INTERACTIVE REASONING IN INTENTIONAL REQUIREMENTS ENGINEERING

by

Ali Akhavan Bitaghsir

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Interactive Reasoning in Intentional Requirements Engineering

Ali Akhavan Bitaghsir
Master of Science
Graduate Department of Computer Science
University of Toronto
2008

i^* and in general goal model evaluation is used to help a requirements analyst build a system that better achieves certain qualities by providing means of evaluating each alternative design. In this work, we improve the evaluation process of i^* by (i) providing new types of analysis and reasoning in the evaluation process and (ii) providing automated support for existing and the newly introduced analysis use case profiles. The proposed framework enables the requirements analyst to navigate through alternative solutions that when propagated would satisfy certain high level goals of the organization; also the analyst can specify preferences at different levels from basic desires to multiagent preferences and ask the framework to return the most preferred solutions first. The framework also produces deterministic partial solutions resulting from analyst's inputs by running cautious queries over the logical model corresponding to the goal model.

Contents

Chapter 1

Introduction

The concept of goal has been used in AI for planning to characterize an objective state of the world [11]. Software requirements can be viewed as the goals that the software is supposed to achieve and on this basis, goals have been used in software engineering to model early requirements [2] and non-functional requirements [8] in a system. As an example, for a hospital information management system, an early requirement might be to *make patients information available through the entire system for the staff that are authorized to access patient's data* while a non-functional requirement might be *to make the system as easy as possible to use for patients/personnel (user-friendly).*

Using goals for modeling stems from intentional modeling of systems. In a modeling framework an abstract conceptual model is constructed from the real system. Every modeling language focuses on certain characteristics and properties of the real-world domain that it is modeling. In [10], it is discussed that these characteristics (ontologies) can be categorized to static, dynamic, intentional and social. In static abstraction, the model captures entities and their attributes and the static relationship between them. The dynamic abstraction on the other hand can capture the changes occurred in system elements and can potentially model characteristics such as processes, states and state transitions. The classical software modeling diagrams used nowadays such as UMLs (unified modeling language) or ERDs (entity relation diagram) are examples of dynamic and static modeling frameworks respectively. The state-diagrams in UMLs enable them to capture different states of the system as well as possible transitions of the system among them. Although these models are nowadays addressing the majority of industry's demand for modeling systems, they don't go beyond describing the "what" and "how" of system aspects [15, 7]. For example, ERD diagrams describe *what* entities exist in the system along with *how* these entities relate to each other; or UMLs describe *what* states the system can go through and *how* would the system change its state (under *what* conditions or actions). The one aspect of systems that is not addressed within static or dynamic modeling is the reason behind the current functionality of the system or in other words the "why" questions: why do we have these states in the system? Why is the system designed to move from state A to B and not to C or can the system better satisfy that *reason* (goal) better if the system was changing its state in a different way? These valuable questions that can lead to constructing better systems are usually asked when designing large scale systems, however, static and dynamic modeling frameworks do not capture intentional concerns. A typical software architect for example deals with these questions on a daily basis. The "why"s or motivations of the systems arise from the autonomous parties involved in the system: people and the organizations involved in the system. In the literature, *goal models* are typically used for modeling intentional aspects of systems [14].

Goal models represent involving agents' goals with explicit modeling notations and provide modeling support for refining goals into more specific subgoals and eventually tasks that are technical representation of the system. Goals can potentially be decomposed and refined in more than one way to subgoals; this variety determines the space of possible solutions to a high level goal. However, an *evaluation* of each solution (*alternative*) can determine to what extent can each solution meet the goal. The other kind of modeling is the *social* modeling in which the organizational structure and the interaction of agents' goals, roles and positions can be also modeled using proper modeling notations. The i^* modeling framework [16] is capable of capturing all the four category of concerns in systems; The organizational interactions are modeled through dependency links in a network of goal models that address the intentional aspect of the system. i^* also model limited static and dynamic concerns through such constructs like resources (as entities) and tasks (as processes). The i^* model and in general organizational goal models provide a form of abstraction that enable simple yet effective forms of evaluations and analysis to be performed on social/intentional aspects of organizations. These analysis can lead to fundamental improvements in the strategic structure of the organizations and further satisfying organizational goals. [7] provides an evaluation support framework for i^* that accounts for the social modeling capability of i^* as well as intentional.

In this work, we intend to improve the evaluation process of i^* by (i) providing new types of analysis in the evaluation process and (ii) providing automated support for existing and the newly introduced analysis profiles. In the following, we provide a review of the existing approaches for evaluating goals models and also specifically i^* models and discuss their limitations.

1.1 Existing Evaluation Frameworks

From a user-centric point of view, there are certain *use cases* that when supported by an evaluation software tool, can help the analyst in improving his/her understanding of the goal/i^* model (henceforth called goal model for briefness) and available alternative solutions. We can view these use cases under two categories:

1. **Structural Use Cases:** the ones that deal with the construction of the goal model such as modifying the goal model like adding or removing elements, importing and exporting the goal model, etc; and

2. **Evaluational Use Cases:** the use cases that help the analyst evaluate, compare, modify and find the best alternative solution for fulfilling organizational goals with respect to organizational *soft goals* and the way they will be fulfilled, such as bottom up reasoning, top-down reasoning, etc.

Our framework extends the current evaluational use cases of i^* evaluation procedure (which is for example supported by OpenOME [17]) and provides automation support for them. In chapter 4 we describe a line of research as an extension of this paper in which automation support is provided even for structural use cases.

Throughout the literature, we can observe the following major approaches to *evaluational* use cases in evaluating goals models:

- **Bottom-Up Reasoning:** In this scenario, given a goal model, the user can specify a set of initial labels for certain elements of the model and use the tool to propagate the labels up to the high level goals using a set of propagation rules that depend upon the evaluation framework in use. For example, in an i^* evaluation procedure described in [7], the propagation rules are driven from table 3.1. The analyst can, for example, use this use case to monitor the effect of changing the way a soft goal is fulfilled on the satisfaction of the high level goals of the organizational model. The *flow* of reasoning in this case is bottom up. In other words the user can not specify desired high level satisfaction values and derive low level elements' labels.

- **Top down Reasoning Using SAT Solver:** Automated reasoning can be most helpful when the user can specify desired labels and use the reasoning tool to derive other labels based on the model structure: as the size of the model grows, the number of ways that high level goals can be fulfilled increase exponentially with regard to the choices available. Top down reasoning with SAT Solver [12] was an effort to address this kind of reasoning. Given as input a set of desired values D for high level goals, the reasoning mechanism can determine either there exists a solution alternative (an assignment of satisfaction/denial labels to leaf level elements in the model) that when propagated, can produce every label in D.

Limitations of existing approaches

Navigation through solutions In the discussed approaches, the user can't navigate through different plans, but rather can only receive 0 or 1 solution. It is obvious that giving the user the option of navigating through solutions enables the user to choose among more options and better tailor the solution to his needs that are not necessarily expressed in the criteria given as input to the evaluation software.

Preferences among user's objectives Moreover, the user can't specify preferences among the labels in D: for example it may not be possible to fulfill both goals $G1$ and $G2$ at the same time, however, there may exist solutions that address one of them. In this case, the user could have specified a preference between the two goals. The preferences can of course take more complex forms and come into play in cases where organizational goals highly compete with each other and compromising some of them in favor of the others is inevitable. Also, the number of possible alternative solutions for user's objective can grow exponentially with the size of the model and thus the user can not always enumerate them all in order to arrive at a conclusion. This concern too calls for having a mechanism for discriminating among the solution alternatives.

Inferring implicated properties of the model The other direction along which the existing approaches can be extended, is the ability of inferring useful properties of the model based on the current solution and the structure of the model. For example, when the user assigns a certain label to an element, given the user's intentions, or even in some cases independent of that, some other elements' label will be uniquely determined. Identifying these consequences can be helpful to the user since the user will know that no matter what alternative he chooses, the value of these elements should be as they are determined by the reasoning framework.

Incorporating domain knowledge into reasoning The other limitation of the existing reasoning and evaluation approaches is the way they deal with the analyst's domain knowledge when propagating labels: when analyzing goal models, it's often the case that the contributions made towards an element are conflicting each other. Some of the contributions imply the satisfaction of the element while others imply its denial. In i^* evaluation procedure, in order to deal with these contradictions, the human knowledge is taken as input and will be accounted to unify and resolve the conflicting contributions into a satisfaction or a denial label for the element. The automated propagation will be then continued until another conflict arises that needs human input to be resolved and this process repeats until the model's labels converge. Whereas in other evaluation frameworks that use reasoning such as [5], the assumption is that the initial labels along with the structure of the goal model is enough to derive the ultimate labels; when a conflict arises, they either reject the current solution or view the a *conflicted* element as one which is both satisfied and denied and then the satisfaction and denial of this element will be *separately* propagated upwards. We think that both approaches are approximations of the reality. We believe that because of the complexity and diversity of the real world organizations, human interaction is necessary to make requirements analysis a realistic one that can take into account the unique characteristics of that organization.

1.2 Approach

The proposed framework in this paper intends to address the above shortcomings. Navigation through proper solutions is provided by extracting all the solutions and providing them to the user. The underlying computational engine along with provided logic rules in chapter 3 makes it possible to discriminate among solutions based on user preferences. By running queries over the program that represents the goal model in logic rules, we can find the implicated properties of the model from the user's initial input. The framework also effectively solicits human knowledge into logical reasoning through user interaction through two approaches: **(i)** Individually translating human conflict resolution into corresponding rules that simulate the effect of manual resolution in automated reasoning and **(ii)** providing the user with the option of specifying preferences among incoming links to a node N and resolving conflicts arising in N exclusively based on the given order.

On the other hand, the framework takes advantage of faster logical reasoning machinery compared to other computational methodologies: Answer Set Programming *(ASP)* engines. The specific ASP engine that we use, $\mathcal{DLV}$[9], uses **model checking** to compute models, which is much faster than current implementation of i^* evaluation procedure that uses regular propagation algorithms.

The provided implementation is almost completely independent of the way that the underlying engine works and on the other hand, recent advances [6] have shown that the models of an answer set program can be computed using SAT solvers which offer even more potential prospect for faster computation.

1.3 Reasoning on Goal Models with ASP

The proposed logic framework consists of rules that together unify the following pieces of knowledge into a declarative logic program:

- **The structure of the goal model:** the elements, decomposition, contribution and dependency links in the model.
- **User's initial labels** initial labels that user has assigned to leaf level elements.
- **User's desired labels for high level elements:** or in other words, user's goals as to which soft goals or goals need to be satisfied.
- **User's preference for the satisfaction of soft goals:** which soft goals are more important than others, in case all of them can not be satisfied.
- **User's knowledge of conflict resolution:** which is given as input by the user through interactions or other mechanisms.
- **Structural constraints and Plug-in qualities:** The expressive power of the underlying logical language that is used opens windows of opportunity for expressing a wide range of constraints and qualities of the model. These qualities may be context sensitive and thus can be plugged into the set of other rules to produce models of higher quality. For example, one could prefer models that *delegate* as few tasks as possible to external agents over others. Other forms of constraints could enforce levels of trust in the model. One good example of such constraints which have has implemented using declarative logical language is [3].

The framework then feed the resulting program P into an ASP engine. The ASP engine returns answer sets which are minimal sets of ground atoms that conform to all rules of P that correspond to evaluations of the goal model. Through appropriate user controls, the user can navigate through the generated models, pick up a model, modify it, resolve any conflict in that model, re-propagate the results, specify preferences among soft goals and other related tasks that enable the user to interact with the reasoner.

1.4 Paper Organization

In the next chapter, through a top-down approach, we present the *evaluational use cases* that are made possible in our framework. In chapter 3 computational support is provided for each of the introduced use cases using answer set programming. Finally, in chapter 4 we provide the future lines of research that stems from this research as well as conclusions.

Chapter 2

Interactive evaluation framework for i^*

We present our framework in two chapters: this chapter presents the conceptual design of the framework from a user perspective while the next chapter deals with computational support for use-cases introduced in this chapter. In addition to the verbal description of the use cases, a visual snapshot of the application controls that the user can potentially use in OpenOME to go through each use case is also given.

2.1 Bottom-Up Propagation Use Case

Bottom-Up propagation for i^* evaluation is supported in available evaluation procedures such as [7] and implemented in tools such as OpenOME. The main contribution in this part would be the translation of the use case into answer set programming rules in order to semi-automate the use case (automated when there's no need for human conflict resolution and manual otherwise) and integrate this use case with the other ones that our framework supports. While the original implementation of this use case has appeared in [7], we need to incorporate it in our framework since the propagation logic is used in other use cases such as top-down evaluation.

In bottom-up propagation scenario, the user will assign a set of *initial* labels to some nodes in the i^* model and asks the tool to perform a bottom-up propagation. The tool would then propagate the labels based on the propagation algorithm given in [7] and once the propagation converges, shows the result to the user. The other dimension to this use case is conflict resolution. In the presence of conflicting labels in a node's bag of labels, the tool asks the human to resolve the bag of labels to a final label and resumes the propagation procedure afterwards.

For visualizing this use case, consider the i^* diagram in figure 2.1. As can be seen in the figure, the user has pre-assigned a set of initial labels to some of the elements in the lower level of the model. The OpenOME tool has simple button/menu controls for performing a bottom up propagation (see figure 2.2). Once the user presses this button, the bottom up propagation is performed and the result is shown to the user in the form of newly propagated labels in the diagram (see figure 2.3).

When the i^* evaluation procedure is unable to assess a final label based on the current labels in a node's bag of labels, it will ask the user to provide a resolution. The input dialog box shown in figure 2.4 shows this scenario. The user will be repeatedly asked for conflict resolution until the propagation converges into a final state where labels don't change anymore as a result of iteration [7].

The computational support needed to automate this use case is provided in the following chapter.

2.2 Top-down Evaluation Use Case

Top-down evaluation can enormously help a model analyst, given that the model reflects the reality as close as possible and also that the model *includes* at least a near-optimum solution among its alternatives.

In this use case, the user can determine a set of objective labels to be satisfied in the model and ask the software tool to provide solution alternatives that satisfy the user objectives. Currently, OpenOME has a SAT-based implementation of the top down evaluation [4], however, as pointed out in chapter 1 it only provides one possible solution, among potentially many. Our framework produces all the valid alternatives to the user by means of AI planning and thus need some controls to let the user navigate through these alternatives. The UI would provide two simple buttons for going back and forth among the generated alternatives which are shown in figure 2.5. The user first presses the top-down arrow button and then can go backward and forward through the generated alternatives with the provided left and right arrow buttons (see figure 2.5).

