PhD Dissertation

ICT International Doctoral School
Department of Information Engineering and Computer Science
University of Trento

Efficient Reasoning with Constrained Goal Models

Chi Mai Nguyen

Advisor:
Prof. Roberto Sebastiani
University of Trento

Co-Advisor:
Prof. John Mylopoulos
University of Trento

April 2017

Abstract

GOAL models have been widely used in Computer Science to represent software requirements, business objectives, and design qualities. Existing goal modelling techniques, however, have shown limitations of expressiveness and/or tractability in coping with complex real-world problems.

In this work, we exploit advances in automated reasoning technologies, notably *Satisfiability and Optimization Modulo Theories (SMT/OMT)*, and we propose and formalize:

(i) an extended modelling language for goals, namely the *Constrained Goal Model (CGM)*, which makes explicit the notion of *goal refinements* and of *domain assumptions*, allows for expressing *preferences* between goals and refinements, and allows for associating *numerical attributes* to goals and refinements for defining *constraints* and *optimization goals* over multiple *objective functions*, refinements and their numerical attributes;

(i) a novel set of automated reasoning functionalities over CGMs, allowing for automatically generating suitable *realization* of input CGMs, under user-specified assumptions and constraints, that also maximize preferences and optimize given objective functions.

We are also interested in supporting *software evolution* caused by changing requirements and/or changes in the operational environment of a software system. For example, users of a system may want new functionalities or performance enhancements to cope with growing user population (*requirements evolution*). Alternatively, vendors of a system may want to minimize costs in implementing requirements changes (*evolution requirements*).

We propose to use CGMs to represent the requirements of a system and capture requirements changes in terms of incremental operations on a goal model. Evolution requirements are then represented as optimization goals that minimize implementation costs or customer value. We can then exploit reasoning techniques to derive optimal new specifications for an evolving software system.

We have implemented these modelling and reasoning functionalities in a tool, named CGM-Tool, using the OMT solver OptiMathSAT as automated reasoning backend. Moreover, we have conducted an experimental evaluation on large CGMs to support the claim that our proposal scales well for goal models with thousands of elements. To access our framework usability, we have employed a user-oriented evaluation using enquiry evaluation method.

KEYWORDS. *requirements engineering · goal modelling · SMT/OMT*

Acknowledgements

Undertaking this Ph.D. has been a truly precious experience for me and I owe my thanks to many people, without whose support and guidance, it would not have been possible.

* * ★ * *

FIRST and foremost, I would like to express my special appreciation and thanks to my advisor Professor Roberto Sebastiani who has been a tremendous mentor for me. I would like to thank him for his patience, motivation, and for the continuous support and encouragement he gave me over the past five years and beyond. Without his guidance and constant feedback, this Ph.D. would not have been achievable. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Many thanks also to my co-advisor Professor John Mylopoulous for his support, insightful comments, patience, motivation, and immense knowledge. His courses *"Requirements Engineering and Conceptual Modelling"* has provided me the essential background in this work.

I would especially like to thank Professor Paolo Giorgini, who co-authored the article which is an essential part of my dissertation. I appreciate his precious guidance in the evaluation part of this project, his support in building the tool, and invaluable comments, discussions since the very first days of my Ph.D., as well as his time spending as my thesis committee member.

My sincere thanks also go to the rest of my thesis committee: Professor Tacchella Armando, Professor Fabiano Dalpiaz for taking their time serving as my committee members.

I would like to acknowledge in particular Elda Paja, Mattia Salnitri, and Fatma Basak. Without their contribution, I would not be able to complete the evaluation of my proposed framework. I am also in debt to many M.Sc. students who followed the course Security Engineering 2015-2016 at the University of Trento, as well as to the Doctoral students in the Department of Information Engineering and Computer Science at the University of Trento participated in the user-oriented evaluation studies reported in this dissertation.

I am also grateful to Dagmawi Neway for his technical support in developing CGM-Tool, and Patrick Trentin for assistance with the usage of OptiMathSAT.

I would also like to thanks the staffs Anna Broll, Michela Angeli, Andrea Stenico, and Francesca Belton for helping me with administrative works so that I could channel my time into my research.

Many thanks to all of my friends who supported me in writing, and incented me to strive towards my goal, as well as for the pleasure and nice time which make up my social life.

My deepest thanks go to my family. Words cannot express how grateful I am to my parents for all the sacrifices that they have made on my behalf. Last but not least, I would like to express appreciation to my beloved husband Thanh Binh Cao, who spent sleepless nights with and was always my support.

This research was partially supported by the ERC advanced grant 267856, ‘Lucretius: Foundations for Software Evolutions’, and by SRC GRC Research Project 2012-TJ-2266 WOLF.

Table of Contents

List of Tables

LIST OF FIGURES

FIGURE **Page**

ABREVIATIONS

CGM Constrained Goal Model.

DAG Directed Acyclic Graph.

ISO International Organization for Standardization.

LTL Linear Temporal Logic.

OMT Optimization Modulo Theories.

RCP Rich Client Platform.

RE Requirements Engineering.

SMT Satisfiability Modulo Theories.

SMT/OMT Satisfiability and Optimization Modulo Theories.

CHAPTER 1

INTRODUCTION

This chapter presents the motivation of this dissertation as well as summarizes the major contributions of this work. It also describes the structure of this dissertation.

∗ ∗ ★ ∗ ∗

The concept of goal has long been used as a useful abstraction in many areas of computer science, like for example, artificial intelligence planning, agent-based system, and knowledge management. More recently, software engineering has also been using goals to model requirements for a software system, business objectives for enterprises, and design qualities [Ant96, AP98, DvLF93, VL01, GMNS04].

Goal-oriented Requirements Engineering (RE) approaches have gained popularity for a number of significant benefits in conceptualizing and analyzing requirements [VL01]. Goal models provide a broader system engineering perspective compared to traditional requirements engineering methods, a precise criterion for completeness of the requirements analysis process, and rationale for requirements specification, as well as automated support for early requirements analysis. Moreover, goal models are useful in explaining requirements to stakeholders, and goal refinements offer an accessible level of abstraction for validating choices among alternative designs.

Current goal modelling and reasoning techniques, however, have limitations with respect to expressiveness and/or scalability. Among leading approaches for goal modelling, KAOS offers a very expressive modelling language but the reasoning is undecidable and unscalable. I*, on the other hand, is missing constructs such as preferences, priorities, and optimization goals. Although more recent proposals, such as Techne [JBEM10, LMSM10] offer expressive extensions to goal models, they still lack some features of our proposal, notably optimization goals, and also lack scalable reasoning facilities.

As a result of these deficiencies, no goal modelling framework can express goals such as *"Select which new requirements to implement for the next release, such as to optimize customer value while maintaining costs below some threshold"* and be able to reason about it and generate a specification/solution for it. As another example, consider a situation where a goal model changes and a new specification/solution needs to be generated for the new goal model. In this case, the new specification/solution may be required to fulfil the evolution goal *"Minimize implementation effort"* or *"Maximize user familiarity by changing as little as possible the new functionality of the system relative to the old one"*. In both cases, we are dealing with requirements that are beyond the state-of-the-art for goal modelling and reasoning. As we will discuss in chapter 4, our proposal can accommodate such requirements both with respect to modelling and scalable reasoning.

We are interested in advancing the state-of-the-art in goal models and reasoning by proposing a more expressive modelling language that encompasses many of the modelling constructs proposed in the literature, and at the same time offers sound, complete, and tractable reasoning facilities. We are aiming for a goal modelling in the spirit of the qualitative goal model introduced in [SGM04], rather than a social dependency modelling language, such as i*. To accomplish this, we exploit advances in automated reasoning technologies, notably *Satisfiability Modulo Theories (SMT)* [BSST09] and *Optimization Modulo Theories (OMT)* [ST15a], to propose and formalize an extended notion of goal model, namely *Constrained Goal Models (CGMs).*

CGMs treat (AND/OR) refinements as first class citizens, allowing associated constraints, such as Boolean formulas or SMT/OMT formulas. For instance, when modelling a meeting scheduling system, we may want to express the fact that, to fulfil the *nice-to-have* requirements of keeping the scheduling fast enough (e.g., strictly less than 5 hours) we cannot afford both the time-consuming tasks of performing the schedule manually (3 hours) and of calling the participants on-by-one by phone (2 hours). CGMs provide user-friendly constructs by which we can encode constraints like this, either by adding Boolean formulas on the propositions which label such requirements and tasks (e.g., `FastSchedule`, `ScheduleManually`, and `CallParticipants`), or by associating to those propositions numerical variables (e.g., $\mathsf{workTime}$) and by adding SMT formulas encoding mixed Boolean-arithmetical constraints on those variables and propositions (See chapter 4). To the best of our knowledge, this was not possible with previous goal modelling techniques, including that in [SGM04].

Moreover, we are living in an ever-changing world where the only constant is change. Changes need to be accommodated by any system that lives and/or operates in that world, biological and/or engineered. For software system, this is a well-known problem referred to as *software evolution*. There has been much work and interest on this problem since Lehman's seminal proposal for laws of software evolution [Leh80]. However, the problem of effectively supporting software evolution still accounts for more than 50% of total cost in a software's lifecycle. We are only interested in software evolution caused by changing requirements and/or environmental conditions. We propose to model requirements changes through changes to a CGM model, and evolution requirements as optimization goals, such as *"Minimize costs while implementing new functionalities"*.

Taking advantage of CGMs' formal semantics and the expressiveness and efficiency of current SMT and OMT solvers, we also provide a set of automated reasoning functionalities on CGMs. Especially, on a given CGM, our approach allows for:

(a) the automatic check of the CGM's realizability (i.e. check if the goal model has any solution);

(b) the interactive/automatic search for realizations (i.e., specification/solution) of the CGM;

(c) the automatic search for *"best"* realization in term of penalties/rewards and/or of user-defined preferences;

(d) the automatic search for the realization(s) which optimize given objective functions;

(e) the automatic extraction of the *UNSAT core* (i.e., the self-contradictory part) of a unrealizable CGM (i.e., a goal model that does not have solution);

(f) the automatic search for realization(s) which optimize given evolution requirements.

Our approach is implemented as a tool *(CGM-Tool)*, a standalone java application based on the Eclipse Rich Client Platform (RCP) engine. The tool offers functionalities to create CGM models as graphical diagrams and to explore alternatives scenarios running automated reasoning techniques. CGM-Tool uses the OMT solver OptiMathSAT

[ST15a, ST15c, ST15b], which is built on top of the SMT solver MATHSAT5 [CGSS13], as automated reasoning backend.[1] Our CGM-Tool can cope with goal models an order of magnitude beyond what has been reported in the literature in most cases. In some cases involving optimization goals, e.g., *"minimize development costs for the next release of software product S"*, the CGM-Tool performs more modestly, but can still handle models of size in the hundreds of elements as reported in chapter 9.

1.1 Contributions

The main contributions of this work include:

I. An integration within one modelling framework of constructs that have been proposed in the literature in a piecemeal fashion, specifically,

   (i) Allow for explicit labelling of goal refinements with Boolean propositions that can be interactively/automatically reasoned upon;

   (ii) Provide an explicit representation of domain assumptions to represent preconditions to goals;

   (iii) Allow for Boolean constraints over goals, domain assumptions and refinements;

   (iv) Provide a representation of preferences over goals and their refinements, by distinguishing between mandatory and nice-to-have requirements and by assigning preference weights (i.e., penalties/rewards) to goals and domain assumptions. Alternatively, preferences can be expressed explicitly by setting binary preference relations between pairs of goals or pairs of refinements;

   (v) Assign numerical attributes (e.g., resources like cost, worktime, and room) to goals and/or refinements and define constraints and multiple objective functions over goals, refinements and their numerical attributes.

   (vi) Define optimization goals over numerical attributes, such as cost or customer value;

II. Fully support automated reasoning over CGMs that is both sound and complete, i.e., returns only solutions that are consistent with CGM semantics, and all such solutions;

[1] The OMT solver OptiMathSAT can be used also as an SMT solver if no objective function is set: in such case it works as a wrapper of MATHSAT5.

