator on V with $\top$ as unit element ($\alpha \otimes \top = \alpha$) and $\bot$ as absorbing element ($\alpha \otimes \bot = \bot$).
- $\otimes$ distributes over $\oplus$ (i.e., $\alpha \otimes (\beta \oplus \gamma) = (\alpha \otimes \beta) \oplus (\alpha \otimes \gamma)$).

Informally, $\oplus$ helps us compare degrees of preference. For $a, b \in V$, $a \oplus b = b$ suggests that b is at least as preferred as a. Similarly, $\otimes$ helps us combine degrees of preference, as is illustrated in the examples below. Informally, $\bot$ is the least preferred value and $\top$ is the most preferred value.

The c-semiring instance that we have implicitly made reference to in the preceding examples is the *weighted* instance. Formally:

$$S_{weighted} =< \mathcal{R}^+, max, +, 0, +\infty > \tag{1}$$

In other words, we use the set of positive reals to measure the degree of preference (of a state, or a solution, depending on the application). The $\oplus$ operator is max, which means that we prefer higher weights. The $\otimes$ operator is arithmetic addition, which means that we combine weights by adding them. We can also define a fuzzy instance:

$$S_{fuzzy} =< \{x | x \in [0, 1]\}, \max, \min, 0, 1 > \tag{2}$$

In other words, we use reals between 0 and 1 to measure the degree of preference, max to compare fuzzy values (higher values are preferred) and min to combine. A probabilistic instance can be similarly defined:

$$S_{prob} =< \{x | x \in [0, 1]\}, \max, \times, 0, 1 > \tag{3}$$

Consider a qualitative instance: $\langle \{High, Medium, Low\}, \oplus, \otimes, Low, High \rangle$. Here we may choose to define $\oplus$ by explicit enumeration as follows: $High \oplus Medium = High, High \oplus Low = High, Medium \oplus Low = Medium$. *otimes* is also defined by explicit enumeration: $High \otimes Medium = Medium, High \otimes Low = Low, Medium \otimes Low = Low$.

We assume that each goal is written in the following format:

$\langle \phi, \{\langle \pi_0, v_0 \rangle, \ldots, \langle \pi_i, v_i \rangle, \ldots \langle \pi_n, v_n \rangle\}, C \rangle$

Here:

- ϕ is a formal assertion which the goal seeks to *achieve* or *maintain*
- Each of $\pi_0, \ldots, \pi_n$ is a parameter with a specified *target value*
- $v_0, \ldots, v_n$ are the target values for the parameters (v_0 for π_0 and so on)
- C is a collection of constraints, on a signature that is possibly a superset of the set of parameters referred to above, that specify permitted combinations of values of the parameters.

Another instance of c-semirings makes it amenable to supporting reasoning in propositional logic. We have however not used that approach to encoding ϕ in a goal because the resulting representation would be much harder to understand. We shall refer to $\langle v_0, \ldots, v_n \rangle$ as the *level of satisfaction* of the goal. Most goals and sub-goals in a goal model are *normative* in nature - they do not describe current reality but specify how reality ought to be. However, leaf goals assigned to specific actors (which are responsible for their fulfillment) are *descriptive* in nature - they describe the current state of reality. In such situations, the satisfaction level of a leaf goal is contingent on the actor it is assigned to (since different actors can fulfill the same goal to different levels of satisfaction).

We are now in a position to define an extended version of the rules governing the AND-reduction of goals [5].

A set of subgoals $\{G_0, \ldots, G_m\}$ is a valid AND-reduction of a goal G if and only if each of the following conditions are satisfied:

- $\phi_0 \wedge \ldots \wedge \phi_i \wedge \ldots \phi_n \models \phi$ (here ϕ is the formal assertion associated with goal G, ϕ_i the formal assertion associated with G_i and so on)
- $\phi_0 \wedge \ldots \wedge \phi_i \wedge \ldots \phi_n \wedge \phi \not\models \bot$
- There exists no subset of $\{\phi_0, \ldots, \phi_i, \ldots \phi_n\}$ which also satisfies the two conditions above
- For each i, $v_{i1} \otimes \ldots \otimes v_{in} \oplus v_i = v_{i1} \otimes \ldots \otimes v_{in}$, where v_i is the target value for parameter π_i in parent goal G, v_{ij} is the target value for parameter π_i is subgoal G_j and so on.

Figure 1 is an example of a valid AND-decomposition. The setting involves water-bombing a bushfire (of the kind common in Australia) using water-carrying drones. In general, an area affected by bushfire would require water-bombing by a large number of drones, but we consider only two drones here to simplify the example and save space. We also assume the existence of a background knowledge base which specifies the following:

$$\textit{payload-deliv-drone}1 \wedge \textit{payload-deliv-drone}2 \models \textit{water} - \textit{bombed}$$

Figure 2 is an example of an invalid AND-decompposition according to the definition above. In addition to the water-quantity parameter, we introduce an additional parameter for precision, which assesses how precise a drone is in delivering its payload to the specified target area. The decomposition is invalid because while the first drone exhibits a high degree of precision, the second drone exhibits only a medium degree of precision and $High \otimes Medium = Medium$ (and the parent goal requires a high degree of precision.