Example Consider the i^* model given in figure 2.6. As can be seen in the model, the objective of the user is to set the label of goal a to be fully satisfied(`fs`). As such, based on i^* propagation rules, since goal a is decomposed into sub goals b and c, the label of these sub goals should be at least `fs` which implies the label of `fs` for both goals b and c. Moreover the label of nodes g and f is uniquely determined as partially denied and

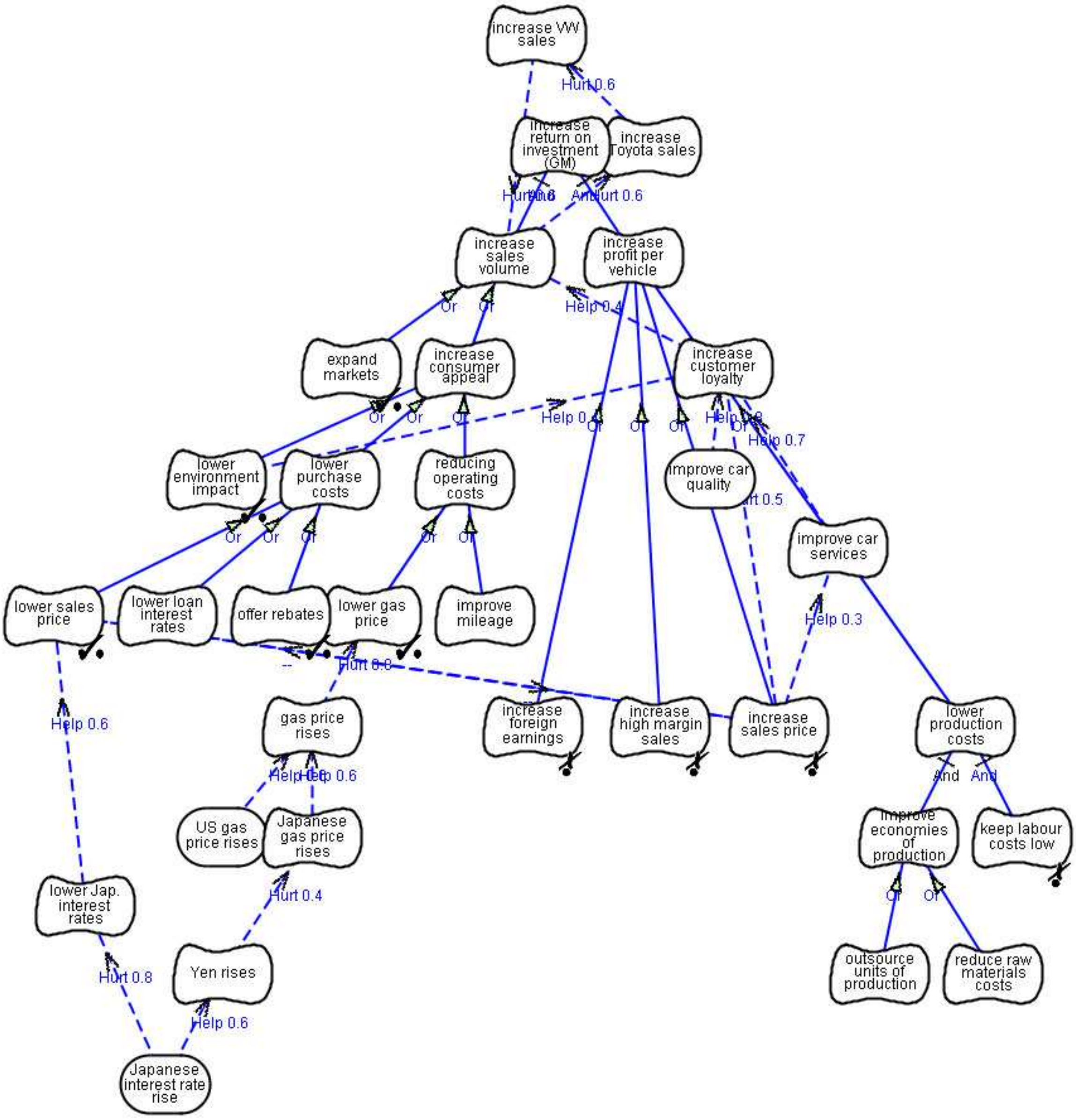


Figure 2.1: Snapshot of a sample goal model which is appeared in OpenOME example models on which the user performs Bottom-Up propagation. The initial labels are placed at the bottom right of nodes.

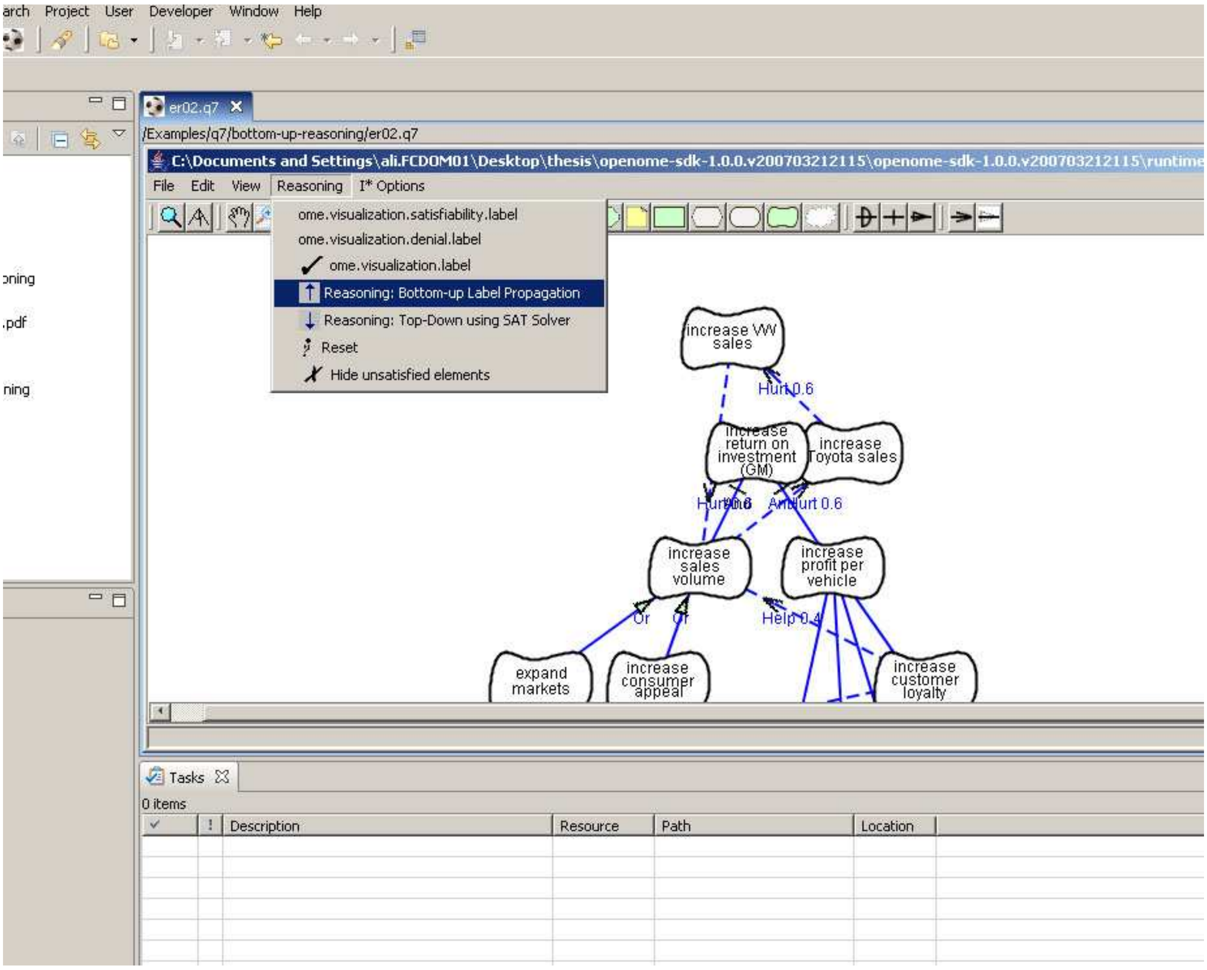

arch Project User Developer Window Help
er02.q7
/Examples/q7/bottom-up-reasoning/er02.q7
C:\Documents and Settings\ali.FCDOM01\Desktop\thesis\openome-sdk-1.0.0.v200703212115\openome-sdk-1.0.0.v200703212115\runtime
File Edit View Reasoning I* Options
ome.visualization.satisfiability.label
ome.visualization.denial.label
ome.visualization.label
Reasoning: Bottom-up Label Propagation
Reasoning: Top-Down using SAT Solver
Reset
Hide unsatisfied elements
oning
.pdf
ning
increase VW sales
Hurt 0.6
increase return on investment (GM)
increase Toyota sales
Hurt 0.6
And
Hurt 0.6
increase sales volume
increase profit per vehicle
Or
Or
Help 0.4
expand markets
increase consumer appeal
increase customer loyalty
Tasks
0 items
Description
Resource
Path
Location

Figure 2.2: The bottom-up propagation control in OpenOME

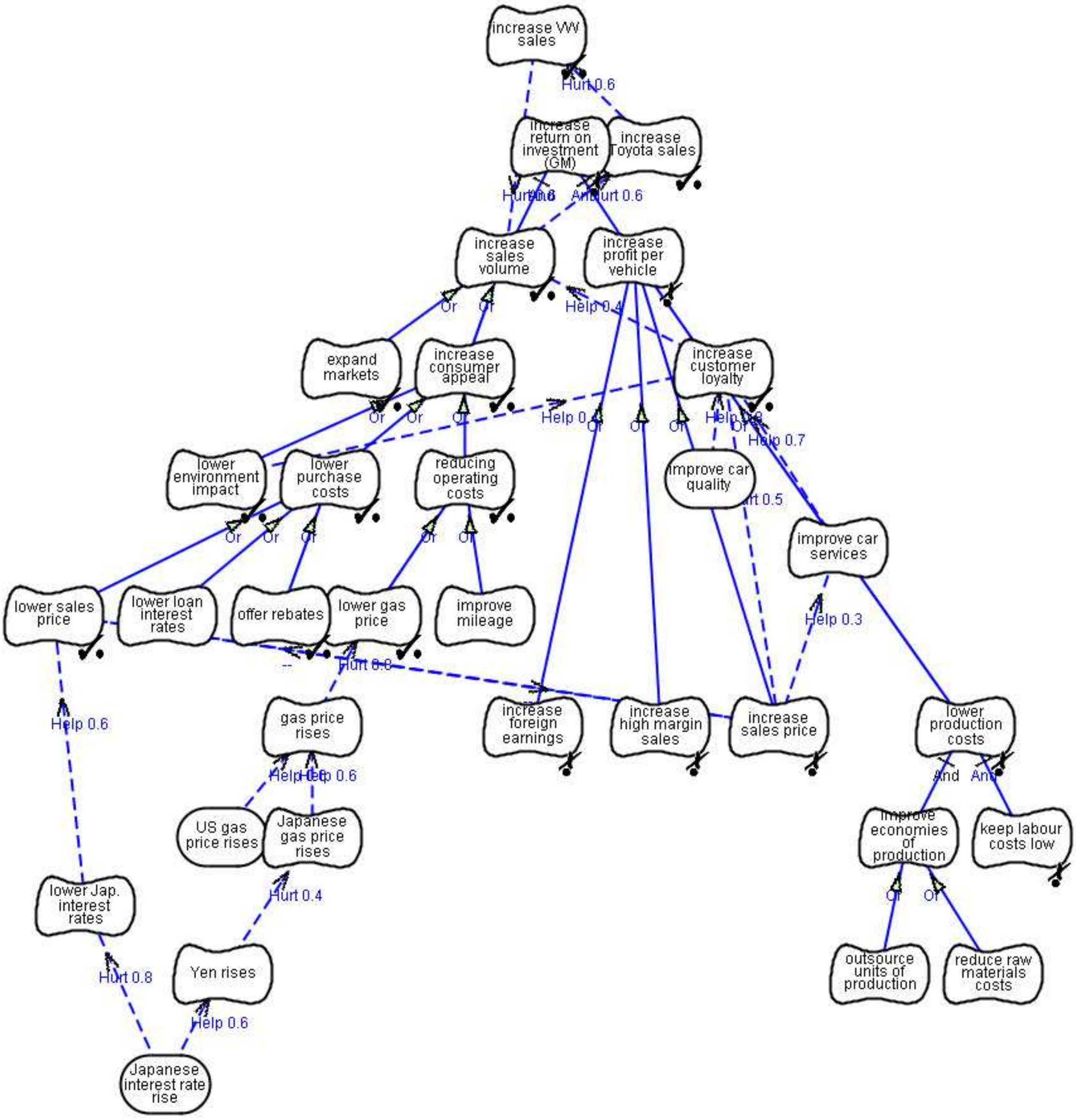
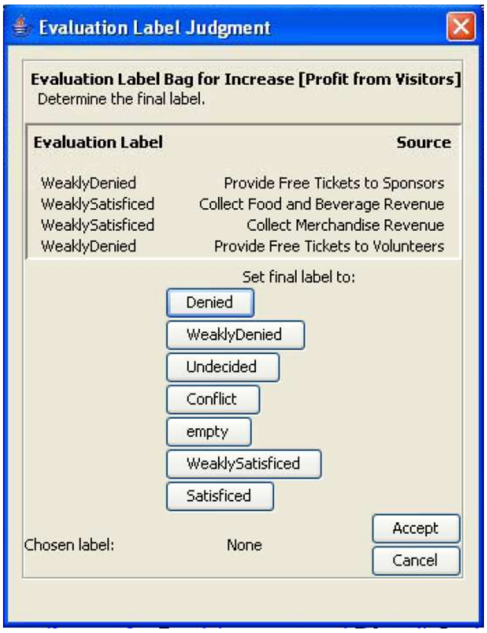


Figure 2.3: The result of bottom up propagation on the i^* model given in figure 2.1



Evaluation Label Bag for Increase [Profit from Visitors]
Determine the final label.

Evaluation Label	Source
WeaklyDenied	Provide Free Tickets to Sponsors
WeaklySatisficed	Collect Food and Beverage Revenue
WeaklySatisficed	Collect Merchandise Revenue
WeaklyDenied	Provide Free Tickets to Volunteers

Set final label to:

Chosen label: None

Figure 2.4: The dialog that enables the user to resolve the conflicts.

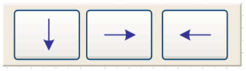


Figure 2.5: The top-down, left and right arrows will provide the user the ability to perform top-down evaluation as well as navigating through the generated alternative solutions.

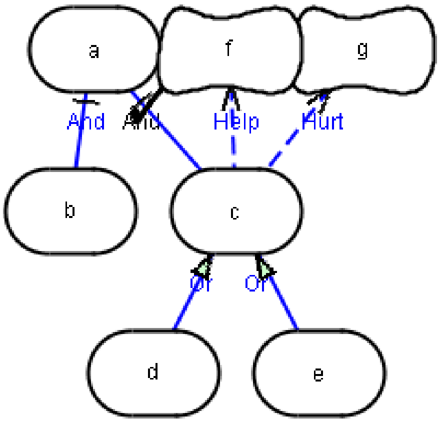


Figure 2.6: The initial i^* model on which the user intends to perform a top down evaluation.

partially satisfied; however for the subgoals of goal c we have several options: only one of d or e needs to be necessarily fully satisfied. The two most significant alternative solutions are the two solutions in which either d or e is satisfied. The user can navigate to these two solutions as well as other valid solutions by the left and right arrow buttons. One of the outcomes of the top-down evaluation is shown in figure 2.7. If the user navigates through the possible alternatives, among the other alternative solutions, The user is eventually provided with the one alternative depicted in figure 2.8.