III. Establish that reasoning with CGM models is scalable with models including thousands of elements.

IV. A proposal for modelling changing requirements in terms of changes to a CGM model.

V. The identification of a new class of evolution requirements, expressed as optimization goals in CGM.

VI. Fully support automated reasoning over changed goal models and evolution requirements.

VII. A full experiment for evaluating the usability of CGM and CGM-Tool using enquiry evaluation method.

1.2 Structure of the Dissertation

Besides motivation and conclusion, the dissertation is organised into three main parts.

- chapter 1 presents the motivation of this project and summarizes the main contribution of the dissertation.

The first part of the dissertation, which focuses on the introduction of the state of the art and research baselines of the project, consists in the following chapters:

- chapter 2 gives overview of the state of the art and related work
- chapter 3 provides a succinct account of necessary background on goal modelling and on SMT/OMT;

The second part of the dissertation presents the main contributions of the dissertation, which are conveyed in the following chapters:

- chapter 4 introduces the notion of CGM through an example;
- chapter 5 introduces the syntax and semantics of CGMs;
- chapter 6 presents the set of automated reasoning functionalities for CGMs;

- chapter 7 introduces the notion of evolution requirements and requirements evolution through a working example, as well as formalizes the problem of automatically handling CGM evolutions and evolution requirements for CGMs;

The third part shows how the framework proposed in the dissertation is implemented and evaluated. This part consists of the following chapters:

- chapter 8 gives a quick overview of our tool based on the presented approach;
- chapter 9 provides an experimental evaluation of the performances of our tool on large CGMs, showing that the approach scales well with respect to CGM size.
- chapter 10 presents the user-oriented evaluation on the CGM-Tool.

Finally, chapter 11 presents conclusions and future research challenges.

1.3 Publications

A significant part of the content of this dissertation has been published in

- Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos.

  Multi-objective reasoning with constrained goal models.

  Requirements Engineering Journal, pages 1–37, 2016

- Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos.

  Requirements Evolution and Evolution Requirements with Constrained Goal Models.

  In *Proceedings of the 37nd International Conference on Conceptual Modeling*, LNCS. Springer, 2016

- Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos.

  Modeling and reasoning on requirements evolution with constrained goal models.

  Software Engineering and Formal Method, 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, 2017.

  submitted

Part I

State of The Art and Research Baselines

CHAPTER 2

STATE OF THE ART

Past studies always have impacts on present works. This chapter gives a quick overview of, and a brief comparison with, some of the state-of-the-art goal-oriented modelling languages.

* * ★ * *

In this chapter, we will present a brief summary of the literature on goal-oriented modelling languages. [Lap05], [JMF08], and [BDHM13] provide better and deeper debates on the requirements modelling languages and the goal-oriented approach, their advantages and limitations.

2.1 KAOS

KAOS stands for *Knowledge Acquisition in Automated Specification* [DvLF93] or *Keep All Objectives Satisfied* [LL04]. "The overall approach taken in KAOS has three components: (i) a conceptual model for acquiring and structuring requirements models, with an associated acquisition language, (ii) a set of strategies for elaborating requirements models in this framework, and (ii) an automated assistant to provide guidance in the acquisition process according to such strategies." [DvLF93]

KAOS offered a concrete methodology for solving the requirements problem. KAOS includes a considerable number of concepts, (such as object, operation, agent, goal, obstacle, requisite, requirement, assumption, scenario), and a process for eliciting goals, refining them, etc. (for example, specialization, refinement, conflict, operationalization, concern, and so on) [DvLF93, vLLD98, vLL00, LL04]. KAOS also came with a rich logical sublanguage including Linear Temporal Logic (LTL) for describing the elements of a KAOS model formally. KAOS, in other words, is a requirement engineering methodology which is defined on top of LTL. In KAOS framework, goals are refined into requirements. A KAOS specification is a collection of the three core models:

- **Goal Model:** represented goals and their assigned agents.
- **Object Model:** a UML model that refer to objects and their properties, which can be derived from formal specifications of goals.
- **Operation Model:** defines various services to be provided by a software agent.

In summary, KAOS supports a rich ontology for requirements that goes well beyond goals, as well as an LTL-grounded formal language for constraints. This language is coupled with a concrete methodology for capturing and analyzing requirements problems. KAOS supports a number of analysis techniques including obstacle, inconsistency, and probabilistic goal analysis. Overall, KAOS is a well-developed methodology with a solid formal framework. KAOS, however, is undecidable, quite *'static'*, and cannot efficiently cope with the possibility of requirements changing during the process. Moreover, unlike our proposal, KAOS does not support nice-to-have requirements and preferences, nor does it exploit SMT/OMT solver technologies for scalability.

2.2 Qualitative Goal Models

Qualitative goal model is introduced in [MCN92]. It is a modelling language that supports qualitative (strong and weak) evidence both in favour and against propositional goal. In [GMNS04], the model is formalized by replacing each proposition g, standing for a goal, by four propositions (FS_g, PS_g, PD_g, FD_g) representing full (and partial) evidence for the satisfied (and denial) faction of g. A traditional implication such as $p \wedge q \rightarrow r$ is then translated into a series of implications connecting these new symbols, including $FS_p \wedge FS_q \rightarrow FS_r$, $PS_p \wedge PS_q \rightarrow PS_r$, as well as $FD_p \rightarrow FD_r$, $FD_q \rightarrow FD_r$, etc. The conflict between a and b is captured by axioms of the form $FS_a \rightarrow FD_b$, and it is consistent to have both FS_a and FS_a evaluated to true at the same time. As a result, even though the solution to the model is a classical propositional theory, there is no inconsistency that can cause the whole model to be inferred. In fact, a predicate g can be assigned a subset of truth values $\{FS, PS, FD, PD\}$.

[SGM04] extended the approach further by including axioms for avoiding conflicts of the form $FS_a \wedge FD_a$. The approach recognized the need to formalize goal models so as to automatically evaluate the satisfaction of goals. The goal models are defined as AND/OR graphs, in which nodes are goals, and a number of relations is provided to indicate if the interaction is positive or negative, as well as to specify the strength of the interaction. These goal models, however, do not incorporate the notion of conflict as inconsistency, they do not include concepts other than goals, cannot distinguish optional from mandatory requirements and have no notion of a robust solution, i.e. solution without "conflict", where a goal can not be (full or partial) denied and (full or partial) satisfied at the same time.

In summary, this approach does support scalable reasoning using SAT-solving techniques. Our proposal subsumes this work in many ways, including a more expressive language and much more advanced SMT/OMT-solving technology.

There is one construct of [SGM04, GMNS04] that was left our of the CGM language: + and − contributions from goals to goals. There are several reasons for this decision. In (un-constrained) goal models, formalizing (+,−) contributions requires a 4-value logic (fully/partially satisfied/denied). In principle our CGM framework could be extended to such a logic, with the following drawbacks:

(a) The size of the Boolean search space would extend from 2^N to 4^N. Given that reasoning functionality in this dissertation are much more sophisticated and computationally more demanding than those in [SGM04, GMNS04], this might drastically reduce the efficiency of the approach.

(b) Unlike standard 2-value logic, which allows us to give a clear semantics of “realization”, without any vagueness, it is not obvious to us what a “realization” could be in 4-value logic. (E.g, should realization admit partially satisfied/denied tasks/ requirements/assumptions? If yes, how should a user interpret a partially-satisfied/denied requirement/task/assumption in a realization returned by the system? In which sense a realization involving partial values can be considered “optimal" or “optimum”?)

There are other differences between the two proposals. In CGMs, we have made AND/OR-decompositions explicit by making refinement a first class citizen that can be named and talked about (as discussed in chapter 4). Moreover, unlike with [SGM04, GMNS04], we have a backbone AND/OR Directed Acyclic Graph (DAG), where arbitrary constraints can be added. This DAG is such that a non-leaf goal is equivalent to the disjunction (“or”) of its refinements, and each refinement is equivalent to the conjunction (“and”) of its source goals. Relation edges, constraints and assertions further constrain this structure.

2.3 I^* and Tropos

I^* is introduced in [Yu97]. I^* is a modelling language that focuses on the interdependencies of actors within a socio-technical system. In i^*, a goal is related to the organization context. I^* provides two models: the Actor Strategic Dependency Model (SD model) and the actor Strategic Rationale Model (SR model). Typically, SD models are used to analyze the changes in the structure due to the introduction of the system-to-be, whilst SR models are used to explore the rationale behind the processes in the system and organizations.

One of i^* significant advantages is its ability to communicate with the stakeholders thanks to its easy to learn non-formal character. I^*, however, has no notion of conflict. I^* does not provide the concepts to capture preferences, mandatory / optional requirements either. In short, I^* is a lightweight modelling language, intended for early stages of requirements analysis, and did not support formal reasoning until recent thesis work by [Hor12].

Tropos [CKM02] is a requirements-driven agent-oriented development methodology that uses i^* modelling framework as the base. In the development of agent-based system., the Tropos methodology can be used from the early requirements analysis through architectural design and requirements and detailed design to the implementation. I^* modelling framework is used to model and reason about requirements and system configuration choices. Formal Tropos [FLM$^+$04] is a formal specification language that adds constraints, invariants, pre- and post-conditions to Tropos. Tropos and Formal Tropos model can be validated by model-checking. The main deficiency of this work relative to our proposal is that Formal Tropos is expressive but not scalable.

2.4 Techne and Liaskos

Techne [JBEM10] is a recent proposal for a family of goal-modelling languages that supports nice-to-have goals and preferences, but it is strictly propositional and uses hand-crafted algorithms, and therefore does not support optimization goals. [EMBJ10] constitutes a first attempt to reason with nice-to-have requirements (aka preferences). The scalability experiments conducted used the SAT solver of Sebastiani et al. [SGM04] and added local search algorithms to deal with preferences. All experiments were conducted on a model with about 500 elements and the search algorithms returned maximal consistent solution but also near-solutions. [EBJ11] focuses on finding new solutions for a goal model that has changed (new goals were added/removed), such that the change minimizes development effort (EvoR1) or maximizes familiarity (EvoR2). Notice that EvoR1, EvoR2 are evolution requirements. The paper uses a Truth-Maintenance System (TMS) and builds algorithms on top for finding solutions to EvoR1, EvoR2 that “repair” the previous solution and construct a new one. The search algorithms would need to be redone if we used different evolution algorithms, unlike the CGM tool where you can formally express EvoR1, EvoR2 or variants, and search is handled by the backend OMT/SMT solver. [EBMJ12, EBJM14] continue the study of reasoning with Techne models and use SAT solvers and hand-crafted search algorithms to establish scalability for models size O(1K). Nevertheless, the resulting tools from this work still can’t handle quantitative optimization problems and other features of CGMs.

Liaskos [LMSM10, Lia12] has proposed extensions to qualitative goal models to support nice-to-have goals and preferences, as well as decision-theoretic concepts such as utility. This proposal is comparable to our proposal in this paper, but uses AI reasoners for reasoning (AI planners and GOLOG) and, consequently, does not scale very well relative to our proposal.

2.5 Feature Models and Search-Based Software Engineering

Feature models [CBH11] share many similarities with goal models: they are hierarchically structured, with AND/OR refinements, constraints, and attributes. However, each feature represents a bundle of functionality or quality and as such, feature models are models of software configurations, not requirements. Moreover, reasoning techniques for feature models are limited relative to their goal model cousins.

Search-Based Software Engineering. Scalable reasoning for optimization problems has been studied by Harman et al in the context of formalizing and solving the next release problem [ZHM07]: given a set of preferences with associated cost and customer value attributes, select a subset of preferences to be included in the next release that optimizes given attributes. That work uses genetic algorithms and other search techniques that may return close-to-optimal solutions and use heuristics (meaning that reasoning is not complete).

CHAPTER 3

Research Baselines

This chapter provides some preliminary background of this dissertation. As prerequisite knowledge, we assume that the reader is familiar with the syntax and semantics of standard Boolean logic and of linear arithmetic over the rationals.