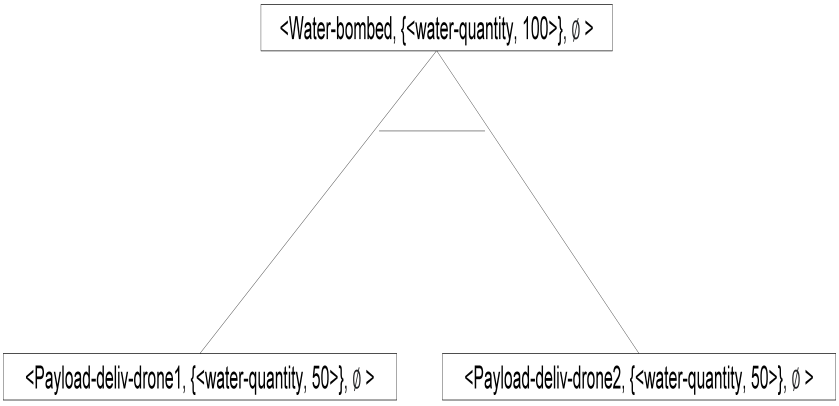

Fig. 1. Example of valid AND-decomposition

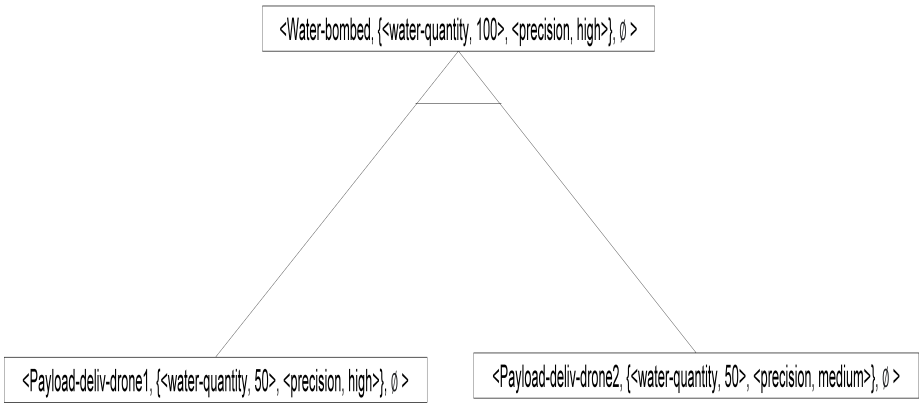

Fig. 2. Example of invalid AND-decomposition

3 Cumulative (preferential) goals

The notion of a *softgoal* has long been recognized in the literature. Softgoals are goals that do not mandatorily have to be satisfied (as opposed to *hard goals*) and admit some modicum of relaxation. They are typically used to model non-functional requirements, although they can also be useful in a range of other settings.

The conception of cumulative goal presented earlier in the paper allows us to view each such goal providing a preference ordering on a set of states.

A goal G of the form:

$$\langle \phi, \{\langle \pi_0, v_0 \rangle, \ldots, \langle \pi_n, v_n \rangle\}, C \rangle$$

leads to a preference ordering (most generally a partial order) $\prec_G$ on the set of states (this is based on a result in [6] that establishes that a c-semiring in general leads to a partially ordered set of preference values). Let state s_1 be of the form:

$$\langle \phi_1, \{\langle \pi_j, v_j \rangle, \ldots, \langle \pi_s, v_s \rangle\}, C_1 \rangle$$

Let state s_2 be of the form:

$$\langle \phi_2, \{\langle \pi_k, v_k \rangle, \ldots, \langle \pi_t, v_t \rangle\}, C_2 \rangle$$

$s_1 \prec_G s_2$ if and only if:

- $\phi_1 \models \phi$ and $\phi_2 \models \phi$
- Each of C, C_1 and C_2 are individually satisfiable
- $\{\pi_0, \ldots, \pi_n\} \cap \{\pi_j, \ldots, \pi_s\} \cap \{\pi_k, \ldots, \pi_t\} = \{\pi_u, \ldots, \pi_r\}$
- $v_{u1} \oplus v_{u2} = v_{u2}.v_{r1} \oplus v_{r2} = v_{r2}$ and so on, where v_{u1} is the value associated with parameter π_u in state s_1, v_{u2} is the value associated with parameter π_u in state s_2 etc.

4 Coordination with cumulative goals

We work from the premise that dynamic nature of the operating context triggers the need for goal-oriented coordination. New actorss with new capabilities may become available while existing actors might retire or otherwise become unavailable. The overall coordination model we work with assumes that there is a central coordinator agent that maintains the system-wide goal model while agents at the edge assume responsibility for the fulfillment of specific goals and opportunistically handover, or swap goals with other agents.

We outline a procedure (Algorithm 1 below) for maintaining goal models as new sub-goals become feasible to fulfill, or when existing goals or sub-goals become infeasible to fulfill. We use the notion of a *k-replacement goal model maintenance strategy*. In the case of new sub-goals becoming available, this involves taking any subset of size k of the set of new sub-goals and replacing a set of *sibling goals* (i.e., goals that share the same parent) with this set of size k. If the input set of newly feasible goals is of size n, we need to consider $\binom{n}{k}$ possibilities. In the case of existing sub-goals becoming infeasible, this involves picking any subset of length k from the set of available sub-goals (in Algorithm 1, this is SUBGOAL-LIBRARY, which is the set of sub-goals that the actor knows how to fulfill) and seeking to replace the infeasible sub-goal(s) with this set. If the size of the set of available sub-goals is of size m, we need to work through $\binom{m}{k}$ possibilities. Note that m is often much larger than n, but a variety of relevance reasoning techniques could be used to reduce the size of the set of available sub-goals at the pre-processing stages. In the following, we use a function SLEVEL() which returns the *satisfaction level* (as defined earlier) for the goal provided as input.