2.3 Partial Solutions Resulting from User inputs

As the user specifies his desired labels as well as initial labels for the goal model, some of the labels in the model can be determined as implications of the user's input; in other words, the evaluation rules along with user's desired labels can impose certain labels in the graph.

We provide the user with this feature in both explicit and implicit ways: Once the user has provided a sequence of inputs, she can explicitly ask the tool whether the label of a specific element or a set of elements is uniquely determined with respect to users' inputs and objectives. Also, the tool can implicitly inform the user about the implications of her actions (inputs/objectives) by examining the the neighborhood (technically by running a set of queries [see chapter 3]) of the elements that user is modifying. This use case is available to the user in top-down evaluation scenario. Chapter 3 provides user interface

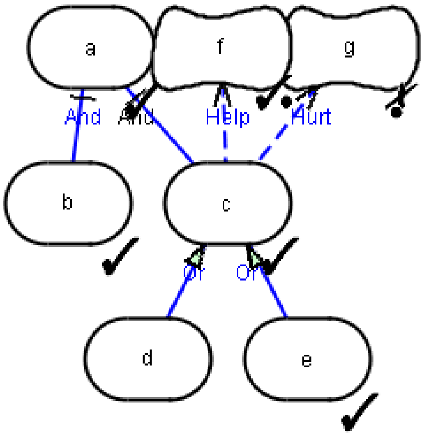


Figure 2.7: A generated alternative that satisfies user objectives given in figure 2.6.

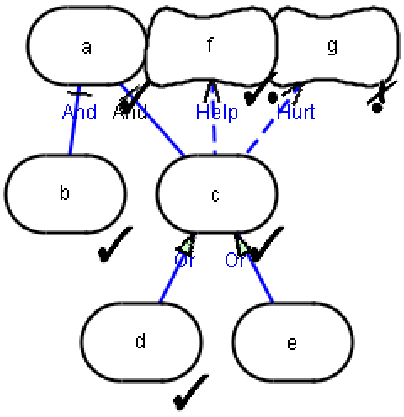


Figure 2.8: One of the solution alternatives that satisfies user objectives given in figure 2.6.

snapshots for this use case. There's no specific UI control needed for this use case, however, the user can potentially determine the diameter of the neighborhood on which the application will run the queries when the user changes the model.

2.4 Preferential Use Case

We address this case by providing the user with the ability to define soft-constraints (as opposed to hard-objectives) in the form of *preferences* to prioritize the solution alternatives. Based on recent advances in planning and logic literature [13], we address this feature by enabling the analyst to define preference among qualities of alternatives. In i^* the goals that doesn't have a clear cut definition - in terms of i^* constructs and relationships - are represented as soft goals. Therefore, discriminating alternatives based on their qualities can be interpreted as discriminating alternatives based upon the soft goals corresponding to those qualities (non-functional requirements). Although we ground our approach in preferences among soft-goals, it can be easily extended to a framework that considers preferences among hard goals (that correspond to functional requirements) as well.

In the following paragraphs, we provide different forms of preference notations and semantics that we incorporate in our framework. In chapter 3 we'll express this semantics in answer set logic programming;

1. In the simplest form, we'd like to be able to express a simple desire among an agent's (soft) goals. E.g., formula 2.1 indicates the desire of full satisfaction of *usability* and *privacy* soft goals and partial denial of *flexibility*.

   $$\varphi = label(usability, fs) \wedge label(privacy, fs) \wedge label(flexibility, pd) \tag{2.1}$$

   The `label` predicate is the standard that we use in our ASP encoding which is presented in more detail throughout the next chapter.

2. Due to lack of resources, the user's most preferred desire can't necessarily be satisfied by any alternative; therefore, the preference language needs to be able to incorporate several desire formulas and somehow express a priority among them. As an example, the preference formula shown in formula 2.2 indicates that the user's most preferred desire is to have both the *usability* and *privacy* soft goals be fully satisfied, and moreover it shows that if full satisfaction of both of these qualities is not possible, the user prefers to have the soft goal *privacy* to be satisfied

over *usability*.

$$\begin{aligned}\psi = label(usability, fs) \wedge label(privacy, fs) &\quad \lhd \\ label(usability, fd) \wedge label(privacy, fs) &\quad \lhd \\ label(usability, fs) \wedge label(privacy, fd) &\quad \lhd \\ label(usability, fd) \wedge label(privacy, fd) &\end{aligned} \tag{2.2}$$

3. In the presence of multiple agents in the model with individual preferences among desires for each agent, the preferences of each agent may potentially compete with each other. If the agents involved in the model are of equal importance to us, we won't prioritize their individual preferences; however, when agents are different to us in terms of importance, then we would like to favor a solution that favors more important agents' preferences over less important agents'. As an example, suppose that the preference formula for agents A, B, C and D is $\psi_A, \psi_B, \psi_C, \psi_D$ respectively and moreover that agents A and B's preferences are both equally more important than agents C and D's preferences and also that agents C and D's preference are of equal importance to us. In this case, equation 2.3 captures our description.

$$\psi_A || \psi_B \lhd \psi_C || \psi_D \tag{2.3}$$

In the rest of this section, we attempt to provide a user interface as well as an example scenario that can better depict the feature provided in this use case. The UI controls provided in this section enable the user to define a subset of the different types of preferences that we define in the computational section; specifically, the proposed UI supports defining partial ordering among the goals of an agent as well as defining partial ordering among agents' *single agent* preferences (see chapter 3 for definition). Also, we assume that the intention of the user is always to *satisfy* a goal as opposed to denying it (which is supported computationally).

In order to enable the user to express a order among soft goals of an agent or among agents themselves, we have to provide the user with the ability to define an ordering over a set of objects. A typical user interface control for this purpose would be an *ordered list*: the user initially populates the list with a set of soft goals or agents (selecting objects of different types is of course supported by OpenOME) and then using **move up** and **move down** control buttons, any object in the list can be selected and moved to the top of the list or to the bottom. The top most object will have the higher priority in planning.

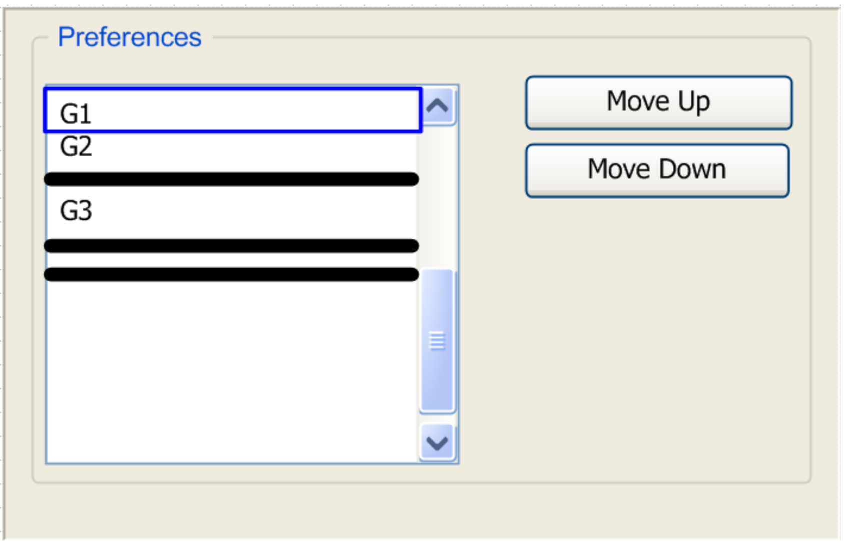


Figure 2.9: The user interface control by which the user can define an ordering among a set of objects (in this figure the objects are specifically goals). $G1$ and $G2$ are in priority level 1 (highest) while $G3$ is in the second priority group.

The user can move the bars up and down to change the formation of the priority groups.

$$label(G1, fs) \vee label(G1, ps) | label(G2, fs) \vee label(G2, ps) \quad \lhd \\ \ldots label(G3, fs) \vee label(G3, ps)$$

Figure 2.9 depicts the UI control for fictitious goals $G1$, $G2$ and $G3$. More formally, the following items describe the way that these preferences are interpreted:

1. **Ordering among soft goals of an agent**: Assume that the user has specified an order among soft goals $G_1, G_2, \ldots, G_n$. If the ordering is a total ordering, we interprete this input such that alternative α would be more preferred that alternative solution β iff there exist a goal G_i that is satisfied or partially satisfied in α while it is not satisfied or partially satisfied in β and moreover, for every goal G_j that has a higher rank in user's prioritization, G_j is either satisfied in both α and β, partially satisfied in both of them or not satisfied at all in both of them. If the ordering is a partial ordering the semantics in chapter 3 imply a more complicated verbal description (see chapter 3).

   In terms of the preference language that is presented in the next chapter, the interpretation of the preference specified in figure 2.9 amounts to having the following preference formula:

   Generally, assuming that the goals in priority group i are represented as $G_{i_1}, G_{i_2}, \ldots, G_{i_{k_i}}$, the following formula would indicate the user's preference:

$$\begin{array}{ll} label(G_{1_1}, fs) \vee label(G_{1_1}, ps)| \ldots |label(G_{1_{k_1}}, fs) \vee label(G_{1_{k_1}}, ps) & \lhd \\ label(G_{2_1}, fs) \vee label(G_{2_1}, ps)| \ldots |label(G_{2_{k_2}}, fs) \vee label(G_{2_{k_2}}, ps) & \lhd \\ \ldots & \\ label(G_{n_1}, fs) \vee label(G_{n_1}, ps)| \ldots |label(G_{n_{k_n}}, fs) \vee label(G_{n_{k_n}}, ps) & \lhd \end{array}$$

$$\psi_{A_1} \lhd \psi_{A_2} \lhd \ldots \psi_{A_N}$$

2. **Ordering among singe agent preferences**: Assuming that the user has specified an ordering among agents' single agent preference formulae as $A_1 > A_2 > \ldots > A_n$, given the semantics of the preference language, a multi agent preference formula will be generated as follows:

   where ψ_{A_i} represents the single agent preference formula that is generated for agent A_i based on the interpretation given earlier. For partial ordering, the same pattern as the one for goals would be used for translating the user's intention to our framework's preference language.

Considering the example i^* model in figure 2.1 is the model of a system containing one agent A and this model is the goal model of that one agent and also that the user has specified the provided a total ordering among the goals as : $d \lhd e$, then the tool would *first* return the most preferred solution alternatives which according to our framework's interpretation should amount to returning the goals model given in the figure shown in table 2.1 first and depending on either the user wants to navigate to more alternatives, all the other alternatives subsequently.

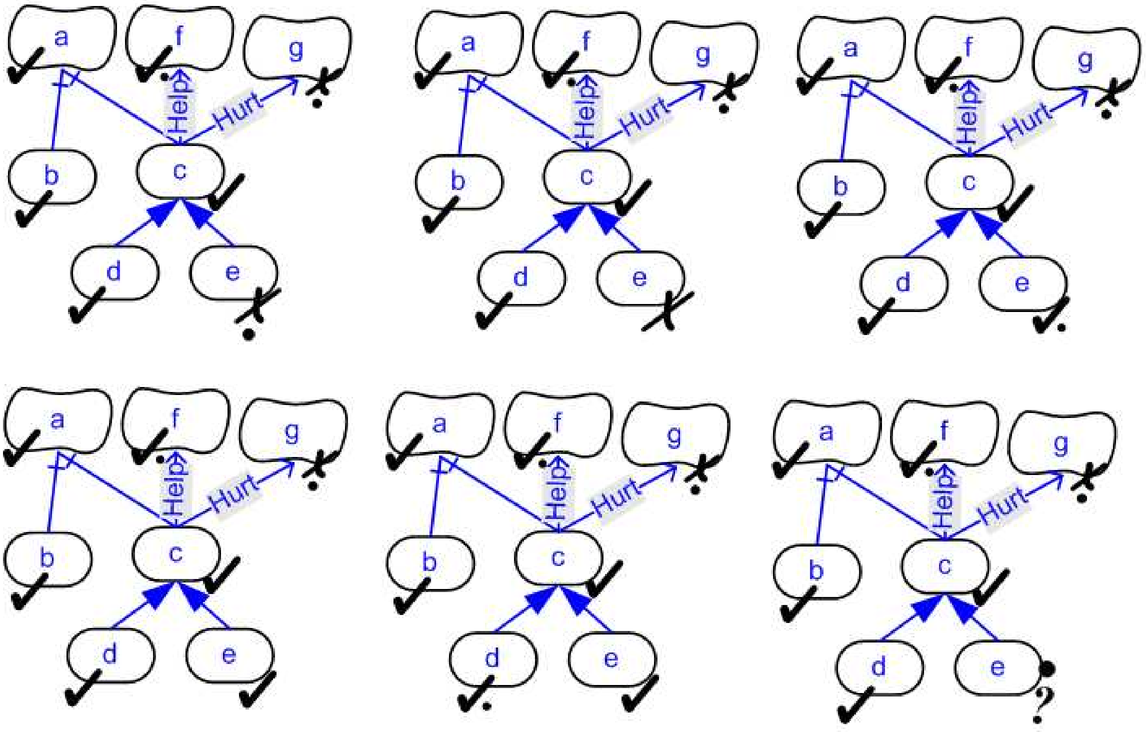


Table 2.1: The most preferred alternative solutions for user's objective specified in figure 2.1 with respect to user's preference which is $d \lhd e$

Chapter 3

Computational Support for i^* Evaluation/Reasoning framework

In order to support the use cases visited in chapter 2, in this chapter, computational machinery is provided for them in the order they are visited in chapter 2. As discussed in section 1.3, the general approach is to translate the problem constraints, objectives and the model structure into a unified logic program and feed this program into an ASP engine in order to get answer sets that correspond to the solution alternatives in which the user is interested. In the first section, we provide some foundational predicates of the program that is shared among all use cases. The following sections lay out the translation of the problem into logic program rules per each use case and in the end of each section an example is provided to demonstrate the ideas concretely. The key point here is for us to be able to express the constraints and rules that we have in terms of logic program rules; Computing answers that follow these rules is delegated to the ASP solver engine.

Our framework provides the translation engine for the use cases discussed throughout chapter 2 which is responsible for translating the problem into a logic program whose answer sets provide the solutions for the use case. In future work, we intend to fully integrate this engine with OpenOME using the UI prototypes provided in chapter 2.