∗ ∗ ★ ∗ ∗

Our research baseline consists of our previous work on qualitative goal models and of Satisfiability and Optimization Modulo Theories (SMT and OMT respectively). The aim of this chapter is to introduce the necessary background notions on these topics.

3.1 Goal Models

Qualitative goal models are introduced in [MCN92], where the concept of goal is used to represent respectively a functional and non-functional requirement in terms of a proposition. A goal can be refined by means of AND/OR refinement relationships and qualitative evidence (strong and weak) for/against the fulfilment of a goal is provided by contribution links labelled $+, -$ etc. An example of the approach goal graph is showed in Figure 3.1. In [GMNS04], goal models are formalized by replacing each proposition g, standing for a goal, by four propositions (FS_g, PS_g, PD_g, FD_g) representing full (and partial) evidence for the satisfaction/denial of g. A traditional implication such as

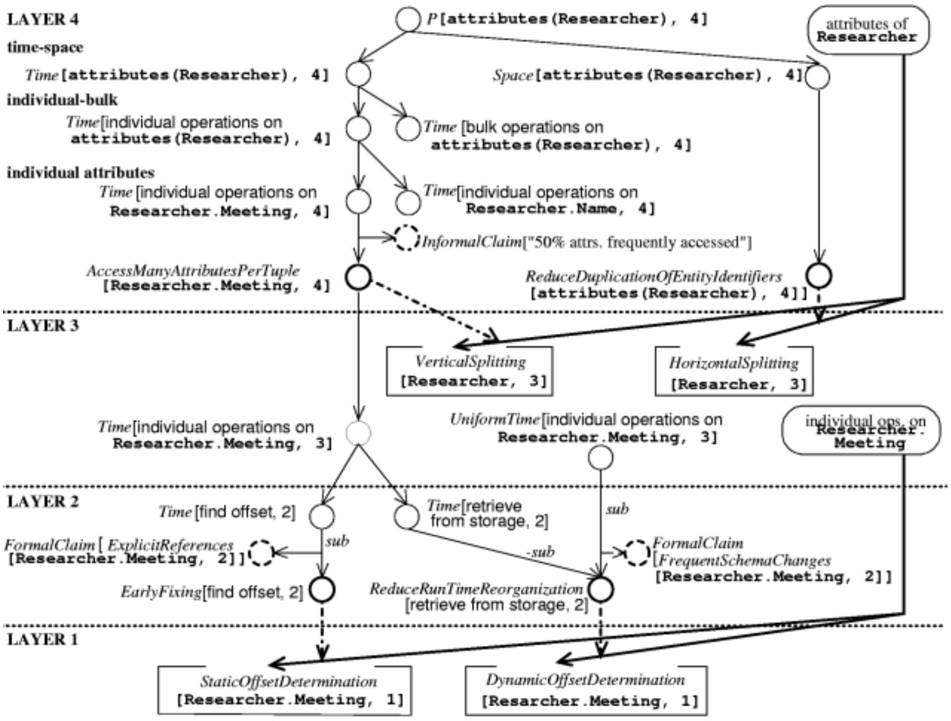

LAYER 4
time-space
individual-bulk
individual attributes
LAYER 3
LAYER 2
LAYER 1
P[attributes(Researcher), 4]
attributes of Researcher
Time[attributes(Researcher), 4]
Space[attributes(Researcher), 4]
Time[individual operations on attributes(Researcher), 4]
Time [bulk operations on attributes(Researcher), 4]
Time [individual operations on Researcher.Meeting, 4]
Time[individual operations on Researcher.Name, 4]
InformalClaim["50% attrs. frequently accessed"]
AccessManyAttributesPerTuple [Researcher.Meeting, 4]
ReduceDuplicationOfEntityIdentifiers [attributes(Researcher), 4]]
VerticalSplitting [Researcher, 3]
HorizontalSplitting [Resarcher, 3]
Time[individual operations on Researcher.Meeting, 3]
UniformTime[individual operations on Researcher.Meeting, 3]
individual ops. on Researcher. Meeting
Time[find offset, 2]
Time[retrieve from storage, 2]
sub
-sub
FormalClaim[ExplicitReferences [Researcher.Meeting, 2]]
FormalClaim [FrequentSchemaChanges [Researcher.Meeting, 2]]
EarlyFixing[find offset, 2]
ReduceRunTimeReorganization [retrieve from storage, 2]
StaticOffsetDetermination [Researcher.Meeting, 1]
DynamicOffsetDetermination [Resarcher.Meeting, 1]

Figure 3.1: A performance goal graph of qualitative goal model as showed in [MCN92].

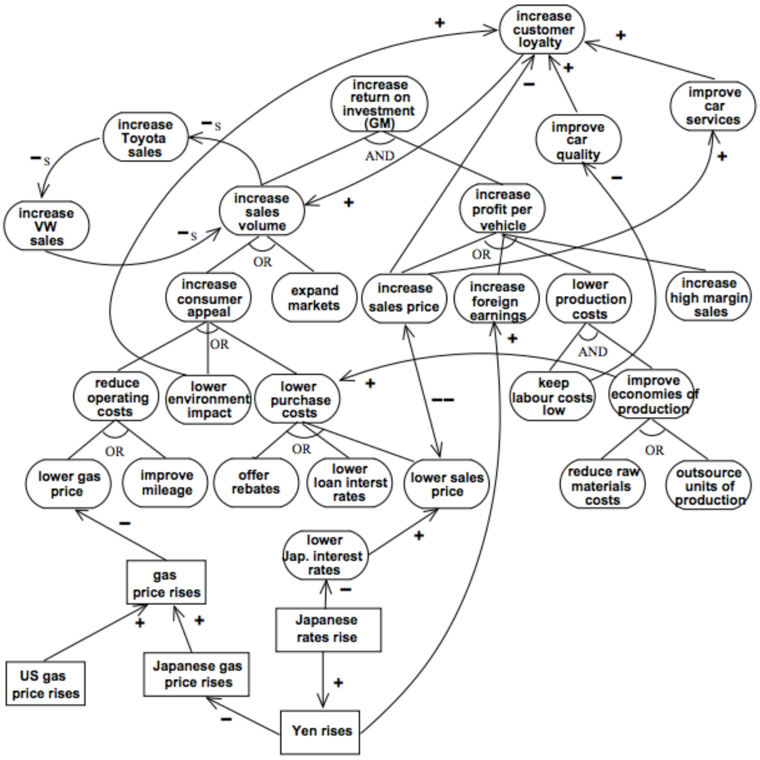


Figure 3.2: A partial and fictitious goal model of a US car manufacture (GM) as showed in [GMNS04].

$(p \wedge q) \rightarrow r$ is then translated into a series of implications connecting these new symbols, including $(FS_p \wedge FS_q) \rightarrow FS_r$, $(PS_p \wedge PS_q) \rightarrow PS_r$, as well as $FD_p \rightarrow FD_r$, $FD_q \rightarrow FD_r$, etc. The conflict between goals a and b is captured by axioms of the form $FS_a \rightarrow FD_b$, and it is consistent to have both FS_a and FD_a evaluated to true at the same time. As a result, even though the semantics of a goal model is a classical propositional theory, inconsistency does not result in everything being true. In fact, a predicate g can be assigned a subset of truth values $\{FS, PS, FD, PD\}$. An example of this approach goal model is showed in Figure 3.2.

[SGM04] extended the approach further by including axioms for avoiding conflicts of the form $FS_a \wedge FD_a$. The approach recognized the need to formalize goal models so as to automatically evaluate the satisfiability of goals. These goal models, however, do not incorporate the notion of conflict as inconsistency, they do not include concepts other than goals, cannot distinguish “nice-to-have” from mandatory requirements and have no notion of a robust solution, i.e. solution without "conflict", where a goal can not be (fully or partially) denied and (respectively, fully or partially) satisfied at the same time.

3.2 Satisfiability Modulo Theories and Optimization Modulo Theories

3.2.1 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) is the problem of deciding the satisfiability of a quantifier-free first-order formula Φ with respect to some decidable theory $\mathcal{T}$ (see [Seb07, BSST09]). In this paper, we focus on the theory of *linear arithmetic over the rationals, $\mathcal{LRA}$*: SMT($\mathcal{LRA}$) is the problem of checking the satisfiability of a formula Φ consisting in atomic propositions $A_1, A_2, ...$ and linear-arithmetic constraints over rational variables like “$(2.1x_1 - 3.4x_2 + 3.2x_3 \leq 4.2)$”, combined by means of Boolean operators $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. (Notice that a Boolean formula is also a SMT($\mathcal{LRA}$) formula, but not vice versa.) An *$\mathcal{LRA}$-interpretation* μ is a function which assigns truth values to Boolean atoms and rational values to numerical variables; μ *satisfies* Φ in $\mathcal{LRA}$, written “$\mu \models \Phi$” –aka, μ is a *solution* for Φ in $\mathcal{LRA}$– iff μ makes the formula Φ evaluate to true; Φ is $\mathcal{LRA}$-satisfiable iff it has at least one $\mathcal{LRA}$-interpretation μ s.t. $\mu \models \Phi$.

A *SMT Solver* for a theory $\mathcal{T}$, $\mathcal{T}$-solver, is a procedure able to decide the $\mathcal{T}$-satisfiability ($\mathcal{T}$-consistency) of a conjunction/set μ of $\mathcal{T}$-literals. Modern $\mathcal{T}$-solvers support several features which are relevant to SMT($\mathcal{T}$).

Below is an example of SMT problem in SMT-LIBv2 format.

SMT-LIBv2 Example

```
(declare-fun x () Int)
(declare-fun y1 () Int)
(declare-fun y2 () Int)
(declare-fun z () Int)
(assert (= x y1))
(assert (not (= y1 z)))
(assert (= x y2))
(assert (and (> y2 0) (< y2 5)))
(check-sat)
(get-value (x z)
```

Here, four integer variables x, $y1$, $y2$ and z were declared. Four SMT constraints were formulated: $(x = y1)$, $\neg(y1 = z)$, $(x = y2)$, and $(y2 > 0) \wedge (y2 < 5)$. The last two rows of code asks for the satisfiability of the problem and return the values of x and z variable in case of SAT. In other words, our SMT problem is to find if the formula

$$(x = y1) \wedge \neg(y1 = z) \wedge (x = y2) \wedge ((y2 > 0) \wedge (y2 < 5)$$

satisfiable (x, $y1$, $y1$, z are integer) and in case of SAT, find the value of the variable x and variable z. One possible result of the problem is $(x = 2)$ and $(z = 10)$.

3.2.2 Optimization Modulo Theories

An *Optimization Modulo Theories over $\mathscr{LRA}$ (OMT($\mathscr{LRA}$))* problem $\langle \Phi, \langle obj_1, ..., obj_k \rangle \rangle$ is the problem of finding solution(s) to an SMT($\mathscr{LRA}$) formula Φ which optimize the rational-valued objective functions $obj_1, ..., obj_k$, either singularly or lexicographically [NO06, ST12, ST15a, ST15c]). A solution *optimizes lexicographically* $\langle obj_1, ..., obj_k \rangle$ if it optimizes obj_1 and, if more than one such obj_1-optimum solutions exists, it also optimizes obj_2,..., and so on.

Very efficient SMT($\mathscr{LRA}$) and OMT($\mathscr{LRA}$) solvers are available, which combine the power of modern SAT solvers with dedicated linear-programming decision and minimization procedures (see [Seb07, BSST09, CGSS13, NO06, ST12, ST15a, ST15c, ST15b]). For instance, in the empirical evaluation reported in [ST15a] the OMT($\mathscr{LRA}$)

solver OptiMathSAT [ST15a, ST15b] was able to handle optimization problems with up to thousands Boolean/rational variables in less than 10 minutes each.

Below is an example of OMT problem in SMT-LIBv2 format.