When a new sub-goal is introduced or an existing sub-goal is removed, the entire sub-tree rooted at the corresponding sub-goal is also removed. Algorithm

1 can be repeatedly called (in the REMOVAL mode) to re-populate these subtrees. Including pseudo-code for this in Algorithm 1 would make it complex and difficult to explain (and understand); hence we have chosen to summarize it in this text fragment above.

Algorithm 1 Agent coordination with central coordinator

Input: model: goal model, indicator: represented as ADDITION or REMOVAL, NEW: a set of new subgoals in case of ADDITION, REMOVE: the specific subgoal to be removed in case of REMOVAL, SUBGOAL-LIBRARY: for available subgoals used in case of REMOVAL
Output: A new goal model

```
1: if indicator=ADDITION then
2:    for each non-root goal GOAL do
3:       SAT-LEVEL := satisfaction level of GOAL
4:    end for
5:    for i= 1 To |NEW| do
6:       for each subgoal G of GOAL do
7:          for each REPLACE = subset of size i of NEW do
8:             set GOAL-TMP := GOAL with G replaced with REPLACE, provided
               the replacement satisfies the rules for valid AND-decompositions
9:             if SLEVEL(GOAL-TMP) ⊕ SLEVEL(GOAL) = SLEVEL(GOAL-
               TMP) then
10:               set SAT-LEVEL := SLEVEL(GOAL-TMP) and GOAL := GOAL-
                  TMP
11:            end if
12:         end for
13:      end for
14:   end for
15: else if indicator=REMOVAL then
16:   set GOAL to be the parent goal of the goal being removed
17:   set SAT-LEVEL := satisfaction level of GOAL
18:   for each REPLACE= subset of size i of SUBGOAL-LIBRARY do
19:      set GOAL-TMP := GOAL with REPLACE added as one or more additional
         subgoals, provided the result satisfies the rules for valid AND-decompositions
20:   end for
21:   if SLEVEL(GOAL-TMP) ⊕ SLEVEL(GOAL) = SLEVEL(GOAL-TMP) then
22:      set SAT-LEVEL := SLEVEL(GOAL-TMP) and GOAL := GOAL-TMP
23:   end if
24: end if
```

We next outline a procedure (Algorithm 2) to be used by agents to decide if a goal swap or handover is necessary. Recall that leaf-level goals are assigned to individual actors. These actorss thus assume responsibility for fulfillment of these leaf-level goals. In Algorithm 2, we only deal with leaf-level nodes, since these are only ones that can be assigned to different actors. $\text{SLEVEL}_A(\text{G})$ refers to the satisfaction level of goal G when assigned to actor A.

Algorithm 2 Goal swap/handover

Input: model: goal model, A: an agent with specified capabilities
Output: A possibly modified goal model. The possible modification would involve the assignment of the input agent to a leaf-level goal.

```
1: set CANDIDATES := ∅
2: for each leaf-level goal G currently assigned to agent A' do
3:    if SLEVEL_A(G) ⊕ SLEVEL_A'(G) = SLEVEL_A(G) then
4:       set CANDIDATES := CANDIDATES ∪ {G}
5:    end if
6: end for
7: for each distinct GOAL∈CANDIDATES do
8:    for any DGOAL∈CANDIDATES do
9:       if SLEVEL_A(GOAL)⊕ SLEVEL_A(DGOAL) = SLEVEL_A(GOAL)  then
10:         set CANDIDATES := CANDIDATES − DGOAL
11:         non-deterministically select any goal from CANDIDATES and assign A to
            it
12:      end if
13:   end for
14: end for
```

5 Empirical evaluation

In the following, we assume that the set of constraints is empty (i.e., the parameters are independent of each other). Including constraint satisfaction will lead to the well-known complexity of satisfiability checking.

5.1 Goal satisfaction check

: We implemented the cumulative goal satisfaction check to show the achievability of our concept in reality. The aim of cumulative goal check is to investigate the efficiency of the c-semiring approach for measuring the cumulative goal satisfiability in various scenarios. We implement six respective checkers under respective scenarios; boolean, fuzzy, weight, quality, probability and integration scenarios. Through this experiment, we expect to assure the feasibility of the cumulative goal satisfiability check for all the respective checkers in a timely manner. The experimental result answers the question which expresses doubt in the realization of our ideas. This experiment consists of several steps.

1. Checker design: Each case is designed based on the definitions discussed in the previous sections. There are three main components in each checker. The additive ($\oplus$) and multiplicative ($\otimes$) are the operators which compute the satisfiability level of each individual and cumulative goals. Each operator is structured differently by following the definitions.

2. Goal generation: 10,000 goals are prepared for all the checkers except the integration checker (the total number of goals is 40,000). Each goal possesses five target variables. To simplify the experiment, target variable's desirable value

as 1 for the boolean, fuzzy, weight and probability checkers. The value is set as 3 for the quality checker. In the integration checker, 1 is the desirable value for the boolean, fuzzy, weight and probability checker parts while 3 is used for the quality checker as well. The satisfiability level of these individual goals determine the cumulative goal satisfiability level.