The rules produced by the translation engine is provided as input to $\mathcal{DLV}$. $\mathcal{DLV}$[9] is a deductive database system based on disjunctive logic programming that finds all minimal sets of ground atoms that follow the rules in a given program. Answer set programming is usually used to solve combinatorially hard problems: once should specify the problem search space and also problem constraints in terms of logic rules; The ASP engine then returns all minimal solutions (containing minimal number of atoms) for the logic program. An advantage of using the $\mathcal{DLV}$ framework is that we can have disjunctive expressions in

Description	**Predicate**
Goal	`goal`
Soft Goal	`softGoal`
Task	`task`
Label	`label`
element with incoming contribution links	`contributedTo`

Table 3.1: The i^* element types in our framework's scope

Relationship Type	**Predicate**
And	and(P, C)
Or	or(P, C)
Help	help(P, C)
Hurt	hurt(P, C)
Make	make(P, C)
Break	break(P, C)
Dependency	depend(P, C)

Table 3.2: The i^* link types in the scope of our work; P, C indicate the parent and child node respectively.

the head of rules and that's the key for us to be able to express the space of possibilities (see the top-down evaluation section in this chapter).

3.1 Foundational Rule Groups

3.1.1 Structural rules

The first group of rules are responsible for representing the basic structure of i^* models. For each given type of element in i^* [16], we introduce a predicate that holds for all instances of that type in a given goal model. So, if p is an element's name in i^*, say the name of a goal or of a task, we'll have the predicate $Type(p)$ where $Type$ is the type of p. The types that we included in our implementation are listed in table 3.1.

The other set of structural rules are the ones that define the relationships among the elements. The relationships that we deal with in our framework are listed in table 3.2.

The structural rules presented will be used when translating supported use cases in this chapter to logic rules in order to reflect the structure of the model.

Label	Ground atom
Fully Satisfied	`fs`
Fully Denied	`fd`
Partially Satisfied	`ps`
Partially Denied	`pd`
Conflict	`cf`
Unknown	`un`

Table 3.3: The i^* relation types in the scope of our framework

3.1.2 Evaluation Rules

Labels

i^* evaluation procedure [7] is an enhanced version of an evaluation procedure originally proposed for NFR framework in [8]. A range of labels are used to indicate the extent to which an element in the model is satisfied or denied. In our logical formulation, we have a unary predicate for each label which holds for the elements with that label. Also an element may not have more than one label at a time. Table 3.3 represents the different labels and their corresponding ground atoms in our framework.

So if for example element e is fully satisfied in an evaluation E, `label(e, fs))` would hold in $L(E)$: the logical model that corresponds to E and the underlying goal model.

3.2 Bottom-Up Propagation

For bottom-up evaluation, we adopt the propagation mechanism in [7] and translate it into our framework by reusing our other predicates for labels and other properties of the model. Based on [7], the propagation flows through the links and determines the label of high level elements based upon their relationship with other elements in the graph which is defined by various types of links in the model.

3.2.1 Dependency Links

In i^* evaluation, the evaluation value is directly transferred from dependee to dependum (the element depended upon) to depender. A tool based on the framework would automatically generate the rules based on the following rule scheme for each dependency link: For each `depender` element that depends on `dependee` element upon `dependum`, the

```
index(fd, -3).
index(pd, -2).
index(un, -1).
index(cf, 0).
index(ps, 1).
index(fs, 2).
```

Table 3.4: Rules that assign indices to i^* labels. Note that the specific choice of numbers as indices is not material: only the order of the numbers should be consistent with the order given in equation 3.2.

following rule is added:

`label(depender, L) :- label(dependee, L), label(L).` (3.1)

Please note that syntactically, if a name begins with upper-case letter, it is interpreted as a variable. On the other hand, predicates start with lower-case letters. Also, a predicate with no parameter is considered to be a ground atom.

3.2.2 Decomposition Links

Decomposition links in i^* determine the required elements that should be satisfied in order for a *task* to be satisfied. The definition naturally amounts to and AND semantic for the decomposition links when it comes to evaluation. There's however a degree to which each of the required elements can be satisfied and the evaluation picks up the minimum label (with regard to the following ordering) from the *children*. The ordering of i^* labels is as follows:

`fd < pd < un < cf < ps < fs` (3.2)

The same ordering is used in means-ends links (paragraph 3.2.3). For implementation simplicity, The above ordering can be expressed by assigning an integer index to each label and comparing the indices when we need to compare labels.

For representing the semantic of decomposition links, the framework follows the following scheme: Suppose that task p is decomposed to elements $c_1, c_2, \ldots, c_n$; the following rules are consequently included in the logical model:

The rules will select the *minimum* labels among p's children as its label.

```
bagContains(p, L) :- label(c1, L).
bagContains(p, L) :- label(c2, L).
...
bagContains(p, L) :- label(cn, L).
label(p, Y) :- bagContains(p, Y), not bagContainLessThan(p, Y).
bagContainsLessThan(E, X) :- bagContains(E, Y), Y < X.
```

Table 3.5: Rule scheme that represents decomposition links propagation in i^*. Note that we assume that parent node p is decomposed into elements $c_1, c_2, \ldots, c_n$

```
bagContains(p, L) :- label(c1, L).
bagContains(p, L) :- label(c2, L).
...
bagContains(p, L) :- label(cn, L).
label(p, Y) :- bagContains(p, Y), not bagContainMoreThan(p, Y).
bagContainsMoreThan(E, X) :- bagContains(E, Y), Y > X.
```

Table 3.6: Corresponding logical rules for means-ends links

3.2.3 Means-ends Links

The means-ends links are devised to model the different tasks t_i that each can satisfy a goal g. In other words, one or more alternative tasks can accomplish a goal and consequently the evaluation label of g would be the *maximum* label among these alternatives; the maximum will be computed with regard to the same ordering given in equation 3.2. In this case, we will include the rules in table 3.6 in the logical dual model, assuming that elements $c_1, c_2, \ldots, c_n$ are the means-ends for element p:

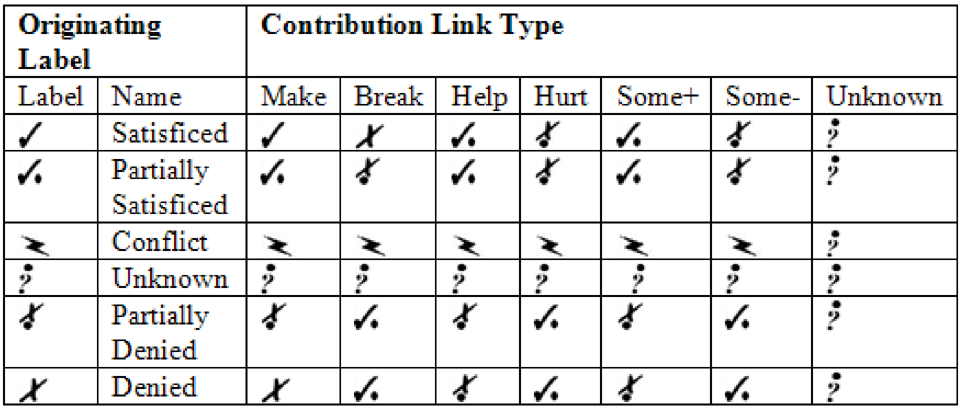
3.2.4 Contribution Links

The evaluation of contribution links in i^* is done according to table 3.1 taken from [7].

We divide the necessary rules that follow table 3.1 for propagation into two sets: `(i)` the set of rules that populate the parent's bag based on child's label and the type of the contribution link that goes between the child and the parent node and `(ii)` the set of rules that resolve the labels in the parent node (the node that has incoming contribution links).

For each entry in table 3.1 we include one rule from the first set that adds a label to the parent node's bag of labels. For example, in the case of a *help* contribution link, we'll have the following rules that represent the column in table 3.1 that corresponds to help links:

Note that the predicate `contributedTo` is used to increase the performance: only



Originating Label		**Contribution Link Type**						
Label	Name	Make	Break	Help	Hurt	Some+	Some-	Unknown
✓	Satisficed	✓	✗	✓•	✗•	✓•	✗•	?
✓•	Partially Satisficed	✓•	✗•	✓•	✗•	✓•	✗•	?
⚡	Conflict	⚡	⚡	⚡	⚡	⚡	⚡	?
?	Unknown	?	?	?	?	?	?	?
✗•	Partially Denied	✗•	✓•	✗•	✓•	✗•	✓•	?
✗	Denied	✗	✓•	✗•	✓•	✗•	✓•	?

Figure 3.1: Propagation Rules Showing Resulting Labels for Contribution Links

rule for the first row (source label is `fs`):
`bagContains(P, ps) :-`
`contributedTo(P), help(P, C), hasLabel(C, fs).`
...
...
rule for the last row (source label is `fd`):
`bagContains(P, pd) :-`
`contributedTo(P), help(P, C), hasLabel(C, un).`

Table 3.7: The Corresponding logical rules for Help Contribution links

Case	**Resulting Label**
1. If the bag has only one label	the single label
2. If the parent goal has multiple full labels of the same polarity, and no other labels, such as {`fs, fs, fs`} or {`fd, fd`}	the full label
3. If all labels in the bag are of the same polarity, and a full label exists in the set of labels, such as {`ps, fs, ps`} or {`fd, pd`}	the full label
4. If the previous human judgment produced `fs` or `fd`, and a new contribution is of the same polarity	the full label

Table 3.8: Cases where the final label for parent elements can be automatically determined [7].

```
hasLabel(E, L) :- isLabel(L), bagContains(E, L), not bagContainsOtherThan(E,
L), contributedTo(E).
bagContainsOtherThan(E, L) :- isLabel(L), bagContains(E, L), bagContains(E,
L2), L2 <> L.
positiveBag(g) :- bagContains(g, ps).
positiveBag(g) :- bagContains(g, fs).
negativeBag(g) :- bagContains(g, pd).
negativeBag(g) :- bagContains(g, fd).
label(g, fs) :- bagContains(g, fs), not negativeBag(g).
label(g, fd) :- bagContains(g, fd), not positiveBag(g).
```

Table 3.9: Rules that reflect the automatic resolution cases (see table 3.8).

those atoms of type *contributedTo* will be checked against the rule. Also, it enhances readability of the program.

The second set of rules are responsible for resolving the set of labels in a node (of type *contributedTo*) to a single final label. In the cases described in table 3.8 the final label in a bag can be automatically determined.

We easily incorporate these cases into our logical program through the rules defined in table 3.9. The last case where human judgement is reused is translated into rules later on after we discuss how we handle human judgement in the program.

Conflict Resolution

An important aspect of i^* evaluation procedure is handling cases where automatic resolution is not available. In these cases, the proposed procedure asks the human to resolve the bag of labels to a final label. In order to integrate this effect in our approach, we offer two solutions: **(i)** allowing human intervention during reasoning and **(ii)** allowing human to specify preferences among incoming links to nodes and automating the resolution on that basis. We'll discuss the first approach here and discuss the second one in

```
resolvedLabel(p, l) :- label(c1, l1), ..., label(cn, ln).
```

Table 3.10: The rules that handle user's input in cases where automated resolution of labels is not available.

```
label(g, L) :- resolvedLabel(g, L), samePolarityOrUn(c(n+1), L), ...,
samePolarityOrUn(c(m), L).
```

```
samePolarityOrUn(X, L) :- label(X, J), samePolarity(J, L).
samePolarity(J, L) :- L = fs, J = fs.
samePolarity(J, L) :- L = ps, J = fs.
...
...
```

Table 3.11: The logical rule that reuses human judgement for element p.

the future work chapter. Also, the two approaches can be integrated to give the user a semi-automated handling of these cases.

In the first approach, the framework assigns a distinguished predicate to the node in which the the bag of labels can't be automatically resolved to a final label. The UI then picks that predicate and asks the user for a resolution. The user then selects a set of children based on which he's deciding to resolve the label of the parent to say l. Consequently, the framework includes a rule to the logic program such that whenever the set of user selected children have labels equal to the current labels, the label of the parent will be determined as l. Table 3.10 demonstrates the rules that produce this effect in the model, assuming that the user has determined the label of parent element p to be l, based on the labels of elements in $S = \{c_1, c_2, \ldots, c_n\}$, where S is a subset of p's children. Also Assume that the labels of $c_1, c_2, ..., c_n$ are respectively equal to $l_1, l_2, \ldots, l_n$ when this rule is being injected to the program. In the following sections, we will provide a second approach that given initial user's input on link priorities can determine the label of the parent automatically.

As mentioned earlier, the framework provides the ability of reusing human judgment: If a human judgment is provided for a parent element p when among all its children, $c_1, c_2, \ldots, c_n$ have had a known label, based on the last case described in table 3.8 the framework includes a rule in the program (see table 3.11) such that if every child of p other than $c_1, c_2, \ldots, c_n$ have a label of the same polarity to the human-determined label l, the label of p would be automatically resolved to l.

At this point, our logic program $P(M)$ of a goal model M is capable of propagating the initial labels.

3.2.5 User Specific Propagation

A user can have his own way of handling non-automatic cases for label bag resolution; e.g. *if we have more partially satisfied contributions than partially denied ones, the ultimate label of the parent should be partially satisfied.* Or even in a lower level, the very basic propagation rules which in our work are based on i^* evaluation procedure could be different. Although it is not straightforward all the times, due to the expressive power of the language, user specific propagations can be incorporated into the logical translation by modifying the propagation rules.

3.2.6 Demonstration of Bottom-Up propagation

For demonstrating our approach in action, we use a portion of the goal model taken from GM [5] that includes the different types of links in i^* and encoded the goal model along with some initial labels in terms of logic program rules. Figure 3.2 shows the initial diagram along with the initial labels assigned.

The code snippet shown in table 3.12 demonstrates the equivalent logic program that our tool generates for the given goal graph. The program uses propagation rules as described throughout this section based upon the type of incoming links to a node to determine the relation between the labels of its neighbors and itself.

When the program appearing in table 3.12 is fed into $\mathcal{DLV}$, there's only one model (answer set) returned; The atoms in the answer set are shown in table 3.13. The corresponding evaluation result to this answer set is depicted in figure 3.3. We can verify the result by observing that if elements a, b and c have initial labels `fd`, `fs` and `fd` respectively, the label of element d is uniquely determined by the means-ends rules created specifically for d to be the greater label of elements b and c which is `fs`. The label of d along with the label of element c would then determine the label of element e through decomposition link rules pertaining to element e; from this point on, the label of elements f, g and h is uniquely determined based on the propagation rules driven from table 3.1 appearing in 3.12 for these elements and their outgoing contribution links.