SMT-LIBv2 Example

```
(declare-fun x () Int)
(declare-fun y1 () Int)
(declare-fun y2 () Int)
(declare-fun z () Int)
(assert (= x y1))
(assert (not (= y1 z)))
(assert (= x y2))
(assert (and (> y2 0) (< y2 5)))
(minimize (+ x z))
(check-sat)
(get-value (x z))
```

The problem is similar to the example presented in subsection 3.2.1 excepted that this problem asked for the minimization of the sum $(x+z)$. The solution of the problem is $(x=1)$, $(z=0)$

Part II

Contributions

CHAPTER 4

CONSTRAINED GOAL MODELS (CGMS)

One main contribution of this dissertation is the formalization of the Constrained Goal Models (CGMs). This chapter introduces the main ideas of CGMs and the main functionalities of our CGM-Tool through a meeting scheduling example.

* * ★ * *

In this chapter, we introduce the notions of constrained goal model (CGM), and of realization of a CGM; we also present the automated-reasoning functionalities of our CGM-Tool through a meeting scheduling example (Figure 4.5), without getting into the formal details yet. The formal semantics and reasoning of CGM will be presented in chapter 5 and chapter 6.

4.1 The Backbone: Goals, Refinements, and Domain Assumptions

We model the requirements for a meeting scheduling system, including the functional requirement `ScheduleMeeting`, as well as non-functional/quality requirements `LowCost`, `FastSchedule`, `MinimalEffort` and `GoodQualitySchedule`. They are represented as root goals.

Notationally, round-corner rectangles (e.g., ScheduleMeeting) are root goals, representing stakeholder *requirements*; ovals (e.g. CollectTimetables) are *intermediate goals*; *hexagons* (e.g. CharacteriseMeeting) are *tasks*, i.e. non-root leaf goals; rectangles (e.g., ParticipantsUseSystemCalendar) are *domain assumptions*. We call *elements* both goals and domain assumptions.

Figure 4.1 shows all the elements of the Schedule Meeting CGM.

Labeled bullets at the merging point of the edges connecting a group of source elements to a target element, as showed in Figure 4.2, are *refinements* (e.g., (GoodParticipation,MinimalConflict) $\xrightarrow{R_{20}}$ GoodQualitySchedule), while the R_is denote their labels.

Remark 1. *Unlike previous goal modelling proposals, refinements are explicitly labeled, so that stakeholders can refer to them in relations, constraints and preferences. (This fact will be eventually discussed with more details.) The label of a refinement can be omitted when there is no need to refer to it explicitly.*

Intuitively, requirements represent desired states of affairs we want the system-to-be to achieve (either mandatorily or preferably); they are progressively refined into intermediate goals, until the process produces actionable goals (tasks) that need no further decomposition and can be executed; domain assumptions are propositions about the domain that need to hold for a goal refinement to work. Refinements are used to represent alternatives of how to achieve a non-leaf element, i.e., a refinement of an element represents one of the alternative of sub-elements that are necessary to achieve it.

The principal aim of the CGM in Figure 4.2 is to achieve the requirement ScheduleMeeting, which is *mandatory*. (A requirement is set to be mandatory by means of user assertions, see below.) ScheduleMeeting has only one candidate refinement R_1, consisting in five sub-goals: CharacteriseMeeting, CollectTimetables, FindASuitableRoom, ChooseSchedule, and ManageMeeting. Since R_1 is the only refinement of the requirement, all these sub-goals must be satisfied in order to satisfy it. There may be more than one way to refine an element; e.g., CollectTimetables is further refined either by R_{10} into the single goal ByPerson or by R_2 into the single goal BySystem. Similarly, FindASuitableRoom and ChooseSchedule have three and two possible refinements respectively. The subgoals are further refined until they reach the level of domain assumptions and tasks.

The requirements that are not set to be mandatory are "*nice-to-have*" ones, like `LowCost`, `MinimalEffort`, `FastSchedule`, and `GoodQualitySchedule` (in blue in all mentioned above figures). They are requirements that we would like to fulfil with our solution, provided they do not conflict with other requirements.

4.2 Boolean Constraints: Relation Edges, Boolean Formulas and User Assertions.

Importantly, in a CGM, elements and refinements are enriched by user-defined *Boolean constraints*, which can be expressed either graphically as *relation edges*, or textually as *Boolean or SMT($\mathcal{LRA}$) formulas*, or as *user assertions*.

4.2.1 Relation Edges

As showed in Figure 4.3, we have four kinds of relation edges.

- Contribution edges "$E_i \xrightarrow{++} E_j$" between elements (in green in Figure 4.3), like "`ScheduleAutomatically` $\xrightarrow{++}$ `MinimalConflicts`", mean that if the source element E_i is satisfied, then also the target element E_j must be satisfied (but not vice versa).

- Conflict edges "$E_i \xleftrightarrow{--} E_j$" between elements (in red), like "`ConfirmOccurrence` $\xleftrightarrow{--}$ `CancelMeeting`", mean that E_i and E_j cannot be both satisfied.

- Refinement bindings "$R_i \longleftrightarrow R_j$" between two refinements (in purple), like "$R_2 \longleftrightarrow R_7$", are used to state that, if the target elements E_i and E_j of the two refinements R_i and R_j, respectively, are both satisfied, then E_i is refined by R_i if and only if E_j is refined by R_j. Intuitively, this means that the two refinements are bound, as if they were two different instances of the same global choice.

  For instance, in Figure 4.3, the refinements R_2 and R_7 are bound because such binding reflects a global choice between a manual approach and an automated one.

- Preferred edges "$E_i \succeq E_j$" between elements (in light brown), like "`UsePartnerInstitutions` $\succeq$ `UseLocalRoom`", mean that we would prefer a solution with E_i (and with or without E_j) over a solution which has E_j but not E_i.

4.2.2 Boolean Formulas

It is possible to enrich CGMs with Boolean formulas, representing arbitrary constraints on elements and refinements, as showed in Figure 4.4. Such constraints can be either *global* or *local to elements or to refinements*, that is, each goal G can be tagged with a pair of *prerequisite* formulas $\{\phi_G^+, \phi_G^-\}$ –called *positive* and *negative* prerequisite formulas respectively– so that ϕ_G^+ [resp. ϕ_G^-] must be satisfied when G is satisfied [resp. denied]. (The same holds for each requirement R.)

For example, to require that, as a prerequisite for `FastSchedule`, `ScheduleManually` and `CallParticipants` cannot be both satisfied, one can add a constraint to the positive prerequisite formula of `FastSchedule`:

$$(4.1) \qquad \phi^+_{\texttt{FastSchedule}} \overset{\text{def}}{=} \quad ... \wedge \neg(\texttt{ScheduleManually} \wedge \texttt{CallParticipants}),$$

or, equivalently, add globally to the CGM the following Boolean formula:

$$(4.2) \qquad \texttt{FastSchedule} \quad \rightarrow \quad \neg(\texttt{ScheduleManually} \wedge \texttt{CallParticipants}).$$

Notice that there is no way we can express (4.1) or (4.2) with the relation edges above.

4.2.3 User Assertion

With CGM-Tool, one can interactively mark [or unmark] requirements as satisfied (true), thus making them mandatory (if unmarked, they are nice-to-have ones). In our example `ScheduleMeeting` is asserted as true to make it mandatory, which is equivalent to add globally to the CGM the unary Boolean constraint:

$$(4.3) \qquad (\texttt{ScheduleMeeting}).$$

Similarly, one can interactively mark/unmark (effortful) tasks as denied (false). More generally, one can mark as satisfied or denied every goal or domain assumption. We call these marks *user assertions*, because they correspond to asserting that an element must be true, i.e., it is part of the solutions we are interested in, or false, i.e., we are interested in solutions that do not include it.

Notice that the process of marking/unmarking elements is conceived to be more *interactive* than that of adding/dropping relation edges or constraints.

4.3 Arithmetical Constraints: Numerical Attributes and SMT($\mathscr{LRA}$) Formulas

In addition to Boolean constraints, it is also possible to use numerical variables to express different numerical attributes of elements (such as cost, worktime, space, fuel, etc.) and to add arithmetical constraints in the form of SMT($\mathscr{LRA}$) formulas over such numerical variables, as showed in Figure 4.5.

4.3.1 Numerical Attributes

For example, suppose we estimate that fulfilling `UsePartnerInstitutions` costs 80€, whereas fulfilling `UseHotelsAndConventionCenters` costs 200€. With CGM-Tool one can express these facts straightforwardly by adding a global numerical variable `cost` to the model;

then, for every element E in the CGM, CGM-Tool automatically generates a numerical variable $\texttt{cost}_{\texttt{E}}$ representing the attribute `cost` of the element E, it adds the following defaultglobal constraint and prerequisite formulas:

$$(\texttt{cost} = \sum_{E} \texttt{cost}_{\texttt{E}}), \tag{4.4}$$

$$\text{for every element } E, \qquad \phi^{+}_{\texttt{E}} \overset{\text{def}}{=} \ldots \wedge (\texttt{cost}_{\texttt{E}} = 0) \tag{4.5}$$

$$\phi^{-}_{\texttt{E}} \overset{\text{def}}{=} \ldots \wedge (\texttt{cost}_{\texttt{E}} = 0), \tag{4.6}$$

that set the default value 0 for each $\texttt{cost}_{\texttt{E}}$. (Notice that (4.4) is a *default* global constraint: the user is free to define his/her own objective functions.) Eventually, for the elements E of interest, one can set a new value for $\texttt{cost}_{\texttt{E}}$ in case E is satisfied: e.g., $\texttt{cost}_{\texttt{UsePartnerInstitutions}} := 80e$ and $\texttt{cost}_{\texttt{UseHotelsAndConventionCenters}} := 200e$. When so, CGM-Tool automatically updates the values in the positive prerequisite formulas (4.5), e.g.:

$$\begin{aligned} \phi^{+}_{\texttt{UsePartnerInstitutions}} &\overset{\text{def}}{=} \ldots \wedge (\texttt{cost}_{\texttt{UsePartnerInstitutions}} = 80) \\ \phi^{+}_{\texttt{UseHotelsAndConventionCenters}} &\overset{\text{def}}{=} \ldots \wedge (\texttt{cost}_{\texttt{UseHotelsAndConventionCenters}} = 200), \end{aligned} \tag{4.7}$$

whereas the corresponding constraint (4.6) is not changed. Similarly, one can set a new value for $\texttt{cost}_{\texttt{E}}$ in case E is denied by updating the values in the negative prerequisite formulas (4.6).

Remark 2. *Notationally, we use variables and formulas indexed by the element they belong to (like, e.g.,* $\texttt{cost}_{\texttt{UsePartnerInstitutions}}$ *and* $\phi^{+}_{\texttt{UsePartnerInstitutions}}$*) rather than* *attribute variables and formulas of the elements in an object-oriented notation (like, e.g.,* UsePartnerInstitutions.cost *and* UsePartnerInstitutions.ϕ^+*) because they are more suitable to be used within the SMT($\mathcal{LRA}$) encodings .*

4.3.2 SMT($\mathcal{LRA}$) Formulas

Suppose that, in order to achieve the nice-to-have requirement LowCost, we need to have a total cost smaller than 100€. This can be expressed by adding to LowCost the prerequisite formula:

$$\phi^+_{\texttt{LowCost}} = \ldots \wedge (\texttt{cost} < 100). \tag{4.8}$$

Hence, e.g., due to (4.4)-(4.8), LowCost and UseHotelsAndConventionCenters cannot be both satisfied, matching the intuition that the latter is too expensive to comply to the nice-to-have LowCost requirement.