3. Data generation: Data is generated in order to simulate a real situation. For the boolean, fuzzy and weight, we generate random numbers for each actual observed value from the range of 0 to 2. Each observed value in the generated data for the quality checker is chosen from the integer number between 1 to 5. The probability checker uses the data with variables assigned either 'A', 'B', 'C', 'D' or 'E'. One character among them is selected under uniform probability. The integration checker uses these three dataset types since it combines all the different checkers. All the datasets consist of 10,000 records (goals) with eight attributes for the first dataset and five attributes for the second and third dataset. For the first dataset, the labels for each checker are added as respective attributes beside the variables. The other datasets entail one label for the quality or probability checker. The last row in all the datasets show the labels for the cumulative goal satisfiability level of each checker. We create three datasets for each dataset type and hence nine different datasets are generated. In total, we execute the same experiment with different datasets to increase the result reliability.

4. Individual goal satisfiability check: Each goal's target variable is compared with the corresponding actual observed value. A returned result from the individual goal satisfiability check function differs from checkers.

Boolean: If a variable is equal to 1, then 1 is returned to the variable. Otherwise, 0 would be output. If all the observed variable values in a goal are the same with the target variable value (if all the variables receive 1), then 1 is returned to the goal. If one of the observed variables obtains 0 for the return, then 0 is returned to the goal as well.

Fuzzy: A membership function is applied in the fuzzy checker, which allows the checker to represent the return value between 0 and 1 by providing a fuzzy level. The experiment sets all the fuzzy levels as 0.5. Any observed values larger than or equal to 0.5 and smaller than or equal to 1.5 return non zero values. If an observed variable value lies on the range, then the return can be expressed as a value larger than or equal to 0 and smaller than or equal to 1. The returned value depends on the satisfiability level of the value. If the observed variable is close to 1, it obtains a value close to 1. The fuzzy expression enables the checker to determine goal satisfiability at a fuzzy level as well. Among the returned results from each target variable, the minimum value is selected as the representation of the goal's satisfiability level.

Weight: Each variable possesses a weight. Five variables for each goal are set as 2, 4, 6, 8 and 10. Each goal obtains the total cost by summing all the possessing variables' weight. The multiplication of weight with corresponding variable's returned value reveals the cost. The returned value is decided based on the variables' values in a fuzzy way.

Quality: Based on the variable numbers, the quality level of each variable is determined. Since the target value is 3 (moderate) and hence the value evaluation is based on the number 3. If the values are lower than 3, either low or cool is assigned depending on the value. In contrast, warm or hot is given to the number higher than 3. The values 1, 2, 3, 4, 5 correspond with cold, cool, moderate, warm and hot.

Probability: Each variable receives either Prob(check) or 1 - Prob(check) as return. Prob(check) is probability that indicates the correctness of the check. We set 'A', 'B', 'C' and 'D' as correct values. Thus, if a variable contains either of the characters, then the checker considers that the variable matches with its target values and Prob(check) is returned. However, if the variable possesses 'E', then the value 1 - Prob(check) is returned. In our experiment, we set the P(check) as 0.7. In reality, the probability depends on the information or criteria from a certain domain. The arithmetic multiplication of all the returned values from each variable leads to the satisfiability level of the goal.

Integration: In this experiment, the integration consists of one boolean, fuzzy, weight, quality and probability checkers. For the boolean, fuzzy and weight parts, the same dataset is given. However, each checker examines the individual goal satisfiability level based on its own method. The quality and probability checkers use its own dataset and compute the satisfiability level based on their operators. Hence, the total number of goals is 40,000. All the parameter settings are the same with the ones discussed in the other subsections.

5. Cumulative goal satisfiability computation: From individual goal satisfiability level, the cumulative goal satisfiability level is calculated by the multiplicative operator in each checker. For the integration checker, the cumulative result is shown as a collection of cumulative goal satisfiability levels from each checker.

The result from each checker is measured by two metrics. Accuracy measures the matching rate of the actual computed results with the labels. Since the experiment is conducted three times with different datasets, we average the observed accuracy. If all the actual components from a checker are identical with those from the expected result, the accuracy becomes 100%. For instance, the total labels for the boolean checker is 10,001 (10,000 goals plus 1 cumulative goal), and we check whether the returned values from the checker are identical with the labels. If all the components' values are the same with those from the expectation, then the accuracy is 100%. When the checker incorrectly computes the goal satisfiability level, then the returned result can differ from the expected result. This gap reduces the accuracy level. Large separation from the expected result leads to lower accuracy and indicates that it is infeasible to demonstrate our concept into actual implementation.

Another metric is time, which measures the computing time of each checker. The calculation includes the individual and cumulative goal satisfiability check. It is vital to achieve the small time value because it allows the checkers to be used in real-time goal satisfiability checking. If the checkers spend a large amount of time, then it is unrealistic to be exploited in reality. Time is averaged by the

result from each round. We use a 4GB memory, Intel(R) Core(TM) i5-7200U CPU, Surface Laptop. If the returned time is small although the device spec is not high, then we can demonstrate that the checkers are applicable to real issues.