3.3 Top-down Evaluation

The objective in this section is to provide logical rules that, given the user's desired labels for high level goal G in the model, can propose alternatives that when propagated, will evaluate G to the given desired label with respect to i^* evaluation procedure. There rules are provided for the UI provided in section 2.2.

```
isLabel(fs).
isLabel(fd).
isLabel(un).
isLabel(cf).
isLabel(ps).
isLabel(pd).

indexLabel(fd, 0).
indexLabel(pd, 1).
indexLabel(un, 2).
indexLabel(cf, 3).
indexLabel(ps, 4).
indexLabel(fs, 5).

hasLabel(b, fs).
hasLabel(a, fd).
hasLabel(c, fd).

bagContainsMoreThan(E, X) :- bagContains(E, Y),
isLabel(Y), isLabel(X), Y > X.
bagContainsLessThan(E, X) :- bagContains(E, Y),
isLabel(Y), isLabel(X), Y < X.

bagContains(d, L) :- hasLabel(b, L).
bagContains(d, L) :- hasLabel(a, L).
hasLabel(d, X) :- bagContains(d, X), not
bagContainsMoreThan(d, X), isLabel(X).

bagContains(e, L) :- hasLabel(d, L).
bagContains(e, L) :- hasLabel(c, L).
hasLabel(e, X) :- bagContains(e, X), not
bagContainsLessThan(e, X), isLabel(X).

bagContains(h, pd) :- hasLabel(e, fd).
bagContains(f, ps) :- hasLabel(c, fd).
bagContains(g, ps) :- hasLabel(f, ps).
contributedTo(h).
contributedTo(f).
contributedTo(g).
hasLabel(E, L) :- isLabel(L), bagContains(E, L),
not bagContainsOtherThan(E, L), contributedTo(E).
bagContainsOtherThan(E, L) :- isLabel(L),
bagContains(E, L), bagContains(E, L2), L2 <> L.
```

Table 3.12: The translated logial program for the goal model in figure 3.2

```
{isLabel(fs), isLabel(fd), isLabel(un),
isLabel(cf), isLabel(ps), isLabel(pd),
indexLabel(fs,5), indexLabel(fd,0),
indexLabel(un,2), indexLabel(cf,3),
indexLabel(ps,4), indexLabel(pd,1),
contributedTo(h), contributedTo(f),
contributedTo(g), hasLabel(b,fs),
hasLabel(a,fd), hasLabel(c,fd),
bagContainsMoreThan(d,fd),
bagContainsMoreThan(d,cf),
bagContainsMoreThan(e,cf),
bagContainsMoreThan(f,fs),
bagContainsMoreThan(f,fd),
bagContainsMoreThan(f,cf),
bagContainsMoreThan(f,pd),
bagContains(d,fs), bagContains(d,fd),
bagContains(e,fd), bagContains(f,ps),
bagContainsLessThan(d,fs),
bagContainsLessThan(d,un),
bagContainsLessThan(d,ps),
bagContainsLessThan(d,pd),
bagContainsLessThan(e,fs),
bagContainsLessThan(e,un),
bagContainsLessThan(e,ps),
bagContainsLessThan(e,pd),
bagContainsLessThan(f,un),
bagContainsOtherThan(d,fs),
bagContainsOtherThan(d,fd), hasLabel(d,fs),
hasLabel(e,fd), hasLabel(f,ps),
bagContains(e,fs), bagContains(h,pd),
bagContains(g,ps), hasLabel(h,pd),
hasLabel(g,ps), bagContainsMoreThan(g,fs),
bagContainsMoreThan(h,fs),
bagContainsMoreThan(e,fd),
bagContainsMoreThan(g,fd),
bagContainsMoreThan(h,fd),
bagContainsMoreThan(g,cf),
bagContainsMoreThan(h,cf),
bagContainsMoreThan(g,pd),
bagContainsLessThan(g,un),
bagContainsLessThan(h,un),
bagContainsLessThan(h,ps),
bagContainsOtherThan(e,fd),
bagContainsOtherThan(e,fs)}
```

Table 3.13: The unique answer set for the logical program in table 3.12

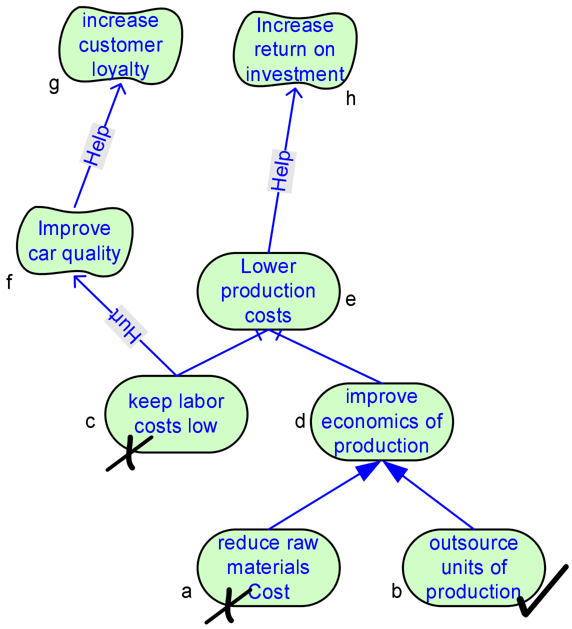

increase customer loyalty
g
Increase return on investment
h
Help
Help
Improve car quality
f
Lower production costs
e
Hurt
keep labor costs low
c
improve economics of production
d
reduce raw materials Cost
a
outsource units of production
b

Figure 3.2: The goal model used for demonstrating the bottom up propagation.

For this purpose, we need rules that can span the space of desired alternatives. From a top-down reasoning point of view, depending on the type of incoming links to a node P and a desired label for P, say L, we specify labels for the contributing nodes to P that along with their links to P will evaluate the label of P as L. For example, in a means-ends relationship among parent p and the set of children C where the desired label for p is l, a valid set of labels for nodes in C would be one in which the minimum label is equal to l.

There are actually two sets of rules required to accomplish this purpose: **(i)** Constraint Rules: rules that determine if a set of labels for nodes in C will evaluate p to l. These rules guarantee the **soundness** of the program. and **(ii)** Generation Rules: rules that *generate* all possibilities for nodes' labels. The possibilities will be then filtered out by the constraint rules for this node and other nodes. These rules provide **completeness** for our program.

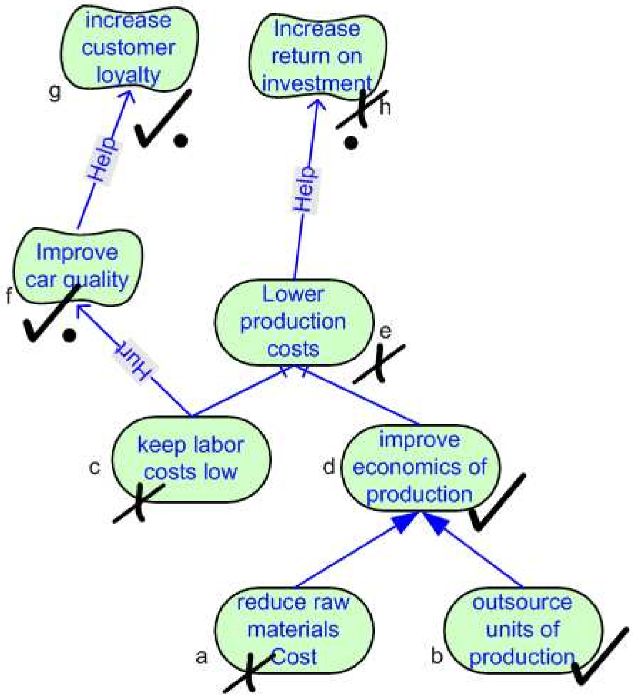

increase customer loyalty
g
Increase return on investment
h
Help
Help
Improve car quality
f
Lower production costs
e
Hurt
keep labor costs low
c
improve economics of production
d
reduce raw materials Cost
a
outsource units of production
b

Figure 3.3: The corresponding evaluation result to the answer set shown in table 3.13

`label(dependee, L) :- label(depender, L), label(L)`

Table 3.14: Top-down evaluation rule for dependency link from `dependee` to `depender` with `L` as the desired label for `depender`.

We provide these rules based on the type of relationship each node participate in the model.

3.3.1 Dependency Links

The dependency links transfer the evaluation label *as it is* from the dependee to the depender. Therefore, for any desired label `L` for node `depender`, we have the rule given in table 3.14.

3.3.2 Decomposition Links

If goal g is decomposed to tasks $t_1, t_2, \ldots, t_n$ the label of g would be evaluated as the minimum member of $L = \{l_i : label(t_i, l_i) : 1 \leq i \leq n\}$. Consequently, desiring label `l` for g would amount to **(i)** having `l` in L and **(ii)** that every label in L be greater than or equal to `l`.

```
minLabel(t1, L) :- label(g, L), labelType(L).
minLabel(t2, L) :- label(g, L), labelType(L).
...
minLabel(tn, L) :- label(g, L), labelType(L).
label(t1, L) v ...v label(tn, L) :- label(g, L), labelType(L).
label(X, ps) v label(X, fs) :- minLabel(X, ps), element(X).
...
```

Table 3.15: rules for top-down evaluation along a decomposition relationship that decomposes `g` into `t1, t2, ..., tn`.

```
maxLabel(t1, L) :- label(L), label(g, L).
maxLabel(t2, L) :- label(L), label(g, L).
......maxLabel(tn, L) :- label(L), label(g, L).
label(t1, L) v ..., label(tn, L) :- label(g, L), labelType(L).
label(X, pd) v label(X, fd) :- maxLabel(X, pd), element(X).
...
```

Table 3.16: rules for top-down evaluation along a means-ends relationship between `g` and `t1, t2, ..., tn`.

As mentioned earlier, $\mathcal{DLV}$ enabled us to have disjunctive expressions in the head of rules and that's the key for us to be able to express the space of possibilities. In the case described above, the rules in table 3.15 generate the valid possibilities for children. This rule schema as well as others is presented in a general form and will be instantiated and customized by the framework for each decomposition link appearing in the goal model.

Basically, the rules consider two options: Either **(i)** `t1`'s label (`l1` is `l` and the label of `t2, t3, ..., tn` is at least `l` or **(ii)** `l1` is greater than `l` and the set $S = \{$`l2, l3, ..., ln`$\}$ contains `l` and moreover all its members are greater than or equal to `l`.

3.3.3 Means-ends Links

A symmetric argument to decomposition links holds for means-ends relationships. Table 3.16 shows the rules that we include in our framework if `g` has means-ends relationship with `t1, t2, ..., tn`.

The rules in table 3.16 are organized such that the generated values for elements `t1`, ..., `tn` are such that at least one label equal to the desired label `L` is generated among them and also that all their labels are less than or equal to `L`.

```
labelSet1(b) :- label(g, ps).
labelSet2(c) :- label(g, ps).
label(b, ps) :- labelSet1(b).
label(c, fd) v label(c, pd) :- labelSet2(c).
```

Table 3.17: rules for top-down evaluation along incoming contribution links to a node.

3.3.4 Contribution Links

In this part, we assume that the top-down propagation along the contribution links towards node g, only enumerates cases where after bottom-up propagation of the enumerated labels, g's bag of labels can be automatically resolved into a label with respect to table 3.8. E.g., if the desired label for node g is full satisfaction (`fs`), the labels of the contributing nodes should be such that when propagated along the connecting link between them and g, generate either `fs` or `ps` and moreover that at least one of them generates `fs` in g's bag of labels.

The rules that should be generated obviously depend on the specific desired label of the parent element g as well as the type of incoming contribution links to node g. For example, Rules in table 3.17 handle the case where g has two children b and c where they have *Help* and *Break* contribution links to element g respectively and the desired label for g is `ps`. In this case, all the contributed labels in g's bag of labels should necessarily be `ps`. Considering the type of contribution links from b and c to g, according to the label propagation table (table 3.1) the label of b should be `ps` and the label of c should be either `pd` or `fd`.

3.3.5 Demonstration of Top-Down evaluation

We use the same diagram as the one used in the demonstration of bottom up propagation, however, instead of assigning initial low level labels, we set high level goals to be achieved and expect the framework to provide appropriate solutions for us. The tool will generate instantiated rules based on the rules introduced in this section along with the rules in the bottom up propagation section. The answer sets of the resulting logic program, *if any*, when translated back to the labels in the goal model would provide suggestions to the analyst for achieving the preset high level goals.

Figure 3.4 depicts the goal model along with objective labels which are surrounded in circles.

The code snippet shown in table 3.18 is the equivalent logic program to the goal model in figure 3.4. Given this program as the input for $\mathcal{DLV}$, it turns out that there's

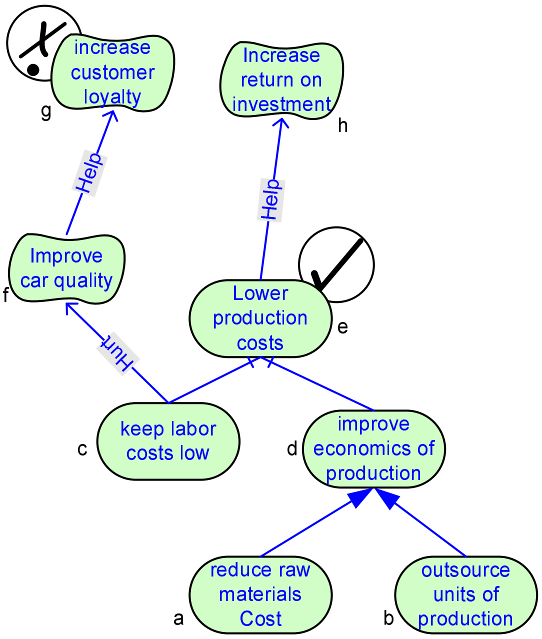

increase customer loyalty
g
Increase return on investment
h
Help
Help
Improve car quality
f
Lower production costs
e
Hurt
keep labor costs low
c
improve economics of production
d
reduce raw materials Cost
a
outsource units of production
b

Figure 3.4: A partial goal model of GM [5] and some objective labels. The framework is reponsible for providing solutions that satisfy these labels.

no answer set for this program.