Similarly to cost, one can introduce, e.g., another global numerical attribute workTime to reason on working time, and estimate, e.g., that the total working time for ScheduleManually, ScheduleAutomatically, EmailParticipants, CallParticipants, CollectFromSystemCalendar are 3, 1, 1, 2, and 1 hour(s), respectively, and state that the nice-to-have requirement FastSchedule must require a global time smaller than 5 hours. As a result of this process, the system will produce the following constraints.

$$(\texttt{workTime} = \sum_E \texttt{workTime}_{\texttt{E}}) \tag{4.9}$$

$$\phi^+_{\texttt{FastSchedule}} \overset{\text{def}}{=} \ldots \wedge (\texttt{workTime} < 5) \tag{4.10}$$

$$\begin{aligned} \phi^+_{\texttt{ScheduleManually}} &\overset{\text{def}}{=} \ldots \wedge (\texttt{workTime}_{\texttt{ScheduleManually}} = 3) \\ \phi^+_{\texttt{ScheduleAutomatically}} &\overset{\text{def}}{=} \ldots \wedge (\texttt{workTime}_{\texttt{ScheduleAutomatically}} = 1) \\ \phi^+_{\texttt{EmailParticipants}} &\overset{\text{def}}{=} \ldots \wedge (\texttt{workTime}_{\texttt{EmailParticipants}} = 1) \\ \phi^+_{\texttt{CallParticipants}} &\overset{\text{def}}{=} \ldots \wedge (\texttt{workTime}_{\texttt{CallParticipants}} = 2) \\ \phi^+_{\texttt{CollectFromSystemCalendar}} &\overset{\text{def}}{=} \ldots \wedge (\texttt{workTime}_{\texttt{CollectFromSystemCalendar}} = 1), \end{aligned} \tag{4.11}$$

plus the corresponding negative prerequisite formula, which force the corresponding numerical attributes to be zero.

As with the previous case, e.g., the arithmetic constraints make the combination of ScheduleManually and CallParticipants incompatible with the nice-to-have requirement FastSchedule.

Notice that one can build combinations of numerical attributes. E.g., if labor cost is $35e/hour$, then one can redefine cost as ($\texttt{cost} = \sum_E \texttt{cost}_{\texttt{E}} + 35 \cdot \texttt{workTime}$), or introduce a new global variable totalCost as ($\texttt{totalCost} = \texttt{cost} + 35 \cdot \texttt{workTime}$).

Remark 3. *Although the nice-to-have requirements* LowCost *and* FastSchedule *look isolated in Figure 4.5, they are implicitly linked to the rest of the CGM by means of arithmetic constraints on the numerical variables* cost *and* workTime *respectively, which implicitly imply Boolean constraints like:*

$$\begin{array}{lrcl}
(4.12) & \texttt{LowCost} & \rightarrow & \neg\texttt{UseHotelsAndConventionCenters} \\
(4.13) & \texttt{FastSchedule} & \rightarrow & \neg(\texttt{ScheduleManually} \wedge \texttt{CallParticipants}) \\
(4.14) & \texttt{FastSchedule} & \rightarrow & \neg\left(\begin{array}{l} \texttt{ScheduleManually} \wedge \\ \texttt{EmailParticipants} \wedge \\ \texttt{CollectFromSystemCalendar} \end{array}\right) \\
 & \ldots & &
\end{array}$$

Nevertheless, there is no need for stakeholders to consider these implicit constraints, since they are automatically handled by the internal OMT($\mathcal{LRA}$) reasoning capabilities of CGM-Tool.

4.4 Realization of a CGM

We suppose now that ScheduleMeeting is marked satisfied by means of an user assertion (i.e. it is mandatory) and that no other element is marked. Then the CGM in Figure 4.5 has more than 20 possible *realizations*. The sub-graph which is highlighted in yellow in Figure 4.6 describes one of them.

Intuitively, a realization of a CGM under given user assertions represents one of the alternative ways of refining the mandatory requirements (plus possibly some of the nice-to-have ones) in compliance with the user assertions and user-defined constraints. It is a sub-graph of the CGM including a set of satisfied elements and refinements: it includes all mandatory requirements, and [resp. does not include] all elements satisfied [resp. denied] in the user assertions; for each non-leaf element included, at least one of its refinement is included; for each refinement included, all its target elements are included; finally, a realization complies with all relation edges and with all Boolean and SMT($\mathcal{LRA}$) constraints. (Notationally, in all of the Figures a realization is highlighted in yellow, and the denied elements are visible but they are not highlighted.)

Apart from the mandatory requirement, the realization in Figure 4.6 allows to achieve also the nice-to-have requirements `LowCost`, `GoodQualitySchedule`, and `FastSchedule`, and `MinimalEffort`; in order to do this, it requires accomplishing the tasks:

`CharacteriseMeeting`,
`CollectFromSystemCalendar`,
`UsePartnerInstitutions`,
`ScheduleAutomatically`,
`ConfirmOccurrence`,
`GoodParticipation`,
`MinimalConflicts`,
`CollectingEffort`,
`MatchingEffort`,
and it requires the domain assumption
`ParticipantsUseSystemCalendar`.

4.5 Preferences in a CGM

In general, a CGM under given user assertions has many possible realizations. To distinguish among them, stakeholders may want to express *preferences* on the requirements to achieve, on the tasks to accomplish, and on elements and refinements to choose. The CGM-Tool provides various methods to express preferences:

- attribute *penalties and rewards* for tasks and requirements;
- introduce *numerical objectives* to optimize;
- introduce *binary preference relations* between elements and between refinements.

These methods, which are described in what follows, can also be combined.

4.5.1 Preferences via Penalties/Rewards

First, stakeholders can define two numerical attributes called `Penalty` and `Reward`, then stakeholders can assign *penalty* values to tasks and *reward* values to (non-mandatory) requirements (the numbers “`Penalty` = ...” and “`Reward` = ..." in Figure 4.5). This implies that requirements [resp. tasks] with higher rewards [resp. smaller penalties] are preferable. Next, stakeholders can define another numerical attribute `Weight`, that represents the total difference between the penalties and rewards. (This can be defined as a global constraint: (Weight = Penalty − Rewards).) When a model represents preferences, an OMT solver will look for a realization that minimizes its global weight. For instance, one minimum-weight realization of the example CGM, as shown in Figure 4.7, achieves all the nice-to-have requirements except MinimalEffort, with a total weight of −65, which is the minimum which can be achieved with this CGM. Such realization requires accomplishing the tasks:

CharacteriseMeeting,

EmailParticipants,

UsePartnerInstitutions,

ScheduleManually,

ConfirmOccurrence,

GoodParticipation, and

MinimalConflicts,

and requires no domain assumption. (This was found automatically by our CGM-Tool in 0.008 seconds on an Apple MacBook Air laptop.)

4.5.2 Preferences via Multiple Objectives

Stakeholders may define rational-valued *objectives* $obj_1,...,obj_k$ to optimize (i.e., maximize or minimize) as functions of Boolean and numerical variables —e.g., cost, workTime, totalCost can be suitable objectives— and ask the tool to automatically generate realization(s) which optimize one objective, or some combination of more objectives (like totalCost), or which optimizes lexicographically an ordered list of objectives $\langle obj_1, obj_2, ...\rangle$. (We recall that a solution optimizes lexicographically an ordered list of objectives $\langle obj_1, obj_2, ...\rangle$ if it makes obj_1 optimum and, if more than one such solution exists, it makes also obj_2 optimum, ..., etc.) Notice that lexicographic optimization allows for defining objective functions in a very fine-grained way and for preventing ties: if the stakeholder wants to prevent tie solutions on objective obj_1, he/she can define one further preference criterion obj_2 in case of tie on obj_1, and so on.

Importantly, our CGM-Tool provides some pre-defined objectives of frequent usage. Weight (see last paragraph) is one of them. Other examples of pre-defined objectives stakeholders may want to minimize, either singularly or in combination with other objectives, are:

- `numUnsatRequirements`: the number of nice-to-have requirements which are not included in the realization;
- `numSatTasks`: the number of tasks which are included in the realization;
- `numUnsatPrefs`: the number of user-defined binary preference relations which are not fulfilled by the realization (see later).

For example, the previously-mentioned optimum-weight realization of Figure 4.7 is such that `Weight` $= -65$, `workTime` $= 4$ and `cost` $= 80$. Our CGM has many different minimum-weight realizations s.t. `Weight` $= -65$, with different values of `cost` and `workTime`. Among them, it is possible to search, e.g., for the realizations with minimum `workTime`, and among these for those with minimum `cost`, by setting lexicographic minimization with order $\langle$`Weight`, `workTime`, `cost`$\rangle$. This results into one realization with `Weight` $= -65$, `workTime` $= 2$ and `cost` $= 0$ achieving all the nice-to-have requirements, as shown in Figure 4.8, which requires accomplishing the tasks:

`CharacteriseMeeting`,
`CollectFromSystemCalendar`,
`GetRoomSuggestions`,
`CancelLessImportantMeeting`,
`ScheduleAutomatically`,
`ConfirmOccurrence`,
`GoodParticipation`,
`MinimalConflicts`,
`CollectionEffort`,
`MatchingEffort`,
and it requires two domain assumptions:
`ParticipantsUseSystemCalendar`, and
`LocalRoomAvailable`.

(This was found automatically by our CGM-Tool in 0.016 seconds on an Apple MacBook Air laptop.)

4.5.3 Preferences via Binary Preference Relations

In general, stakeholders might not always be at ease in assigning numerical values to state their preferences, or in dealing with SMT($\mathcal{LRA}$) terms, constraints and objectives. Thus, as a more coarse-grained and user-friendly solution, it is also possible for stakeholders to express their preferences in a more direct way by stating explicitly a list of *binary preference relations*, denoted as "$P_1 \succeq P_2$", between pairs of elements of the same kind (e.g. pair of requirements, of tasks, of domain assumptions) or pairs of refinements. "$P_1 \succeq P_2$" means that one prefers to have P_1 satisfied than P_2 satisfied, that is, that he/she would rather avoid having P_1 denied and P_2 satisfied. In the latter case, we say that a preference is unsatisfied. Notice that $P_1 \succeq P_2$ allows for having both P_1 and P_2 satisfied or both denied.

Remark 4. *These are* binary *preferences, so that they say nothing on the fact that each P_i is singularly desirable or not, which in case must be stated separately (e.g., by penalties / rewards.)*

Thus, the fact that a binary preference $P_1 \succeq P_2$ allows for having both P_1 and P_2 denied should not be a surprise: if both $\{P_1 = false, P_2 = true\}$ and $\{P_1 = false, P_2 = false\}$ violated $P_1 \succeq P_2$, then P_2 would play no role in the preference, so that it would reduce to the unary *preference "I'd rather have P_1 than not have it." A dual argument holds for the fact that $P_1 \succeq P_2$ allows for having both P_1 and P_2 satisfied.*

Also, this choice is a very general one, since it implements the case in which $\langle P_1, P_2\rangle$ are both desirable / rewarding ("I prefer winning the Turing Award than winning at the lottery.") like the preference between two requirements, as well as the opposite case in which they are both undesirable / expensive ("I prefer being shot than being hanged.") like the preference between two tasks, plus obviously the trivial case in which P_1 is desirable and P_2 is undesirable. If this choice is considered too general, then the stakeholder can add mutual-exclusion constraints, or combine it lexicographically with penalty / rewards, or directly use penalty / rewards instead.

With CGM-Tool, binary preference relations can be expressed either graphically, via a "prefer" arc "$P_1 \xrightarrow{\text{prefer}} P_2$", or via and ad-hoc menu window. Once a list of binary preference relations is set, the system can be asked to consider the number of unsatisfied preference relations as a pre-defined objective (namely `numUnsatPrefs`), and it searches for a realization which minimizes it. It is also possible to combine such objective lexicographically with the other objectives.

One typical usage we envision for binary preferences is between pairs of refinements of the same element –or equivalently, in case of single-source refinements, between their relative source elements. This allows for expressing stakeholders' preferences between possible ways one intermediate element can be refined.

For example, suppose we want to minimize the total weight of our example goal model. As previously mentioned, there is more than one realization with minimum weight −65. Unlike the previous example, as a secondary choice we disregard workTime and cost; rather, we express also the following binary preferences:

$$\begin{aligned}&\texttt{ConfirmOccurrence} \succeq \texttt{CancelMeeting},\\ &\texttt{UsePartnerInstitutions} \succeq \texttt{UseLocalRoom},\\ &\texttt{UseAvailableRoom} \succeq \texttt{CancelLessImportantMeeting}.\end{aligned} \tag{4.15}$$

(Notice that the goal preferences in (4.15) are pairwise equivalent to the following refinement preferences:

$$R_9 \succeq R_{10},\ R_4 \succeq R_6,\ \text{and}\ R_{17} \succeq R_{18} \tag{4.16}$$

because the refinements in (4.16) are all single-source ones, whose sources are pairwise the goals in (4.15).)