The result of the experiment is shown in Table 1. As can be seen, the accu-

Table 1. Accuracy and time from each checker

	Accuracy		time (sec)
	Individual	Cumulative	
Boolean	1	1	0.035
Fuzzy	1	1	0.079
Weight	1	1	0.098
Quality	1	1	0.052
Probability	1	1	0.102
Integration	1	1	0.363

racy recorded as 100% for both individual and cumulative in all checkers, which validates the realization of our goal satisfiability level concept through the implementation. However, the accuracy is not supportive enough to conclude the achievability of our concept. The observation of recorded time is also required to check the ability of the checkers to check the goal satisfiability in real-time. From the Table 1, as the checkers become more complex (from the boolean to integration), the total amount of time increases. However, all the observed time is below one second and remains in small values with 10,000 goals checking for the first four checkers and 40,000 for the integration checker. This result demonstrates that our checkers can accurately check the goal satisfiability levels in a fast manner and can be used in real-time goal analysis if a single checker is given.

5.2 Time increase analysis for integration checker:

In the previous experiment, the accuracy of each checker is validated. In this experiment, we investigate the time increase rate for the integration checker when the total number of checkers in the integration checker increases. The aim of this experiment is to observe the possibility of the integration checker being able to check the goal satisfiability level in a reasonable time manner when the total checker numbers and goals rise. We analyze the effect of checker number increase by different checker types. We expect the time to increase with a linear pace in all the checker types because other increase patterns such as polynomial and exponential growth can lead to prohibitive computational time.

We measure the time change by increasing each checker up to five. Each added checker examines the satisfiability level of additional 10,000 goals. Therefore, when five new checkers are added, the total goals number becomes 50,000. While increasing the total numbers of one checker type, the others checker types are

not included. For example, in order to observe the time change by increasing the boolean checkers, we gradually augment the total number of the boolean checkers in the integration checker from one to five. When measuring the time difference, the other checker types are excluded in the integration checker. We use three different datasets to average the observed time. All the details of parameters in each checker type are the same with the ones in the previous experiment. The result is illustrated in Figure 3.

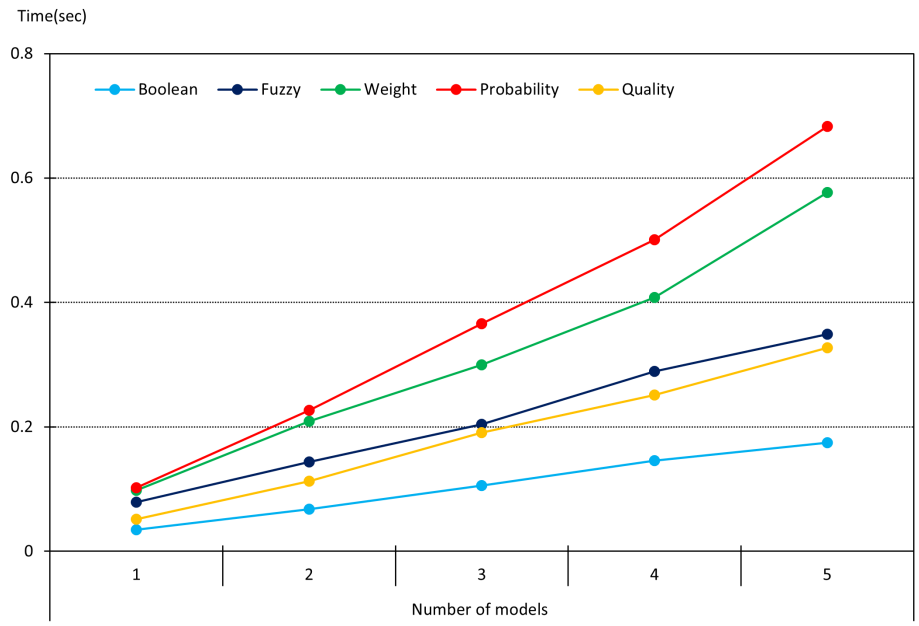

Fig. 3. Time change in integration checker by increasing checker numbers

The boolean checker achieves the lowest time rise at 0.55 seconds when the number is five, followed by quality and fuzzy at around 0.3 and 0.7 seconds respectively. The weight and probability checkers recorded longer time at approximately 0.6 for the former and 0.7 for the latter. One noticeable tendency is that though the slope of each line varies, their increase rates are in a linear form. We expect to observe a linear increase as discussed previously. The checker computational time increase can be suppressed in a controllable level even though the number of any checker types and goals increases. By combining both accuracy and time observed in both experiments, we can demonstrate that our concept is achievable to be utilized in reality under reasonable computational time. Additional experimental results can be found at:
https://github.com/ShunichiroTomura/G2I/blob/main/G2I.pdf

5.3 Algorithm 1 implementation

This experiment is designed to evaluate the achievability of the Algorithm 1. In our experiment, we assume that the indicator is set as addition. By showing that the addition operation can be implemented, then we can say that removal is also feasible because the only differences between the two indications are the motivation and a set of referred goals. In the addition indication, a set of new subgoals is given, and they may be replaced with the existing subgoals depending on the satisfiability level. In the removal indication, a removed subgoal is replaced with one of the achievable subgoals in the subgoal library. The core algorithm behind the scene is the same.