Let's verify the result; The outcome indicates that there doesn't exist a valid assignment of labels to leaf level elements such that both of our objective labels are entailed. By a simple analysis we see that since *lower production costs* goal is decomposed into goals *keep labor costs low* and *improve economics of production*, the label of both of the latter goals should be at least `fs`. This would leave us with one choice for both these goals: they both should be fully satisfied. Now regardless of the labels of the goals *reduce raw materials Cost* and *outsource units of production*, we can see that if the label of *keep labor costs low* is $\widehat{\text{fs}}$, then based on the propagation table (table 3.1) in i^* evaluation procedure, the label of goals *improve car quality* and *increase customer loyalty* would be `pd` and `ps` respectively. This is in contradiction to the fact that the label of *increase customer loyalty* is set to be *ps* as an objective label.

```
isLabel(fs).
isLabel(fd).
isLabel(un).
isLabel(cf).
isLabel(ps).
isLabel(pd).

indexLabel(fd, 0).
indexLabel(pd, 1).
indexLabel(un, 2).
indexLabel(cf, 3).
indexLabel(ps, 4).
indexLabel(fs, 5).

hasLabel(e, fs).
hasLabel(g, ps).
bagContainsMoreThan(E, X) :- bagContains(E, X), bagContains(E, Y), isLabel(Y), isLabel(X),
indexLabel(X, I), indexLabel(Y, J), J > I.
bagContainsLessThan(E, X) :- bagContains(E, X), bagContains(E, Y), isLabel(Y), isLabel(X),
indexLabel(X, I), indexLabel(Y, J), J < I.

bagContains(d, L) :- hasLabel(b, L).
bagContains(d, L) :- hasLabel(a, L).
hasLabelInferred(d, X) :- bagContains(d, X), not bagContainsMoreThan(d, X), isLabel(X).

bagContains(e, L) :- hasLabel(d, L).
bagContains(e, L) :- hasLabel(c, L).
hasLabel(e, X) :- bagContains(e, X), not bagContainsLessThan(e, X), isLabel(X).

bagContains(h, pd) :- hasLabel(e, fd).
bagContains(h, ps) :- hasLabel(e, fs).

bagContains(f, ps) :- hasLabel(c, fd).
bagContains(f, pd) :- hasLabel(c, fs).

bagContains(g, ps) :- hasLabel(f, ps).
bagContains(g, pd) :- hasLabel(f, pd).

contributedTo(h).
contributedTo(f).
contributedTo(g).

hasLabel(E, L) :- isLabel(L), bagContains(E, L), not bagContainsOtherThan(E, L),
contributedTo(E).
bagContainsOtherThan(E, L) :- isLabel(L), bagContains(E, L), bagContains(E, L2), L2 <> L.

hasMinLabel(c, L) :- hasLabel(e, L).
hasMinLabel(d, L) :- hasLabel(e, L).
hasLabel(c, L) v hasLabel(d, L) :- hasLabel(e, L).
hasLabel(E, fs) :- hasMinLabel(E, fs).

hasMaxLabel(a, L) :- hasLabel(d, L).
hasMaxLabel(b, L) :- hasLabel(d, L).

hasLabel(a, L) v hasLabel(b, L) :- hasLabel(d, L).
hasLabel(X, pd) v hasLabel(X, fd) v hasLabel(X, ps) v hasLabel(X, fs) :- hasMaxLabel(X,
fs).

:- hasLabel(E, X), hasLabel(E, Y), X <> Y.
```

Table 3.18: The equivalent logic program for the goal model and objective labels shown in figure 3.4

```
label(E, L) ?
```

Table 3.19: The query that determines the labels which are true in all alternatives (models of the logic program).

Based on the above analysis, we can lift the contradiction by either omitting the objective label on goal *increase customer loyalty* or by setting the objective label to *pd*. By replacing the rule `hasLabel(g, ps).` with `hasLabel(g, pd).` in the program given in table 3.18. When this program is given to $\mathcal{DLV}$ as input, 7 models are returned. By analyzing the possibilities for goals *reduce raw materials Cost* and *outsource units of production*, we can confirm the validity of the outcome: the label of either goals can be any value in the set {`fd, pd, ps, fd`} and one of the two labels should necessarily be equal to `fs`.

3.4 Partial Solutions Resulting from User Inputs

We address this feature by exploiting *queries* in $\mathcal{DLV}$. $\mathcal{DLV}$ have multiple modes of querying including *Cautious* (or *sceptical*). A query is cautiously true for a substitution, if it is satisfied in all models of the program [9]. This would be the logical dual of the implication concept introduced earlier. We simply determine the implicated labels by issuing the query in table 3.19.

3.4.1 Demonstration of Implicated Labels

By this point, we can see that in the example goal model given in the previous two parts regarding bottom-up propagation and top-down evaluation, regardless of the various labels that other elements could take in order to satisfy user goals and/or propagate user's initial labels, the label of some elements will be uniquely determined by user's initial input. For example, in the top-down evaluation demonstration (figure 3.4), consider the soft goals *Improve car quality* and *increase customer loyalty.* We can see that if the user specifies label `fs` as an objective label for goal *Lower production costs*, no matter what labels are assigned to other elements, the label of these two soft goals should be equal to `pd`, `pd` in **all models**. If the user can get this information from the tool when he's determining his objective goals as well as initial labels, he can avoid going over all the alternatives and see the same label for these kind of elements. In order to provide this feature, we reduce the probelm to **cautious reasoning** problem in answer set programming.

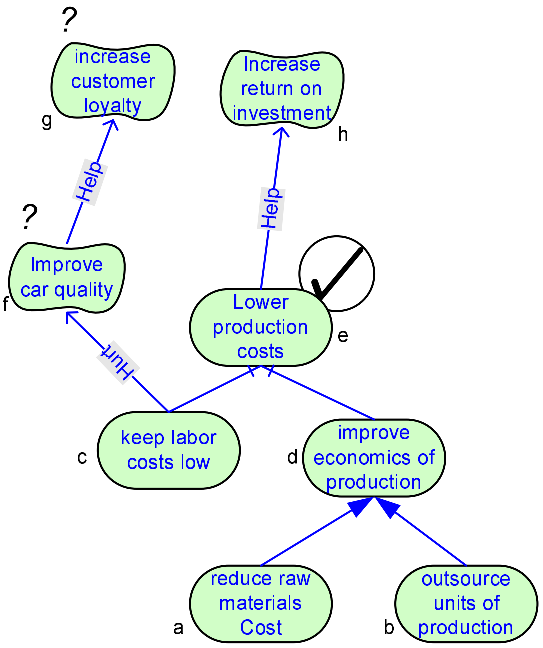

?
increase customer loyalty
g
Increase return on investment
h
Help
Help
?
Improve car quality
f
Lower production costs
e
Hurt
keep labor costs low
c
improve economics of production
d
reduce raw materials Cost
a
outsource units of production
b

Figure 3.5: The goal model used to demonstrate how labels can be implied uniquely, given user's labels as input.

Cautious reasoning deals with user queries and is true for query Q and program P, if and only if Q is true in all models of program P. Therefore, if the query given in table 3.19 is *cautiously* true for any E, L, element E has to have label L no matter how other elements are evaluated/assigned. By running this query in cautious mode over the program given in table 3.18 and replacing the rule `hasLabel(g, ps).` with `hasLabel(g, pd).` (see the previous part for the reason behind this modification); the outcome is shown in table 3.20. Our tool will translate the outcome back to goal model labels and graphically show the result to the user.

If the size of the model grows, these queries specifically can be significantly costlier. To address this issue, we can limit the scope of elements involved in the query. An option would be to consider elements which are in the neighborhood of initial/objective labels. The intuition behind this idea is that the neighbor elements are more likely to

```
label(e, fs).
label(d, fs).
label(c, fs).
label(h, ps).
label(f, pd).
label(g, pd).
```

Table 3.20: The result of issuing the cautious query in table 3.19 on the program in table 3.18

```
label(h, E) ?
label(f, E) ?
label(d, E) ?
label(c, E) ?
```

Table 3.21: The query code snippet corresponding to the neighborhood shown in figure 3.6

be constrained than farther elements. Figure 3.6 demonstrates this idea in a sample goal model. The cautious query issues for the model depicted in figure 3.6 is demonstrated in table 3.21.

3.5 User Preferences in evaluation

A first step towards this ambition is to come up with a language that can express preferences among soft goals and in a higher level even between the soft-goals of different agents involved in the goal model. For this purpose, we adopt the *preference language* semantics and approach proposed in [13] and adapt towards our framework. The adapted language is capable of capturing complex multi-dimensional preferences among qualities as well as simple ones. It's interesting to note that usually an analyst is in charge of adjusting strategies for *cooperative* agents in the business as opposed to *competitive* ones. In chapter 4 we discuss the research venues that will be opened based upon this observation.

In the following paragraphs, we elaborate on the semantics of the preference language and the way they can be integrated in our framework for top down evaluation with preferences.

3.5.1 Preference Language Definition and Semantics

In this part, we provide an adaptation of the language and semantics proposed in [13] that can be directly used in our framework for evaluation.

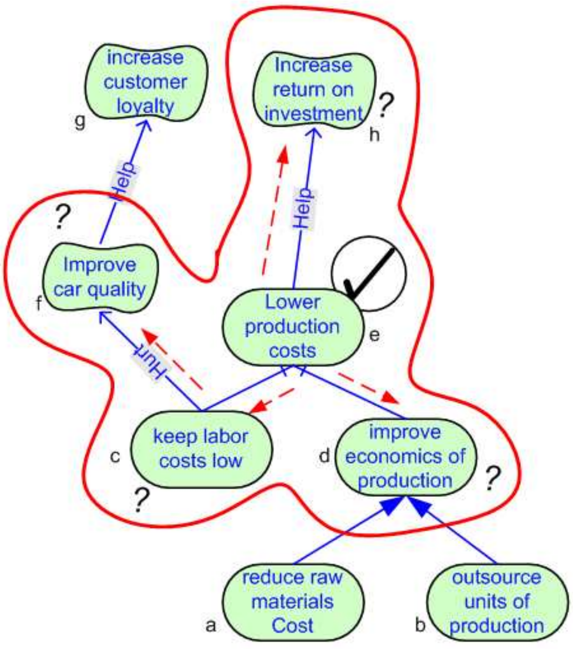

increase customer loyalty
g
Increase return on investment
?
h
Help
?
Help
Improve car quality
f
Lower production costs
e
Hurt
keep labor costs low
c
?
improve economics of production
d
?
reduce raw materials Cost
a
outsource units of production
b

Figure 3.6: The tool can issue cautious queries for an element's neighborhood. The neighborhood scope is highlighted with a boundary line. The elements for which the query is issues are those with a question mark and in this case, the cautious query will return an implicated label for all of them, since they are all uniquely determined by the choice of user for the initial objective label for the goal *Lower production costs*

Definition 3.5-1 (Basic Desire Formula). Basic desire formulas are defined using the following rules:

- $label(g, fs)$, $label(g, fd)$, $label(g, ps)$, $label(g, pd)$ for any goal g in the goal model are each considered a basic desire formula.
- Given basic desire formulas φ_1 and φ_2, each of $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ and $\neg\varphi$ is also a basic desire formula.

Before moving on to preferences in higher level, we provide another definition that ultimately let us find the most preferred solution (alternative) in a goal model.

Definition 3.5-2 (Basic Desire Semantics). Let α be an alternative solution (set of labels assigned to the elements of the model) and let φ be a basic desire formula. α satisfies φ (written as $\alpha \models \varphi$) iff one of the following conditions hold:

- $\varphi = \varphi_1 \wedge \varphi_2$, $\alpha \models \varphi_1$ and $\alpha \models \varphi_2$
- $\varphi = \varphi_1 \vee \varphi_2$, $\alpha \models \varphi_1$ or $\alpha \models \varphi_2$

- $\varphi = \neg\varphi_1$ and $\alpha \not\models \varphi_1$
- $\varphi = label(g, fs)$ or $label(g, fd)$ or $label(g, pd)$ or $label(g, ps)$ and in the logic model of solution α, the atom φ is present.

Given the above definition we can define an ordering among solutions with respect to basic desire formulas:

Definition 3.5-3 (Ordering between solutions w.r.t basic desire formulae). Let φ be a basic desire formula and let α and β be two solutions. Alternative α is **preferred** to β, shown as $\alpha \prec_\varphi \beta$, iff $\alpha \models \beta$ and $\beta \not\models \varphi$. Also α and β are **indistinguishable** with respect to φ, written as $\alpha \approx_\varphi \beta$, iff one of the following cases occurs:

- $\alpha \models \varphi$ and $\beta \models \varphi$
- $\alpha \not\models \varphi$ and $\beta \not\models \varphi$

Following [13], we define non strict preference in the following definition:

Definition 3.5-4 (Weak Ordering between solutions w.r.t basic desire formula). Let φ be a basic desire formula and let α and β be two solutions. α is **weakly preferred** to β, shown as $\alpha \preceq_\varphi \beta$, iff $\alpha \prec_\varphi \beta$ or $\alpha \approx_\varphi \beta$.

The following interesting propositions will be given here with out proof. You can refer to [13] for a comprehensive proof that is based on the original language but can be directly applied to ours.

Proposition 3.5-5. *if φ is a basic desire formula (referred to as* **BDF** *henceforth), the relation $\approx_\varphi$ is an equivalence relation.*

and Moreover:

Proposition 3.5-6. *Given φ as BDF, the relation $\preceq_\varphi$ specifies a partial order over members of equivalence classes of $\approx_\varphi$.*

In the rest of this part, we extend the definitions and semantics to cover the second and third sample preference formulas given earlier; namely the single and multi agent preferences respectively.

Definition 3.5-7 (Single Agent Preference). A single agent preference formula for agent a is of the form $\psi_a = \varphi_1 \lhd \varphi_2 \lhd \ldots \varphi_n$ where φ_i is a basic desire formula over a's goals for $1 \leq i \leq n$.

Definition 3.5-8 prioritizes solutions based on single agent preferences:

Definition 3.5-8 (Ordering between solutions w.r.t. single agent preference formula). Let α, β be two solutions and $\psi = \varphi_1 \lhd \varphi_2 \lhd \dots \varphi_n$ be a single agent preference formula. Then we will have an extended definition for $\prec$, $\approx$ and $\preceq$ relations:

- $\alpha \prec_\psi \beta$ iff $\exists i$ such that $1 \leq i \leq n$ and:
    1. $\alpha \prec_{\varphi_i} \beta$, and
    2. $\forall \ (1 \leq j < i) \Rightarrow \alpha \approx_{\varphi_j} \beta$
- $\alpha \approx_\psi \beta$ iff $1 \leq \forall i \leq n \Rightarrow \alpha \approx_{\varphi_i} \beta$
- $\alpha \preceq_\psi \beta$ iff $\alpha \prec_\psi \beta$ or $\alpha \approx_\psi \beta$

and the *equivalence* proposition also gets extended:

Proposition 3.5-9. *If ψ is a single agent preference formula, the relation $\preceq_\psi$ defines a partial order on the members of equivalence classes of $\approx_\psi$.*

and on this basis, we can define the most preferred alternative solution in the following way:

Definition 3.5-10 (Most preferred alternative solution w.r.t a single agent preference formula). Alternative solution α is **most preferred** with respect to single agent preference formula ψ if there is no solution β such that $\beta \prec_\psi \alpha$.

The last batch of definitions and propositions deal with preferences at a multi agent level, following the third sample we've seen in the beginning of this section.

Definition 3.5-11 (Multiagent Preference Formula). A multiagent preference formula is defined by the following rules:

- A single agent preference formula ψ_a is considered multiagent preference formula.
- if ψ_a and ψ_b are multiagent preference formulae, then $\psi_a \& \psi_b$, $\psi_a | \psi_b$ and $!\psi_a$ are each a multiagent preference formula;
- If $\psi_{a_1}, \psi_{a_2}, \dots, \psi_{a_n}$ are multiagent preference formulae, then $\psi_{a_1} \lhd \psi_{a_2} \lhd \dots \lhd \psi_{a_n}$ is a multiagent preference formula.

Definition 3.5-12 prioritizes solutions based on multiagent preference formulae.