Then we set numUnsatPrefs as secondary objective to minimize after Weight, that is, we set the lexicographic order ⟨Weight,numUnsatPrefs⟩. Then our tool returned the same realization of Figure 4.7 instead of that in Figure 4.8. (This solution was found in 0.018 seconds on an Apple MacBook Air laptop.)

4.6 UNSAT core

It is possible that a CGM is over-constrained, therefore, it is unrealizable, in other words, unsatisfiable. In such case, we can extract the UNSAT-core, i.e., the internal conflicts that cause the CGM to be unrealizable. For example, considered the CGM showed in Figure 4.5, if the domain assumption ParticipantsUseSystemCalendar cannot be achieved, we cannot EmailParticipants for some reason, and we want to have FastSchedule, then clearly this model is over-constrained and unrealizable. The problem is because we will need to add three more constraints to the global constraints:

$$\Phi \overset{\text{def}}{=} \ldots \neg\texttt{ParticipantsUseSystemCalendar} \wedge \neg\texttt{EmailParticipants} \wedge \texttt{FastSchedule}.$$

Such constraints will cause conflicts in the model.

Since the domain assumption ParticipantsUseSystemCalendar is false, the refinement R_{13} will also unachievable, and so does the goal BySystem. Thus, it is impossible to achieve the goal CollectTimetables using the refinement R_3. CollectTimetables can only be obtained by using the refinementR_2 as the result. This means ByPerson must be true. As the refinement R_3 is unachievable, this lead us to unable to use the refinement R_7 to refine the goal ChooseSchedule. This is because the two refinements are bonding. Therefore, the goal ChooseSchedule can only be obtained by using R_8. This means that the task ScheduleManually must be true. Since EmailParticipants is false, ByPerson can only be refined by R_12. This means that the task CallParticipants must be true. However, the requirements FastSchedule cannot be satisfied when both CallParticipants and ScheduleManually are true (as stated in Equation 4.13). Hence, the UNSAT core of the CGM is:

(i) the mandatory requirement ScheduleMeeting,

(i) the constraint (FastSchedule),

(i) the constraint ¬(ParticipantUseSystemCarlendar),

(i) the constraint ¬(EmailParticipants),

(i) the constraint (FastSchedule → ¬(CallParticipants ∧ ScheduleManually)),

(i) the refinement bonding $R_3 \longleftrightarrow R_7$,

(i) the refinement R_1 of ScheduleMeeting,

(i) the refinement R_2 and R_3 of CollectTimetables,

(i) the refinement R_7 and R_8 of ChooseSchedule,

(i) the refinement R_{11} and R_{12} of ByPerson, and

(i) the refinement R_{13} of BySystem.

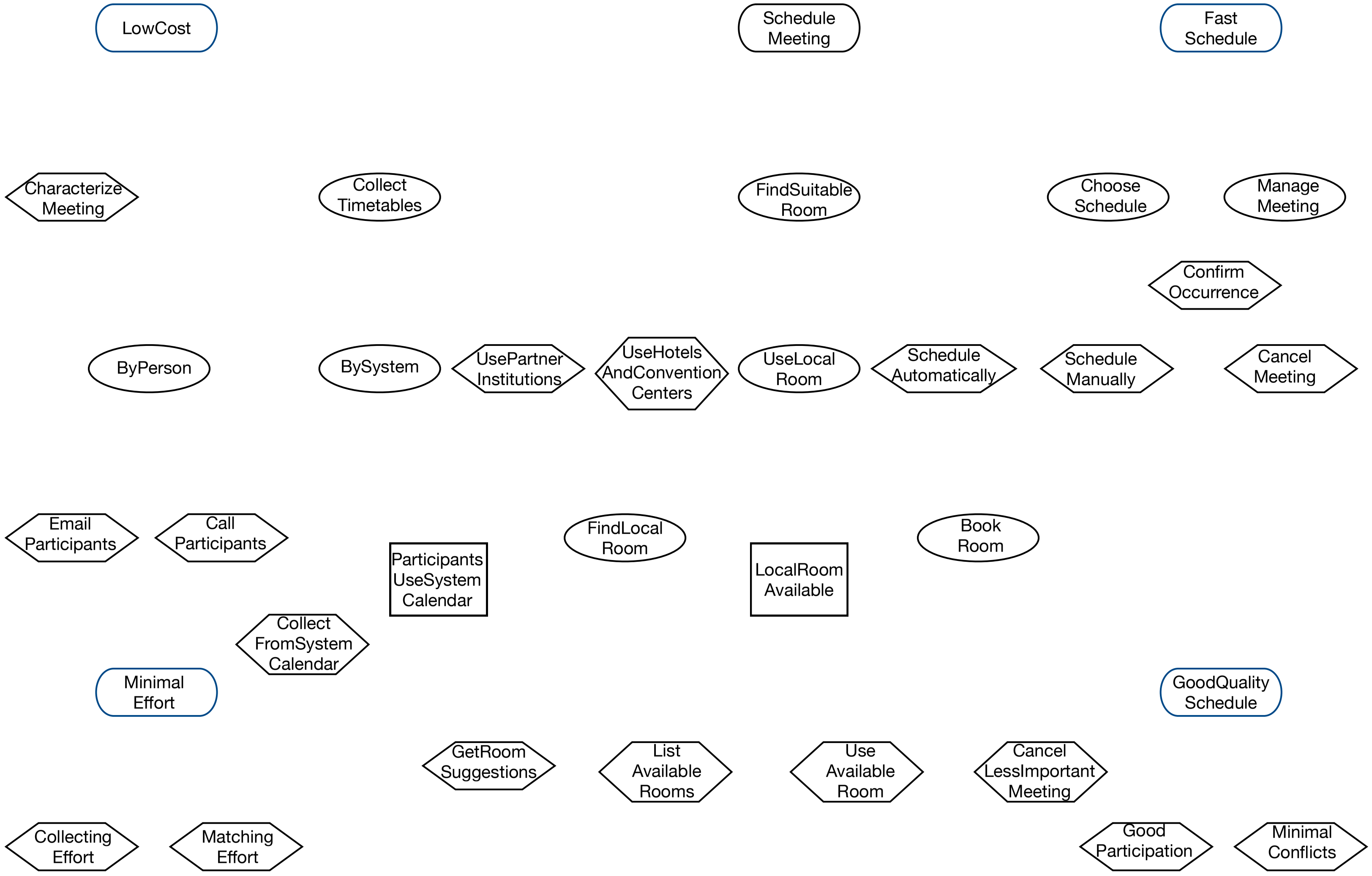


Figure 4.1: Elements of a CGM. Here and elsewhere, round-corner rectangles are requirements; ovals are intermediate goals; hexagons are tasks; rectangles are domain assumptions.

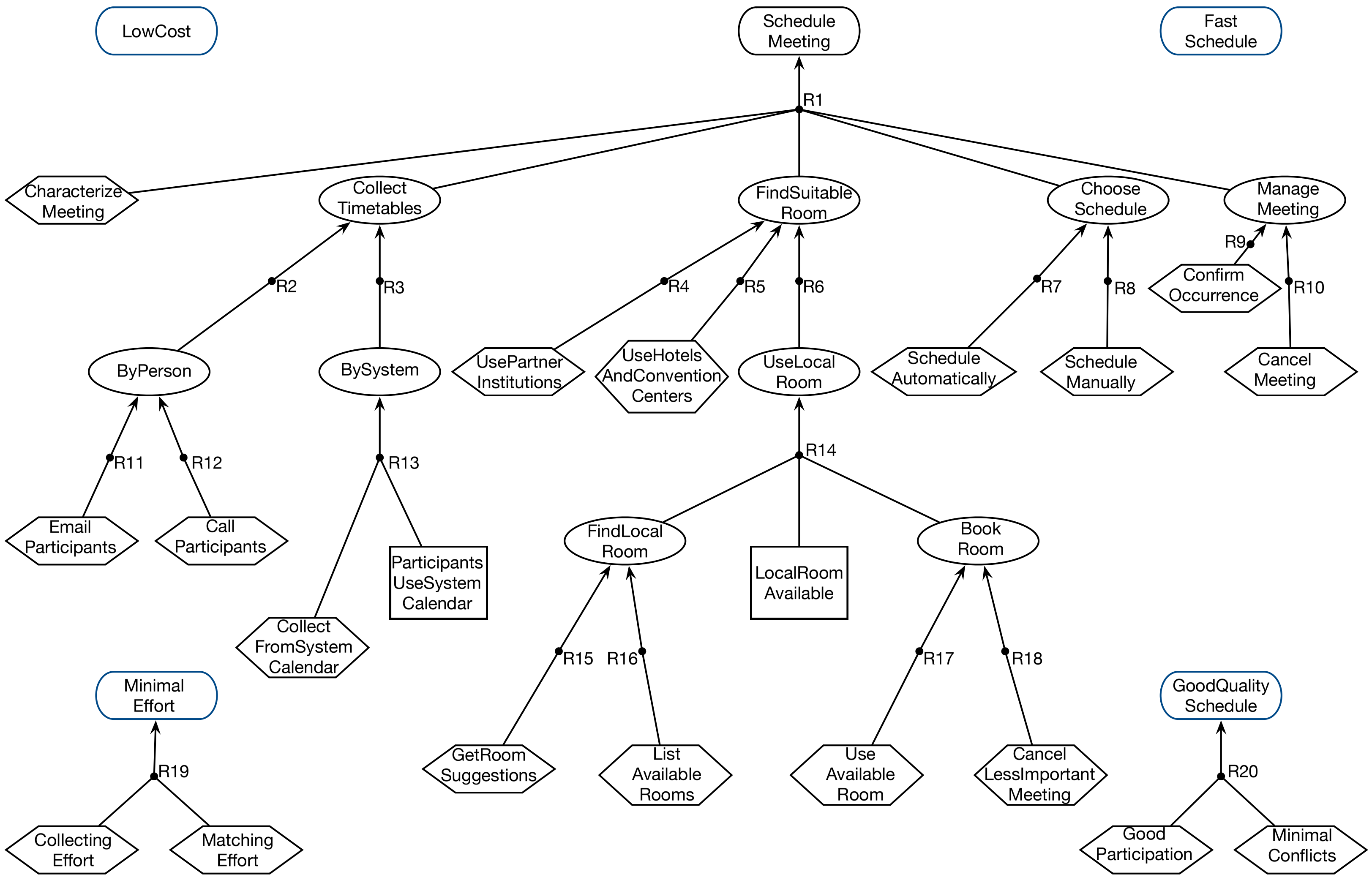


Figure 4.2: Refinements of a CGM. Here and elsewhere, labeled bullets at the merging point of a group of edges are refinements.