We consider one goal decomposition structure which is displayed in Figure 4. The goal 0 is the root goal for the entire goals in this context, which is decom-

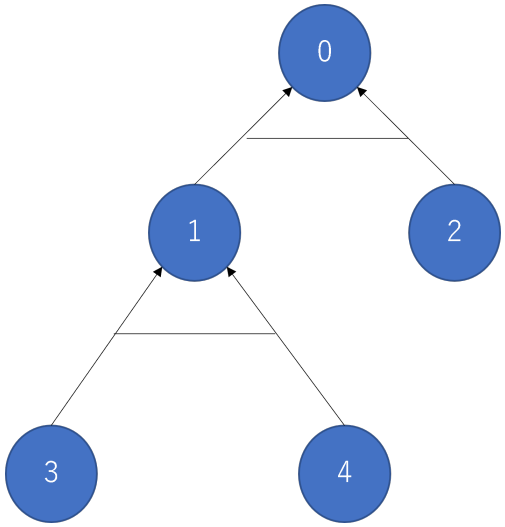

Fig. 4. The goal decomposition structure for this experiment

posed into the goal 1 and 2. The goal 1 is further decomposed into the goal 3 and 4. We assume that all the subgoals are AND-decomposed. Hence, the satisfaction of all subgoals is required to measure the satisfiability of the parent goal, which is one requirement for the algorithm. We test the algorithm in three different scenarios; boolean, fuzzy and weight check. In each scenario. We assume that only one type of checker is required and each goal contains only one variable for simplicity. The initial parameter setting for each goal is shown in Table 2. We assume that all the target variable's value in every goal is 1 and hence if the observed variable value in one variable is 1, then it demonstrates the complete satisfaction of the goal. As can be seen, goal 3 has an issue in the satisfiability level in each case. Goals 0 and 1 are left blank because their results depend on

Table 2. Initial setting

	Observed variable value					Fuzziness	Cost				
	Goal 0	Goal 1	Goal 2	Goal 3	Goal 4		Goal 0	Goal 1	Goal 2	Goal 3	Goal 4
Boolean	-	-	1	0	1	-	-	-	-	-	-
Fuzzy	-	-	1	0.9	1	0.5	-	-	-	-	-
Weight	-	-	1	0.9	1	0.5	2	2	2	2	2

their subgoals. Fuzziness shows the range of the membership function for the fuzzy and weight checks. The value is set as 0.5 for both checks. Cost indicates the weight for each goal used in the weight check. All the weights are set as 2.

By implementing the three different scenarios with different checkers, the initial result of the goal satisfiability is shown as Figure 5. The final goal satisfi-

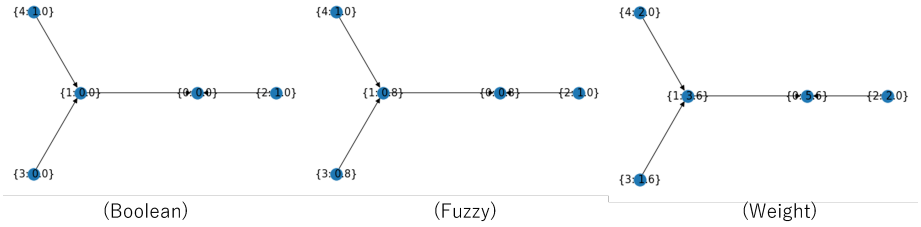

Fig. 5. Initial result from each scenario

ability level for each scenario is 0, 0.8 and 5.6 respectively. As mentioned, due to the low goal satisfiability of the goal 3, the final goal from each scenario cannot achieve the highest satisfiability level.

Now we assume that we obtain several new subgoals which can be replaceable with goal 3. The details are illustrated in Table 3. The three goals are 6, 7 and

Table 3. New goals

	Observed variable values in new goals		
	6	7	8
Boolean	0	0	1
Fuzzy	0	0.6	1
Weight	0	0.6	1

8. As can be seen, goal 8 seems to be able to achieve the highest satisfiability level among the new goals in all the scenarios. Based on the Algorithm 1, we compare the satisfiability of goal 3 with the new goals. We expect that the goal 3 is replaced with the goal 8 due to higher satisfiability level. By implementing the algorithm, we receive the result as Figure 6. Through the addition operation, goal

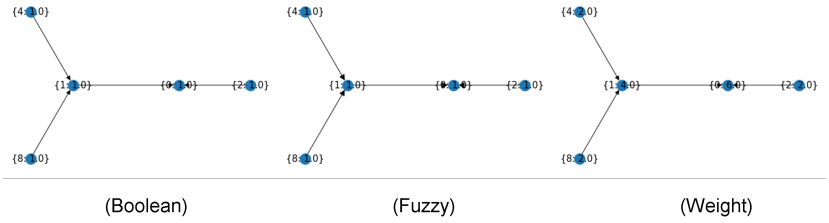

Fig. 6. Addition result in each scenario

3 is replaced with goal 8 because it can achieve higher goal satisfiability level. Consequently, the final goal satisfiability level increases to 1, 1 and 6 in each scenario. This result accords with our expectation and hence the algorithm 1 is demonstrated as an executable mechanism in reality.