Definition 3.5-12 (Ordering between solutions w.r.t. Multiagent Preference Formula). If ω is a multiagent preference formula,

- α is preferred to β w.r.t to ω ($\alpha \prec_\omega \beta$) iff:

1. ω is a single agent preference formula and $\alpha \prec_\omega \beta$ w.r.t. definition 3.5-8.
2. $\omega = \psi_a \& \psi_b$ and $\alpha \prec_{\psi_a}$ and $\alpha \prec_{\psi_b}$
3. $\omega = \psi_a | \psi_b$ and we have one of:
   (a) $\alpha \prec_{\psi_a} \beta$ and $\alpha \prec_{\psi_b} \beta$
   (b) $\alpha \prec_{\psi_a} \beta$ and $\alpha \approx_{\psi_b} \beta$
   (c) $\alpha \approx_{\psi_a} \beta$ and $\alpha \prec_{\psi_b} \beta$
4. $\omega = !\psi_a$ and $\beta \prec_{\psi_a} \alpha$
5. $\omega = \psi_{a_1} \lhd \psi_{a_2} \ldots \lhd \psi_{a_k}$ and $1 \leq \exists i \leq k$ such that: (i) $1 \leq \forall j \leq i : \alpha \approx_{\psi_{a_j}} \beta$ and (ii) $\alpha \prec_{\psi_{a_i}} \beta$ - in other words with respect to all preference formulae which are more prior than ψ_{a_i}, α is indistinguishable to β, however with respect to ψ_{a_i}, α is less preferred to β

- α is **indistinguishable** from β, written as $\alpha \approx_\omega \beta$ iff one of the followings are true:

1. ω is a single agent preference formula and $\alpha \approx_\omega \beta$.
2. $\omega = \psi_a \& \psi_b$, $\alpha \approx_{\psi_a} \beta$ and $\alpha \approx_{\psi_b} \beta$
3. $\omega = \psi_a | \psi_b$, $\alpha \approx_{\psi_a} \beta$ and $\alpha \approx_{\psi_b} \beta$
4. $\omega = !\psi_a$ and $\alpha \approx_{\psi_a} \beta$
5. $\omega = \psi_{a_1} \lhd \psi_{a_2} \lhd \ldots \psi_{a_k}$ and $1 \leq \forall i \leq k$, we have $\alpha \approx_{\psi_{a_i}} \beta$.

- $\alpha \preceq_\omega \beta$ iff $\alpha \prec_\omega \beta$ or $\alpha \approx_\omega \beta$

Proposition 3.5-13. *if ω is a multiagent preference formula, then:*

1. $\approx_\omega$ is an equivalence relation; and

2. $\preceq_\omega$ induces a partial order on the set of representatives of the equivalence classes of $\approx_\omega$.

and the last definition in the final batch characterizes the most preferred solution when having a multiagent preference formula.

Definition 3.5-14 (Most Preferred Solution). Given a multiagent preference formula ω, we say that an solution α is the most preferred one w.r.t to ω iff there is no solution β such that $\beta \prec_\omega \alpha$.

In the following paragraphs, we integrate this language along with its semantics into our framework which enables the framework to find solutions that are the most preferred ones with respect to a user preference.

```
:∽ Conj.  [Weight:Level]
```

3.5.2 Weak Constraints in $\mathcal{DLV}$

Integrity constraints in $\mathcal{DLV}$ have to be always satisfied in models, whereas *weak* constraints should be satisfied if possible. In other words if a weak constraint is not satisfied in model m, m would not be necessarily disqualified to be among the models in the output. According to [1], *the answer sets of a program P with a set W of weak constraints are those answer sets of P which minimize the number of violated weak constraints and are called best models of (P, W).* Obviously, a program may have more than one best model when they share the same number of violated weak constraints.

Weak constraints are specified in the following format:

where `Conj` is a conjunction of literals and `Weight` and `Level` are both positive integers. `Level` specifies the priority level of the constraint: the higher the number `Level` is, the higher prior is the weak constraint. For each violated weak constraint rule r, the weight associated to rule r will be added to the sum of other violated constraints having the same priority (represented by `Level`). The best model would be the model that has the least weight within the highest priority level. In the rules presented in this work, we assume the same level (1) for all weak constraints.

Effectively, one can quantify qualitative constraints and preference with this feature. Following a similar approach in [13], we quantify our framework's preferences using weak constraints. The general intuition behind this approach is to assign lower weights to more preferred models.

We consider preference formulas in order of complexity: In the first take, let's consider basic desire formulas (BDF's). Assuming φ be a BDF, we add an atom called n_φ to the program that corresponds uniquely to φ. Moreover, we add a predicate called `satisfy` that holds in a model m for atom n_φ iff $m \models \varphi$. The following rule is thus added to the program for a given BDF:

$$\Pi_\varphi = \texttt{satisfy(}n_\varphi\texttt{)}\ \ \texttt{:-}\ \ \mathit{expand}(n_\varphi)$$

where $\mathit{exapnd}(n_\varphi)$ is the formula corresponding to φ itself. Having this predicate in the model, we can write a weak constraint based on the introduced predicate: We assign a weight of 1 to the case where $\texttt{satisfy(}n_\varphi\texttt{)}$ is not present in the model.

Proposition 3.5-15. *Given a basic desire formula φ and goal model M, the answer set of the program $\Pi = \Pi_M \cup \Pi_\varphi \cup \{$ `:∽ not satisfy(`n_φ`) [1]` $\}$ corresponds to solutions that are* **most preferred** *w.r.t. φ. (note that when the priority of the weak constraints* *are not set, by default, they are assumed to be 1.)*

As the preference formulas get to the more complex levels (single agent and multiagent), we need a more complex weighting mechanism that can distinguish between models based on the preference formulas they satisfy. Since our preference language structure is very similar to the one defined in [13], we adopt the notion of *admissible weight function* from [13] in order to extend the weighting mechanism to more complex forms of preferences.

Definition 3.5-16 (Admissible Weight Functions). A weight function w_Ψ is called an *admissible* weight function w.r.t. Ψ if for any two alternatives α, β, we have:

1. $w_\Psi(\alpha) < w_\Psi(\beta)$ if $\alpha \prec_\Psi \beta$
2. $w_\Psi(\alpha) = w_\Psi(\beta)$ if $\alpha \approx_\Psi \beta$

With an admissible weight function, we can bridge the gap between the weak constraint feature in $\mathcal{DLV}$ and the semantics of preference in our framework:

Proposition 3.5-17. *If w_Ψ is an admissible weight function w.r.t Ψ, an alternative o which minimizes w_Ψ is a most preferred alternative w.r.t. Ψ.*

This proposition reduces the problem of finding a most preferred alternative to the problem of finding an admissible function for the various preference formulas we have and using the weak constraint feature in $\mathcal{DLV}$ to find the alternative corresponding to the answer set that minimizes this function. The following propositions propose such a function for the single agent and multiagent preference formulae.

Proposition 3.5-18. *Given $\psi = \varphi_1 \lhd \varphi_2 \ldots \lhd \varphi_n$ as a single agent preference formula, the following weight function is admissible w.r.t. ψ:*

$$w_\psi(o) = \sum_{i=n}^{1} 2^{n-i}.w_{\varphi_i}(o) \tag{3.3}$$

where $w_{\varphi_i}(o) = 1$ iff $o \not\models \varphi_i$ and $w_{\varphi_i}(o) = 0$ otherwise.

The correctness of proposition 3.5-18 follows from the fact that the priority of BDF φ_i is strictly higher than φ_j for $i < j$.

The next proposition deals with multiagent preference formulas:

Proposition 3.5-19. *Given Ψ as a multiagent preference formula, w_Ψ is an admissible weight function w.r.t. Ψ where:*

1. $w_\Psi = w_{\Psi_1} + w_{\Psi_2}$ *if* $\Psi = \Psi_1 | \Psi_2$

2. $w_\Psi = w_{\Psi_1} + w_{\Psi_2}$ *if* $\Psi = \Psi_1 \& \Psi_2$

3. $w_\Psi = max(w_{\Psi_1}) - w_{\Psi_1}$ *if* $\Psi = !\Psi_1$

4. $w_\Psi = w_{\Psi_2} + max(w_{\Psi_2}) \times w_{\Psi_1}$ *if* $\Psi = \Psi_1 \lhd \Psi_2$

where $max(w_x)$ *is the maximum weight that an alternative can potentially gain by the weight function* w_x *plus 1 (The addition of 1 is necessary for meeting the first condition of admissible weight function).*

In order to incorporate this weighting mechanism in logic program, we need to inject some predicates as well as some logic to compute the weight of an answer set and find the answer set with minimum weight. To this end, we add the predicate `w(p, n)` to the program where `p` is the unique name corresponding to a preference formula and `n` would be the weight of the answer set w.r.t. preference `p`. We also add the predicate `max(p, m)`, where `p` is the unique name of a preference formula and `m` is the maximum weight that an alternative can achieve w.r.t. preference `p`. Now we need some logic to define the existence of these predicates within our answer sets. Instances of the rules given in table 3.22 will be generated by the framework based on the preferences that the analyst specify in the front-end of the framework and will be injected to the logic program.

Given above rules, the engine will return the most preferred models. In case the user desired more coverage, we can use the *costbound* feature in $\mathcal{DLV}$ to return all the models with cost less than a certain number and gradually increase that number. We leave the elaboration on this feature for future work (see chapter 4).

3.5.3 Demonstration of user preferences in Top Down Evaluation

For the sake of presentation, we extend the same goal model as the one discussed in the top down evaluation section and study the effect of having preferences in the user's objectives. Consider the goal model given in figure 3.7. We can see that in order to satisfy the goal *improve economics of production*, the planner has a choice of either satisfying the goal *reduce raw materials Cost* or *outsource units of production.* Now suppose that the user has specified the single agent preference formula ψ in table 3.23 which is a preference specified over BDF's φ_g and φ_i.

The semantics of single agent preference formula implies that ψ_A favors the alternatives in the following order:

Basic Desire Formula For each BDF φ we add the logic required for generating the predicate `satisfy` along with the following rules for weighting the answer set w.r.t. φ	w(n_φ, 0) :- satisfy(n_φ). w(n_φ, 1) :- not satisfy(n_φ). max(n_φ, 2).
Single agent preference formula $\psi = \varphi_1 \lhd \varphi_2 \lhd \ldots \lhd \varphi_p$	w(n_ψ, S) :- w(n_{ψ_1}, s_1), ...w(n_{ψ_p}, s_p), S = $\sum_{i=p}^{1} 2^{p-i} \times s_i$. max($n_\psi$, 2^p).
Multiagent preference formula Ψ	
Ψ is a single preference formula	The same rules as the case of single agent preference formula.
$\Psi = \Psi_1 \mid \Psi_2$ or $\Psi = \Psi_1 \& \Psi_2$	w(n_Ψ, S) :- w(n_{Ψ_1}, S1), w(n_{Ψ_2}, S2), S = S1 + S2
$\Psi = !\Psi_1$	w(n_Ψ, S) :- w(n_{Ψ_1}, S1), max(n_{Ψ_1}, M), S = M - S1. max(n_Ψ, S) :- max(n_{Ψ_1}, S1), S = S1 + 1.
$\Psi = \Psi_1 \lhd \Psi_2$	w(n_Ψ, S) :- w(n_{Ψ_1}, S1), w(n_{Ψ_2}, S2), max(n_{Ψ_2}, M2), S = S2 + M2 $\times$ S1. w(n_Ψ, S) :- max(n_{Ψ_1}, M1), max(n_{Ψ_2}, M2), S = M2 + M1 $\times$ M2.

Table 3.22: The logic required for generating `satisfy` and `max` predicates as required part of implementing preferences semantics in $\mathcal{DLV}$

$$\begin{aligned}
&\varphi_g = hasLabel(g, fs) \vee hasLabel(g, ps)\\
&\varphi_i = hasLabel(i, fs) \vee hasLabel(i, ps)\\
&\psi_A = \varphi_i \lhd \varphi_g
\end{aligned}$$

Table 3.23: The preference formula used for demonstrating user preference in top down evaluation.

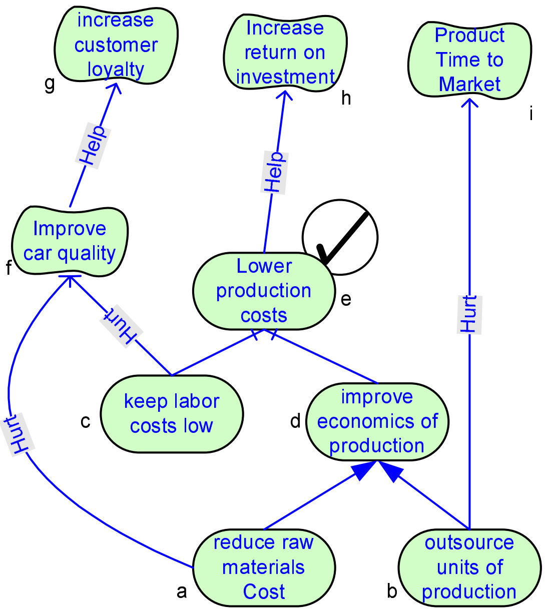

increase customer loyalty
g
Increase return on investment
h
Product Time to Market
i
Help
Help
Improve car quality
f
Lower production costs
e
Hurt
Hurt
Hurt
keep labor costs low
c
improve economics of production
d
reduce raw materials Cost
a
outsource units of production
b

Figure 3.7: The extended goal model of GM.

1. BDF's φ_i and φ_g are both implied ($o \models \varphi_i$ and $o \models \varphi_g$)
2. $o \models \varphi_i$ and $o \not\models \varphi_g$
3. $o \not\models \varphi_i$ and $o \models \varphi_g$
4. $o \not\models \varphi_i$ and $o \not\models \varphi_g$

In other words the user strictly prefers the soft goal (quality) *Product Time to Market* over *increase customer loyalty* and doesn't care either the soft goal *Increase return on investment* is satisficed or not. The code snippet shown in table 3.24 is the extended logic program that corresponds to the goal graph in figure 3.7 and the above preference formulae.