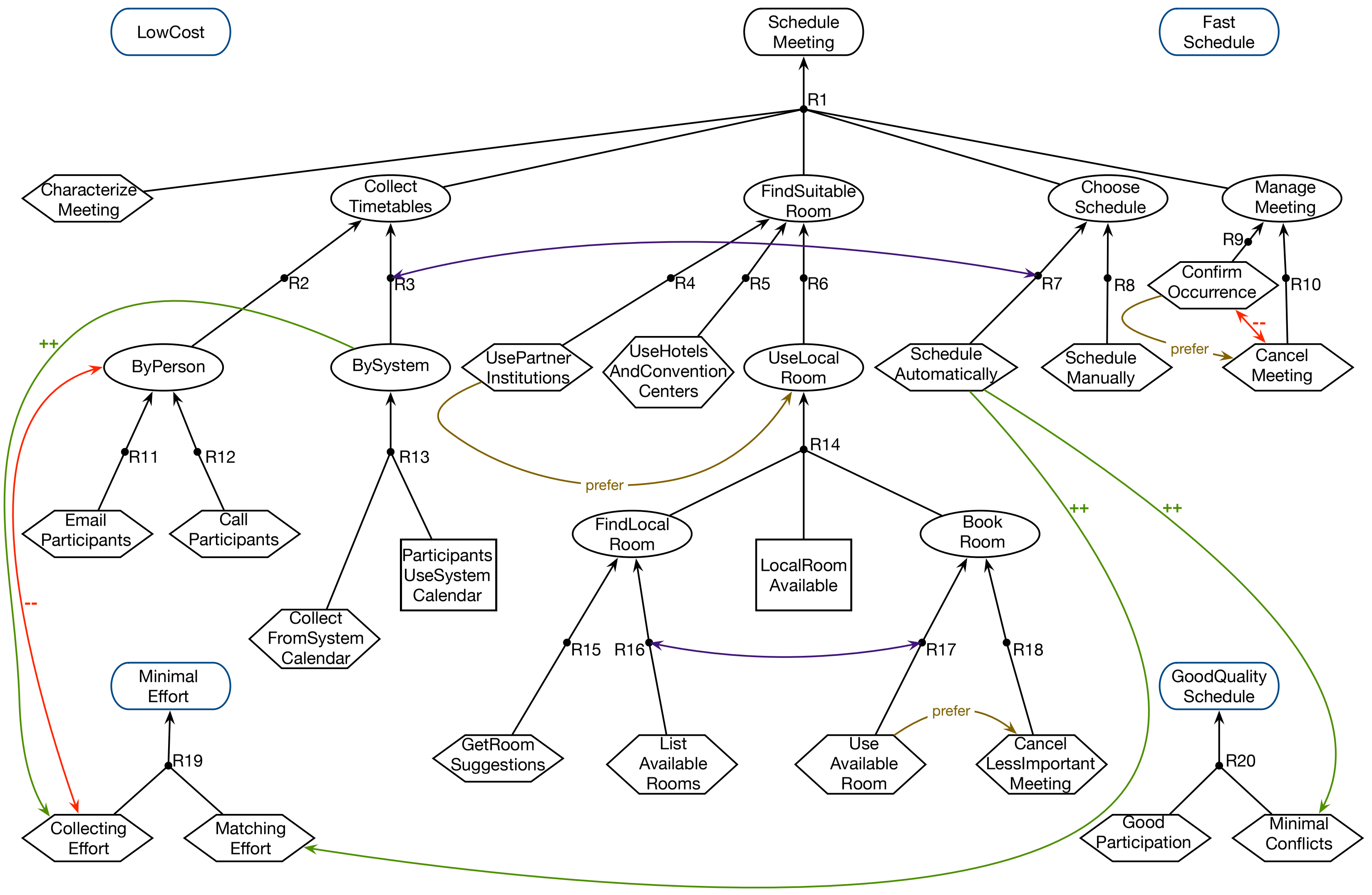


Figure 4.3: Relations in a CGM. Here and elsewhere, contribution edges are labeled with ++ (green); conflict edges are labeled with – (red); preference edges are labeled with *'prefer'* (brown), and refinement bindings are edges between refinements only (purple).

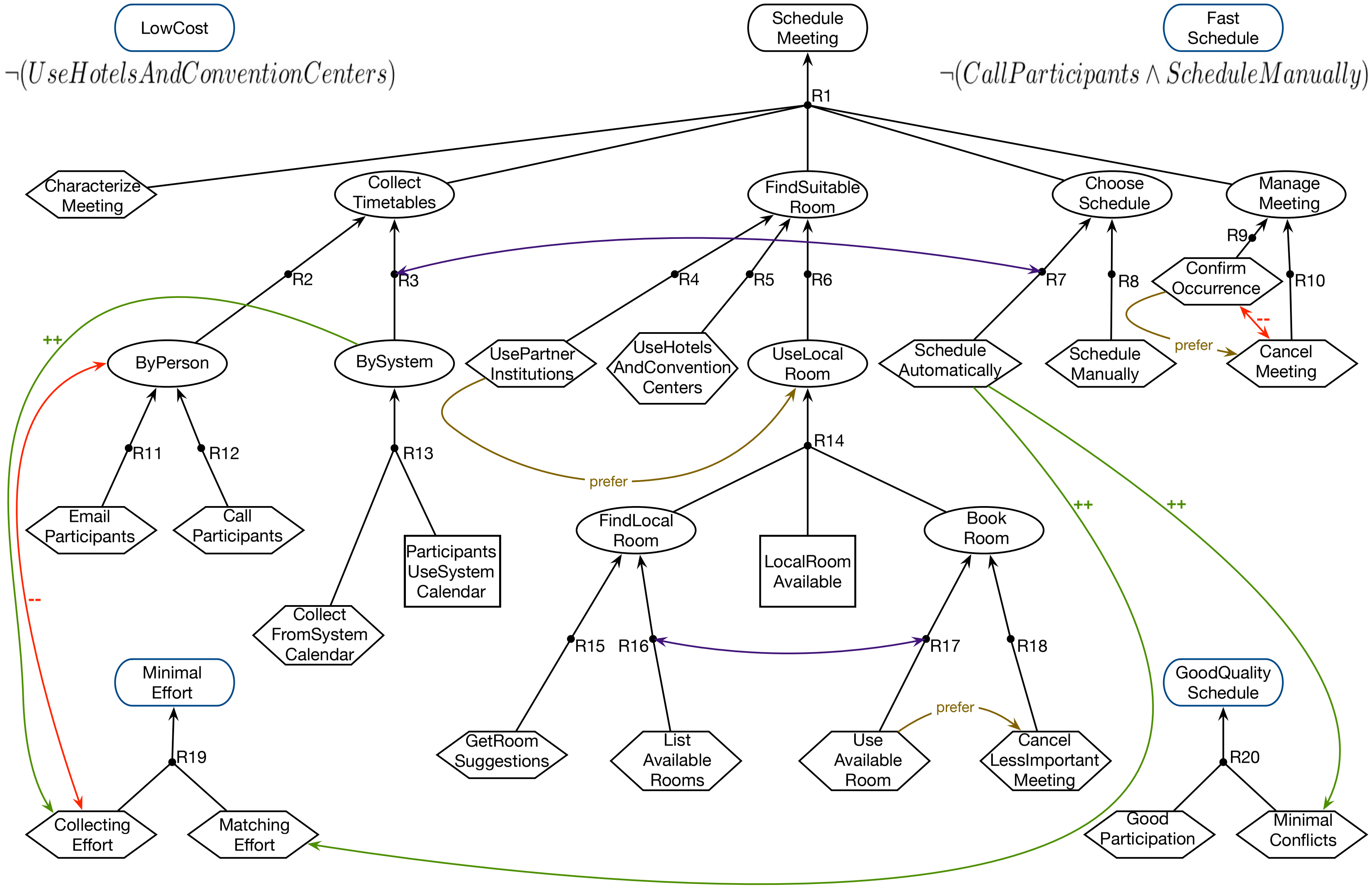


Figure 4.4: An example of a CGM with Boolean formula (written below its associated element.

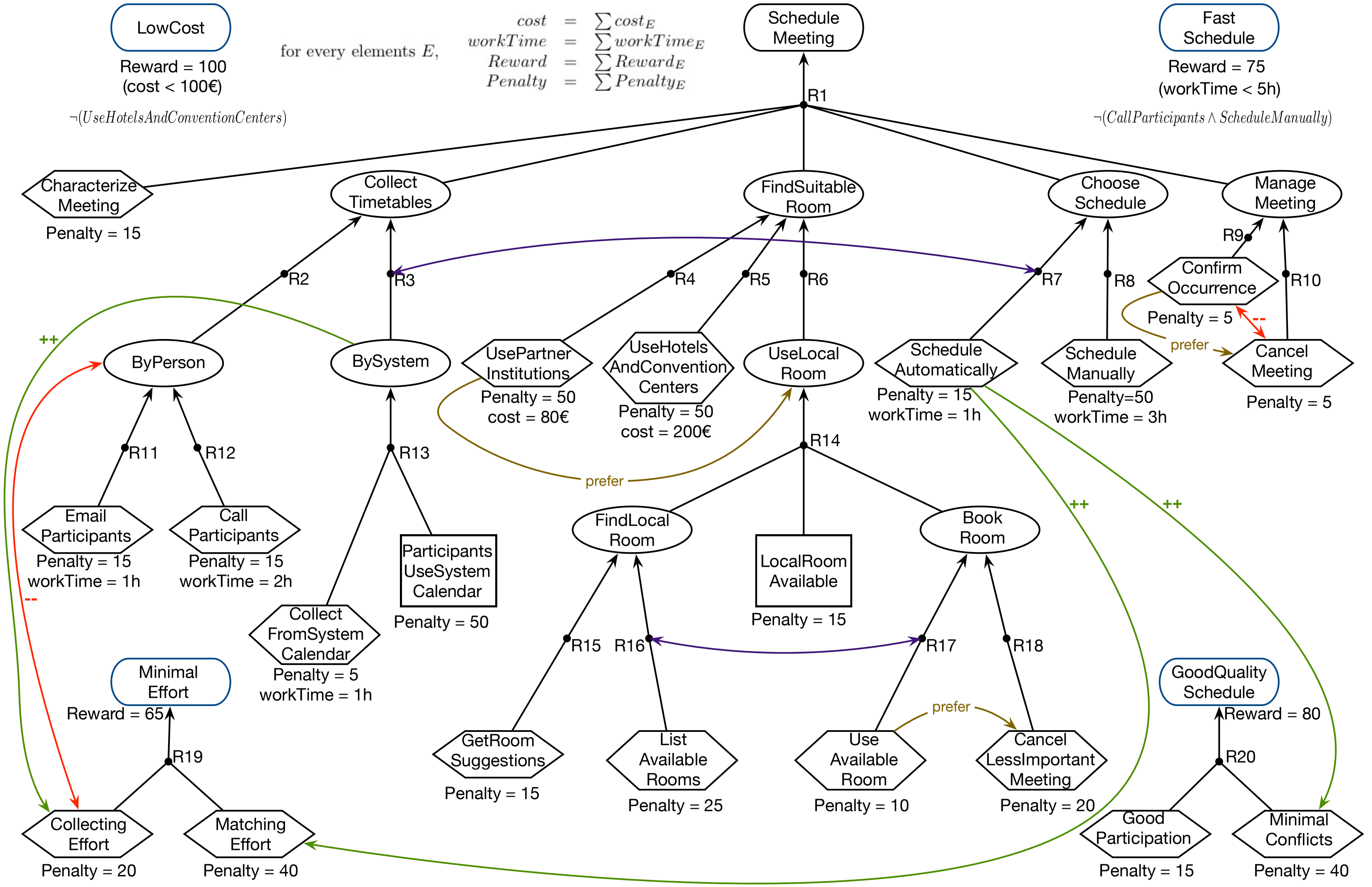


Figure 4.5: An example of a CGM with numerical attributes and arithmetic constraints. Values of numerical attributes associated with the elements and their positive prerequisite formulas are written below the respective elements