5.4 Algorithm 2 implementation

The goal of this experiment is to investigate the achievability of Algorithm 2. In this experiment, we assume the structure of goal decomposition as Figure 7. One noticeable difference from the previous experiment is the further decompo-

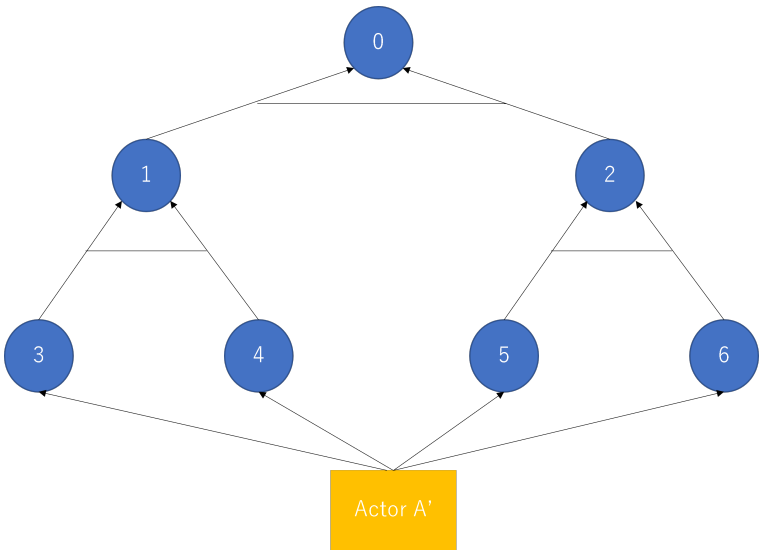

Fig. 7. The goal decomposition structure for this experiment

sition in goal 2. The goal is AND-decomposed into two subgoals 5 and 6. Due to this extension, the goal structure is now represented in a binary form. Another

difference is the existence of an actor. In this experiment, we explicitly denote the actor in order to check the correctness of the implementation. Actor A' is responsible for all the goal satisfiability activities. We also test the implementation of boolean, fuzzy and weight checkers by setting the three scenarios identical to the previous experiment. The initial parameter setting is summarized in Table 4. The Goal5 is the least satisfied goal in all the scenarios at 0. For the fuzzy and

Table 4. Initial setting

	Variable							Fuzziness	Cost						
	Goal 0	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5	Goal 6		Goal 0	Goal 1	Goal 2	Goal 3	Goal 4	Goal 5	Goal 6
Boolean	-	-	-	1	1	0	1	-	-	-	-	-	-	-	-
Fuzzy	-	-	-	1	1	0	0.8	0.5	-	-	-	-	-	-	-
Weight	-	-	-	1	1	0	0.8	0.5	2	2	2	2	2	2	2

weight cases, the second lowest satisfiability level is observed in the goal 6. Two different low satisfiability level are set in both the cases to demonstrate the realization of our algorithm through the implementation. The fuzziness and cost for each goal is set as 0.5 and 2 in the specific cases.

Based on this setting, we implement the boolean, fuzzy and weight cases. The result is shown in Figure 8. Neither of the cases achieves the maximum

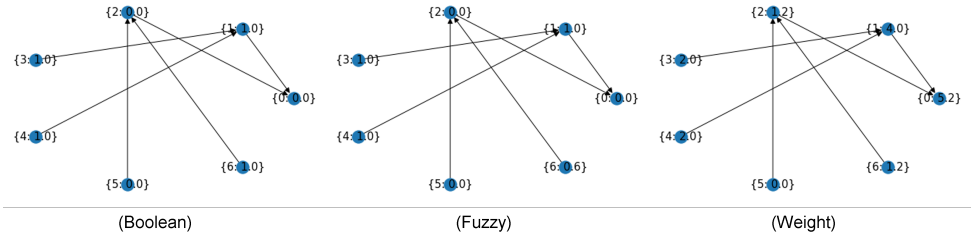

Fig. 8. Initial result from each scenario

satisfiability level at the goal 0. This is because at least one of the subgoal's satisfiability is not represented in the highest value that it can achieve (in this context, the value is 1). In particular, the fuzzy and weight cases possess two subgoals which are not fully satisfied. The actor A' seems to have a weakness at accomplishing the goal 5 for the boolean and goal 5 and 6 for the fuzzy and weight cases. We assume that there is another actor called A. The observed variable values of each subgoals by the actor A is shown in Table 5. In contrast, the actor A can achieve the goal 5 and 6 with higher satisfiability than the actor A' while the goal 3 and 4 cannot be accomplished. From this characteristic, if we assume that only goal can be swapped, then we expect the executor of goal 5 is handed over to the actor A' in all the cases. For the fuzzy and weight, the

Table 5. Observed variable values in each subgoal by actor A

	Observed variables			
	3	4	5	6
Boolean	0	0	1	1
Fuzzy	0	0	1	1
Weight	0	0	1	1

actor A' does not also fully satisfy the goal 6, but the degree of satisfiability is much closer to the highest than the goal5's and thus the goal5 is selected. By applying the algorithm 2, the result changes as Figure 9. The goal 5 is now

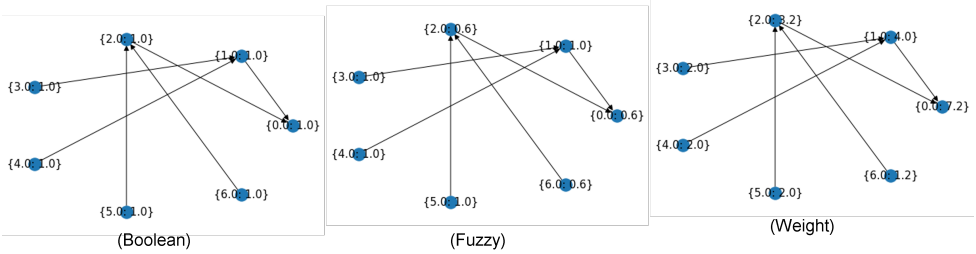

Fig. 9. Goal satisfiability after swapping

under the actor A's responsibility and this swapping raised the final satisfiability level of the root goal. The satisfiability level for each case becomes 1, 0.6 and 7.2 respectively, which matches with our expectation. From this experiment, the achievability of the Algorithm, 2 is validated.