The answer sets returned by the ASP engine for the program in table table 3.24 are listed in table 3.25.

```
#maxint = 5.


isLabel(fs).  indexLabel(fd, 0).
isLabel(fd).  indexLabel(pd, 1).
isLabel(un).  indexLabel(un, 2).
isLabel(cf).  indexLabel(cf, 3).
isLabel(ps).  indexLabel(ps, 4).
isLabel(pd).  indexLabel(fs, 5).
hasLabel(e, fs).

bagContainsMoreThan(E, X) :- bagContains(E, X), bagContains(E, Y), isLabel(Y), isLabel(X),
indexLabel(X, I), indexLabel(Y, J), J > I.
bagContainsLessThan(E, X) :- bagContains(E, X), bagContains(E, Y), isLabel(Y), isLabel(X),
indexLabel(X, I), indexLabel(Y, J), J < I.

bagContains(d, L) :- hasLabel(b, L).
bagContains(d, L) :- hasLabel(a, L).
hasLabelInferred(d, X) :- bagContains(d, X), not bagContainsMoreThan(d, X), isLabel(X).

bagContains(e, L) :- hasLabel(d, L).
bagContains(e, L) :- hasLabel(c, L).
hasLabel(e, X) :- bagContains(e, X), not bagContainsLessThan(e, X), isLabel(X).

bagContains(h, pd) :- hasLabel(e, fd).
bagContains(h, ps) :- hasLabel(e, fs).
bagContains(f, ps) :- hasLabel(c, fd).
bagContains(f, pd) :- hasLabel(c, fs).
bagContains(g, ps) :- hasLabel(f, ps).
bagContains(g, pd) :- hasLabel(f, pd).
bagContains(i, pd) :- hasLabel(b, fs).
bagContains(i, pd) :- hasLabel(b, ps).
bagContains(i, un) :- hasLabel(b, un).
bagContains(i, ps) :- hasLabel(b, pd).
bagContains(i, ps) :- hasLabel(b, fd).
bagContains(f, pd) :- hasLabel(a, fs).
bagContains(f, pd) :- hasLabel(a, ps).
bagContains(f, un) :- hasLabel(a, un).
bagContains(f, ps) :- hasLabel(a, pd).
bagContains(f, ps) :- hasLabel(a, fd).


contributedTo(h).
contributedTo(f).
contributedTo(g).
contributedTo(i).


hasLabel(E, L) :- isLabel(L), bagContains(E, L), not bagContainsOtherThan(E, L),
contributedTo(E).
bagContainsOtherThan(E, L) :- isLabel(L), bagContains(E, L), bagContains(E, L2), L2 <> L.

hasMinLabel(c, L) :- hasLabel(e, L).
hasMinLabel(d, L) :- hasLabel(e, L).
hasLabel(c, L) v hasLabel(d, L) :- hasLabel(e, L).
hasLabel(E, fs) :- hasMinLabel(E, fs).

hasMaxLabel(a, L) :- hasLabel(d, L).
hasMaxLabel(b, L) :- hasLabel(d, L).
hasLabel(a, L) v hasLabel(b, L) :- hasLabel(d, L).
hasLabel(X, pd) v hasLabel(X, fd) v hasLabel(X, ps) v hasLabel(X, fs) :- hasMaxLabel(X,
fs).

:- hasLabel(E, X), hasLabel(E, Y), X <> Y.

positiveBag(E) :- bagContains(E, ps), contributedTo(E).
positiveBag(E) :- bagContains(E, fs), contributedTo(E).
negativeBag(E) :- bagContains(E, pd), contributedTo(E).
negativeBag(E) :- bagContains(E, fd), contributedTo(E).
hasLabel(E, fs) :- bagContains(E, fs), not negativeBag(E), contributedTo(E).
hasLabel(E, fd) :- bagContains(E, fd), not positiveBag(E), contributedTo(E).
hasLabel(E, cf) :- positiveBag(E), negativeBag(E), contributedTo(E).

:- hasLabel(E, cf).

% predicates for preferences
w(ng, 0) :- hasLabel(g, fs).
w(ng, 0) :- hasLabel(g, ps).
w(ni, 0) :- hasLabel(i, fs).
w(ni, 0) :- hasLabel(i, ps).

w(ng, 1) :- not hasLabel(g, fs), not hasLabel(g, ps).
w(ni, 1) :- not hasLabel(i, fs), not hasLabel(i, ps).

max(ng, 2).
max(ni, 2).

w(n, S) :- w(ni, Si), w(ng, Sg), S = Sg + Si2, Si2 = Si * 2.
max(n, 4).

:   w(n, C). [C:1]
```

Table 3.24: The logic program that corresponds to the goal model in figure 3.7 and the preferences given in this section.

```
model 1
Best model:  {hasLabel(e,fs), hasLabel(d,fs),
hasLabel(c,fs), hasLabel(h,ps), hasLabel(a,fs),
hasLabel(f,pd), hasLabel(b,fd), hasLabel(g,pd),
hasLabel(i,ps) }
model 2
Best model:  {hasLabel(e,fs), hasLabel(d,fs),
hasLabel(c,fs), hasLabel(h,ps), hasLabel(a,fs),
hasLabel(f,pd), hasLabel(b,pd), hasLabel(g,pd),
hasLabel(i,ps)}
```

Table 3.25: Answer sets returned by the ASP engine for the program depicted in table 3.24. The answer sets are filtered by the predicate `hasLabel`.

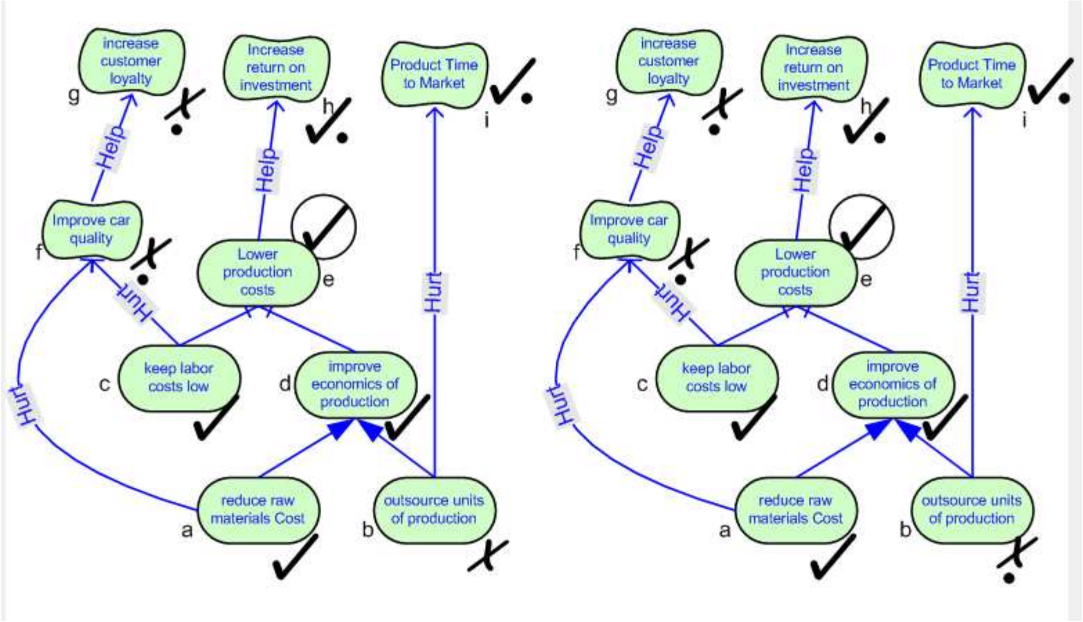

increase customer loyalty
g
Increase return on investment
h
Product Time to Market
i
Help
Help
Hurt
Improve car quality
f
Lower production costs
e
Hurt
Hurt
keep labor costs low
c
improve economics of production
d
reduce raw materials Cost
a
outsource units of production
b

Figure 3.8: The alternatives corresponding to the answer sets given in table 3.25.

The corresponding alternatives to the answer sets given in table 3.25 are shown in figure 3.8.

3.6 Organizational strategies as Game strategies

Each company's mission is to maximize its own achievements. The mission of the company can be formulated as a preference formulae; a company's mission might be to increase its profit regardless of the environmental effects that its process imposes, whereas other companies might specifically want to prioritize environmental issues over their profit to some extent. Due to limited resources, there's always competitions involved in business and thus only the companies that follow the right path can survive in terms of satisfying their preferences. The *score* of a company in this competitive game can be considered to be the level to which the company is able to satisfy its preferences. As can be seen in the following paragraphs, our approach to organizational strategic preferences provides the ground for quantifying the success of a company and also enables us to use game theoretic quantitative theorems in organizational domain.

Definition 3.6-1 (**Strategic Score**). Given G as an organizational goal model involving the set of agents $A = \{a_1, a_2, \dots a_n\}$ where each $a_i (1 \leq i \leq n)$ has a single agent preference formula called ψ_a and an admissible weight function $w_i(o)$, the score of an agent $a_i \in A$ w.r.t. an alternative o is identified by $s(a_i, o)$ and is the value of the weight function for the current alternative: $s(a_i, o) = w_i(o)$

In a competitive corporate environment, each party tries to maximize its own score at the expense of others; it can also be the case that several parties form a cooperative group to maximize their collective preferences.

An important aspect of real world organizational environments is that they are dynamic: each organization seeks better strategies at each point of time and in terms of goal models, might modify its solution alternative or even change its goal model to cope with the constant ongoing competition.

Now, assuming that we have the complete goal model of an agent or in other words the rules of the game are completely defined for the agent, then what would be the best game that an agent can play? and moreover, is there an equilibrium state where each agent is playing its best game assuming that other agents are playing their best games?

Definition 3.6-2 (**Strategic Equilibrium).** Given goal model G that involves agents $\{a_i : 1 \leq i \leq n\}$ with single agent formulas $\psi_{a_i} : 1 \leq i \leq n$ and score functions $s_{a_i} : 1 \leq i \leq n$, an alternative solution (game) is considered a strategic equilibrium if each agent can't gain more score given that other agents don't change their strategy.

The value of finding strategic equilibriums in strategic organizational modeling is in that they can be regarded as the potential *outcome* of the competition. This metric can be used to evaluate how a change in an organization's strategy would actually work in the marketplace: If the change leads to an equilibrium where the goal of the designer is better met (for example environmental issues are better addressed) then that change can potentially regress the market towards the designer's goal. After characterizing strategic equilibriums and given our computational framework, our next target is to compute these states.

The most appealing set of strategies (games) are those that to some sort maximize the overall and individual score of agents. More specifically, we look at the games (o) that maximize the sum of agents' scores: $\sum_{i=1}^{n} s_{a_i}(o)$.

Theorem 3.6-3 (**Existence of Strategic Equilibriums).** *Given strategic game* $\chi =< G, A, S >$ *where* G *is a goal model involving the set of agents* $A = \{a_1, a_2, \ldots, a_n\}$ *with strategic score functions* $S = \{s_{a_1}, \ldots, s_{a_n}\}$*, every solution* o_{max} *that maximizes* $s^*(o) = \sum_{i=1}^{n} s_{a_i}(o)$ *corresponds to a strategic equilibrium.*

Proof: *Proof by contradiction: If agent a can play a better strategy while other agents' strategies are fixed, then the new set of strategies for all agents have a higher value for the function* $s^*(o) = \sum_{i=1}^{n} s_{a_i}(o)$*. However, this is in contradiction with the assumption that* o_{max} *is a solution with maximum value for* $s^*(o)$*.* □

It follows that we can compute the a subset of equilibrium states by finding solutions that maximize the value of function $s^*(o)$

Proposition 3.6-4. *Given strategic game* $\chi =< G, A, S >$, *answer sets with minimum value for the function* $s^{*^-}(o) = -s^*(o)$ *correspond to solutions that are strategic equilibriums.*

Chapter 4

Conclusion and Future Work

In this work, we have revisited evaluation in i^* models and provided extensions to the available evaluation use cases. The extended use cases let the requirements analyst *navigate through alternative solutions* by translating user's desired labels for high level goals along with the model structure into logic rules and using an ASP (answer set programming) engine to return all the minimal answer sets for the program consisting of these rules. An i^* goal model analysis tool such as OpenOME can be integrated with our framework and translate the solutions provided by the framework back into visual representation for the user. In the process of finding the alternative evaluations for high level desired labels, the analyst can specify *preferences* among them and the framework accordingly creates logic rules that reflect these preferences and first returns the most preferred solutions. The framework also enables the analyst to know the deterministic partial solutions resulting from her inputs by running cautious queries on a certain subset of the model such as a neighborhood of the analyst's last modification point.

We have also provided computational machinery for automated support of these use cases where possible. For this purpose, the constraints and objectives of the problems in each use case is translated to logic rules and fed into an answer set programming engine; ultimately, the framework receives the computed answers from the ASP engine. A front-end application such as OpenOME can visualize the answers for users by translating the answers back to its visual elements of i^* model. At its current form, the framework is capable of translating i^* models into logic programs using the rule schemas proposed throughout the paper. In future work, we intend to incorporate the current implementation as well as the UI prototypes provided in chapter 2 into an OpenOME plug-in to visually interact with the user when running the use cases. The current implementation is being tested on moderate size goal models with less than 30 nodes and it's our intention to test its performance against larger goal graphs. Also the range of goal graphs that are being tested need to be expanded to include special cases such as goal graphs with cycles, etc.

The scope of this work doesn't span introducing new refinements for better fulfillment of high level goals or changing the structure of the goal model, however it provides grounds for having the notion of *structure changing actions* that represent the act of changing the structure of the goal model (such as adding or removing agents, goals, etc...) and thus *proposing* new refinements (such as new dependencies, decompositions, etc) which could potentially provide a better solution than the existing ones. The actions that change the structure of the model can be learned from real business experiences.

One possible approach for minimizing human intervention when resolving conflicts in the propagation process is to ask the analyst to specify priorities among incoming contribution links to nodes and when a conflict arises, favor the contributed label from the incoming links with highest priority over other conflicting labels. The implementation of this approach should be straightforward , however there needs to be experimental evaluations to assess its feasibility in real case requirements analysis scenarios. The idea is to factor human's knowledge of the domain in the form of priorities among contributions to elements in the model. On the other hand, the computational machinery provided in this work can be extended to support the use cases required for this approach, if it's found useful.

It is also noteworthy to mention that despite our concentration on i^* evaluation procedure, the framework has the potential to be used for other evaluation procedures of i^* or even other types of goal models as well, as long as the evaluation procedure and the model structure can be encoded using logic rules.

Our view of organizational strategies as games can be potentially extended by characterizing more strategic equilibriums than the specific case mentioned in this work or even all of them. Also, one can view the game being played among cooperative agents where several parties form a cooperative group to maximize their collective preferences as opposed to their individual scores.

Bibliography

[1] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Strong and weak constraints in disjunctive datalog. In *LPNMR '97: Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 2–17, London, UK, 1997. Springer-Verlag.

[2] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. In *6IWSSD: Selected Papers of the Sixth International Workshop on Software Specification and Design*, pages 3–50, Amsterdam, The Netherlands, The Netherlands, 1993. Elsevier Science Publishers B. V.

[3] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements engineering for trust management: Model, 2006.

[4] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal reasoning techniques for goal models, 2004.

[5] Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Reasoning with goal models. In Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors, *ER*, volume 2503 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2002.

[6] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. Sat-based answer set programming. In *AAAI*, pages 61–66, 2004.

[7] Jennifer Marie Horkoff. *Using i* Models for Evaluation.* PhD thesis, Unviersity of Toronto, 2006.

[8] Chung L., Nixon B.A., Yu E., and Mylopoulos J. *Non-Functional Requirements in Software Engineering (Monograph).* Kluwer Academic Publishers, 2000.

[9] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[10] John Mylopoulos. Information modeling in the time of the revolution. *Inf. Syst.*, 23(3-4):127–155, 1998.

[11] Allen Newell and H. A. Simon. Gps, a program that simulates human thought. pages 279–293, 1995.

[12] Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Simple and minimum-cost satisfiability for goal models. In Anne Persson and Janis Stirna, editors, *CAiSE*, volume 3084 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2004.

[13] Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *TPLP*, 6(5):559–607, 2006.

[14] Axel van Lamsweerde. Requirements engineering in the year 00: a research perspective. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 5–19, New York, NY, USA, 2000. ACM.

[15] Eric S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *RE*, pages 226–235. IEEE Computer Society, 1997.

[16] Eric Siu-Kwong Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada, 1996.

[17] Yijun Yu. Openome, an open-source requirements engineering tool. http://www.cs.toronto.edu/km/openome/.