6 Related Work

The versatility of the c-semiring allows the fusion of the concept with other fields. One example is argumentation formulation. [7] applies c-semiring based constraint check on Dung argumentation. The output of the constraint is treated as weights, which can be represented as either boolean, fuzzy or other expressions depending on the requirements. These values are used as the attributes of edges between nodes as weights. The weights are served as attack levels to the directed nodes. This idea is materialized by the development of Conarg [8] [9]. This Java-based software enables checking conflict levels in argumentation formulation. The ability of the proposed method is deepened by defending from attacks by comparing two attacks from the opposite directions [10].

Another application is security. [11] applies the essence of c-semiring to maintain desirable security degrees in multi-levels. One significant issue relating to security is the cascade elimination problems. The solution was considered an NP-hard problem. However, the generalization of this issue by using the c-semiring

approach reduces the difficulty into polynomial time level. The c-semiring approach also can be applied to the security protocol proposed by [12]. One substantial difference from the other approaches is that the c-semiring based approach allows to represent the protocol compliance level in a non-boolean form by allowing constraint check result to express into non-crisp values such as fuzzy theory.

The goal oriented concept has been used in various realms as well. For instance, [13] extends the concept of i* as a text form, which is required to guarantee the scalability of the concept in a large project. Equipping i* with the ability of expression in a text form enables applying the concept into a complex problem. These instances assure the extensibility of the i* concept.

i* can even be utilized into the field of law constraint checking systems. [14] develops a model called Nomos, which allows the i* modeling to assure that any tasks entailing law constraints comply with the rules. For instance, tasks relating to clinical treatments are often restricted with a number of rules and requirements. These rules need to be represented together with the goals as well. Nomos is connected with VLPM [15] to be able to display the law constraints in UML diagrams.

7 Conclusions

The coordination problem is complex, admitting inefficient and brittle solutions, in a variety of important application domains, including logistics, natural disaster management and UXV coordination. This paper addresses the problem of lifting the coordination to the goal-level, thus avoiding some of the drawbacks mentioned above. We offer a novel conception of cumulative goals, which are implicitly softgoals that permit explicitly accounting for the cumulative contributions of a number of individual actors. Cumulative goals are defined in an innovative mathematical framework (previously used for solving over-constrained problems) that is general enough to admit a variety of useful instances. We define algorithms for goal model maintenance and the optimal allocation of actors to goals. We present experimental results that suggest that the approach can be computationally realized. In future work, we expect to be able to report on broader array of coordination algorithms and on more practical implementations.

References

1. Yu, E.S.: Towards modelling and reasoning support for early-phase requirements engineering. In: Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on, IEEE (1997) 226–235
2. Fuxman, A., Kazhamiakin, R., Pistore, M., Roveri, M.: Formal tropos: language and semantics. University of Trento and IRST **55** (2003) 123
3. Giorgini, P., Kolp, M., Mylopoulos, J., Castro, J.: Tropos: a requirements-driven methodology for agent-oriented software. In: Agent-oriented methodologies. IGI Global (2005) 20–45

4. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3) (2004) 203–236
5. Dardenne, A.A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming **20**(1-2) (1993) 3–50
6. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. Journal of the ACM **44**(2) (1997) 201–236
7. Bistarelli, S., Santini, F.: A common computational framework for semiring-based argumentation systems. ECAI 2010: 19th European Conference on Artificial Intelligence **215** (2010)
8. Bistarelli, S., Santini, F.: Conarg: A constraint-based computational framework for argumentation systems. 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence (2011) 605–612
9. Bistarelli, S., Santini, F.: Modeling and solving afs with a constraint-based tool: Conarg. International Workshop on Theorie and Applications of Formal Argumentation (2011) 99–116
10. Bistarelli, S., Rossi, F., Santini, F.: A novel weighted defence and its relaxation in abstractargumentation. International Journal of Approximate Reasoning **92** (2018) 66–86
11. Foley, S.N., Bistarelli, S., O'Sullivan, B., Herbert, J., Swart, G.: Multilevel security and quality of protection. Quality of Protection (2006) 93–105
12. Bella, G., Bistarelli, S.: Soft constraint programming to analysing security protocols. Theory and Practice of Logic Programming **4**(5-6) (2004) 545–572
13. Penha, F., Lucena, M., Lucena, L., Angra, C., Alencar, F.: A proposed textual model for i-star. iStar (2016)
14. Siena, A., Perini, A., Susi, A., Mylopoulos, J.: A meta-model for modelling law-compliant requirements. 2009 Second International Workshop on Requirements Engineering and Law (2009)
15. Villafiorita, A., Weldemariam, K., Susi, A., Siena, A.: Modeling and analysis of laws using bpr and goal-oriented framework. 2010 Fourth International Conference on Digital Society (2010)