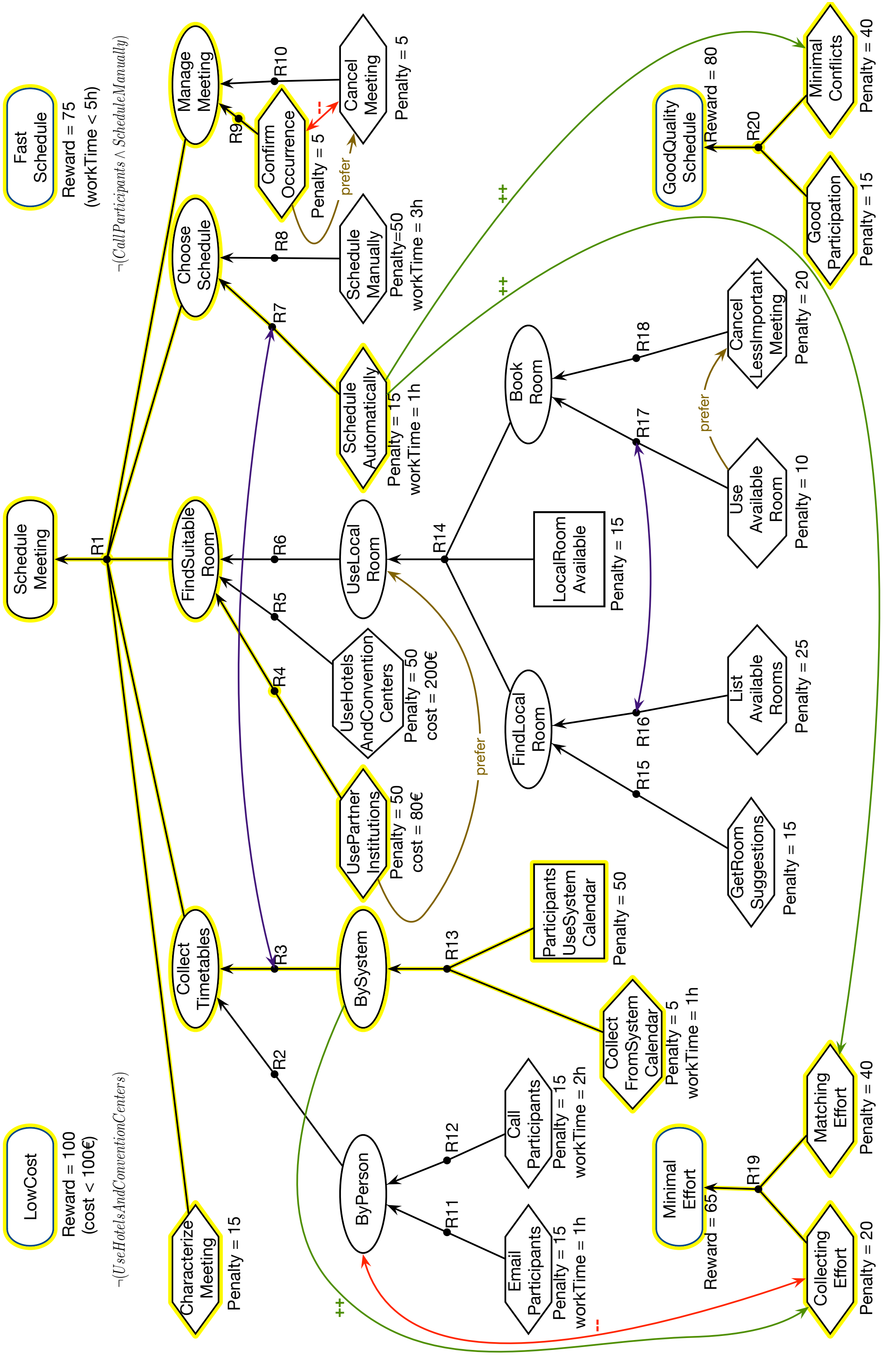


Figure 4.6: An example of a CGM with one of its realizations. Here and elsewhere, the realization is highlighted in yellow, the denied elements are visible but not highlighted.

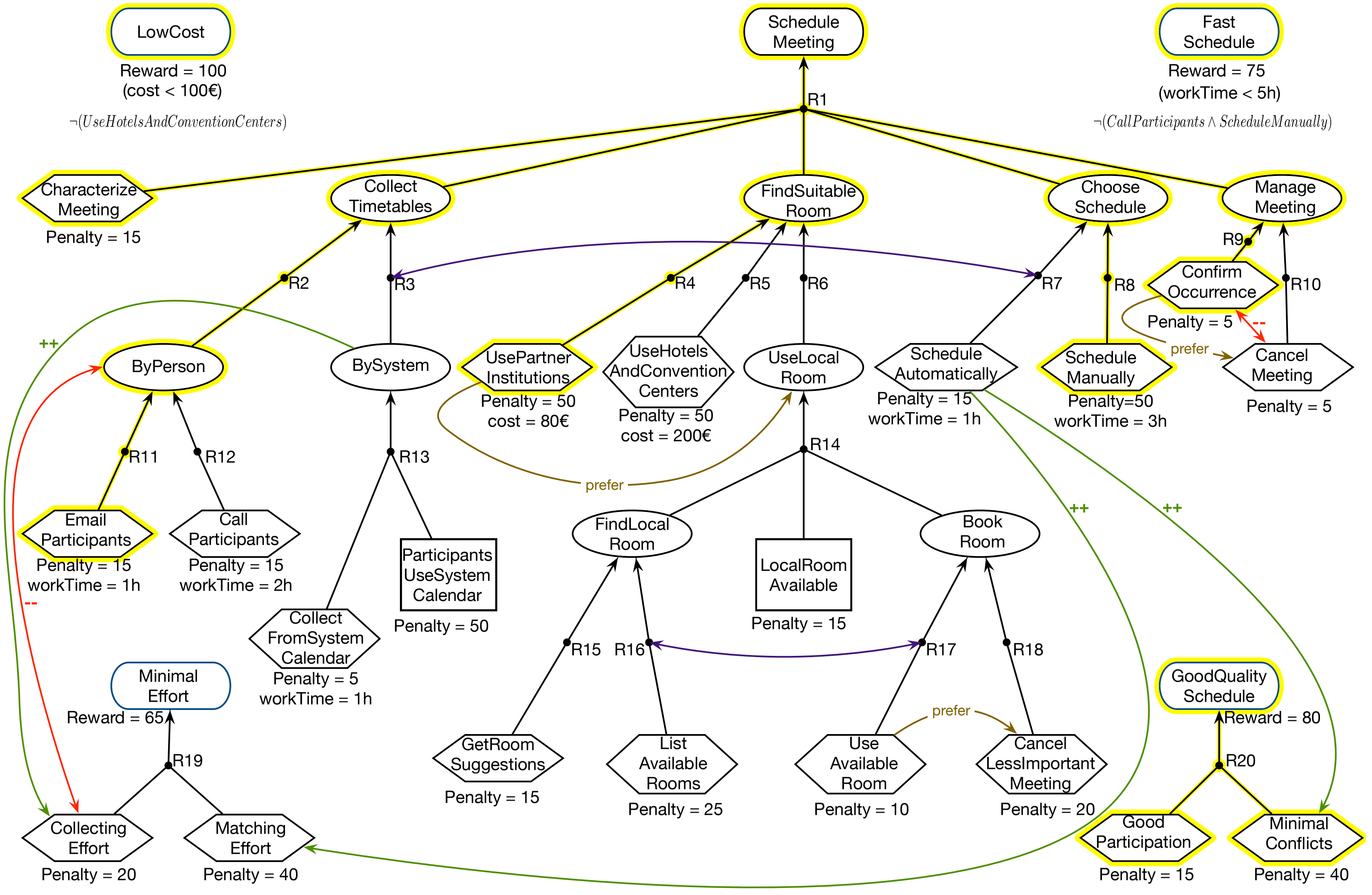


Figure 4.7: An example of a CGM with one of its realization with minimum value of (Penalty − Reward). The value of (Penalty − Reward) is −65

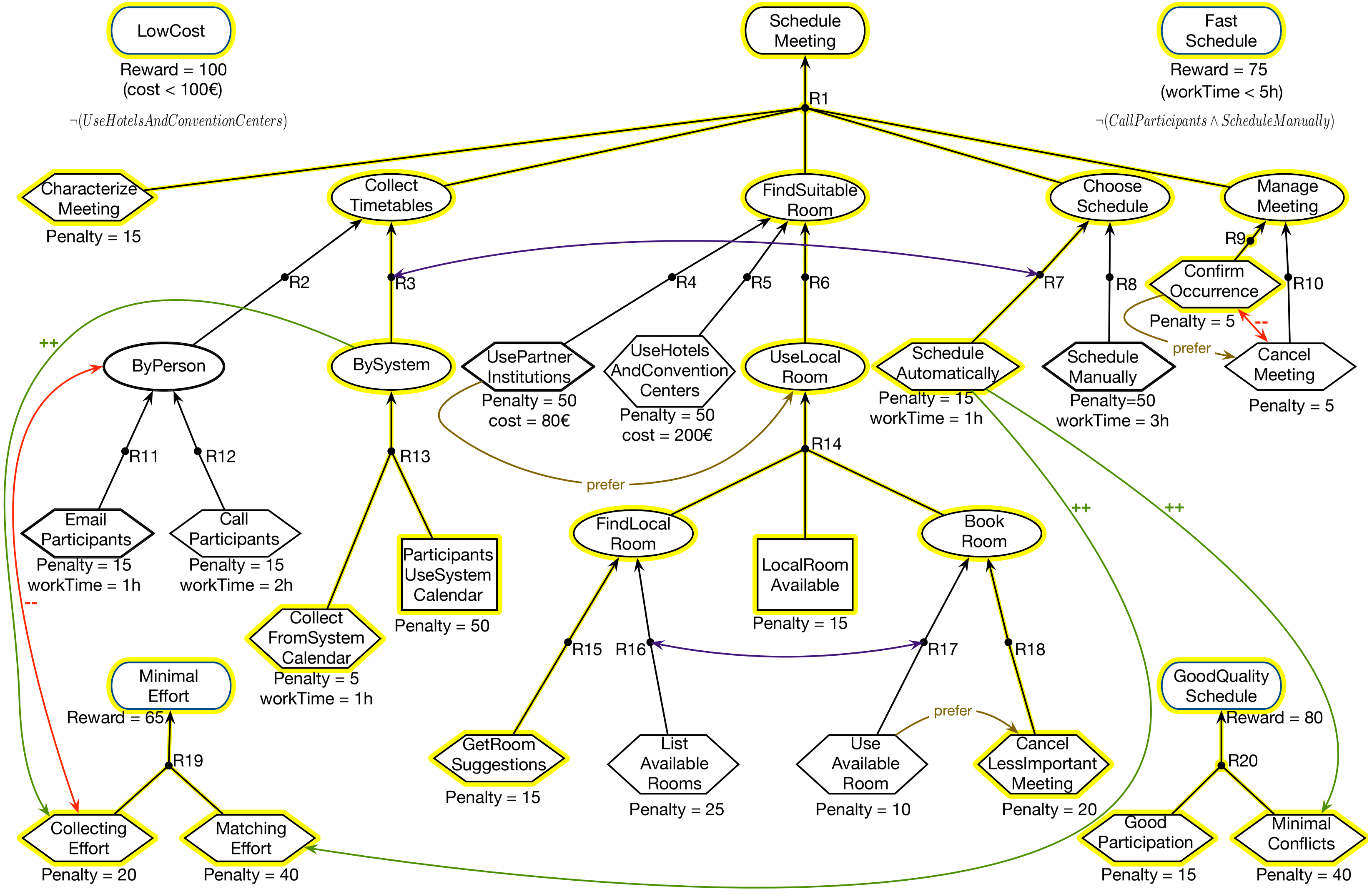


Figure 4.8: An example of a CGM with one of its realization with minimized lexicographically value of [(Penalty − Reward), workTime, cost]. (Penalty − Reward) = −65, workTime = $2h$, and cost = 0€

CHAPTER 5

Abstract Syntax and Semantics of CGM

This chapter presents the abstract syntax and semantics of CGMs, defining formally the building blocks of a CGM and of its realizations, which has already been introduced informally in the previous chapter.

∗ ∗ ★ ∗ ∗

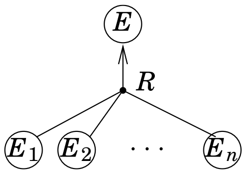
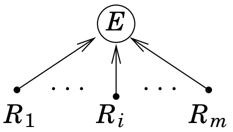
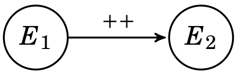
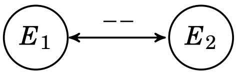
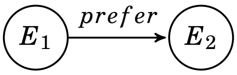
In this chapter we describe formally the abstract syntax and semantics of CGMs.

5.1 Abstract Syntax

We introduce first some general definitions. We call a *goal graph* $\mathscr{D}$ a directed acyclic graph (DAG) alternating element nodes and refinement nodes (collapsed into bullets), s.t.: (a) each element has from zero to many outgoing edges to distinct refinements and from zero to many incoming edges from distinct refinements; (b) each refinement node has exactly one outgoing edge to an element (*target*) and one or more incoming edges from distinct elements (*sources*).

We call a *root element node* any element node that has no outgoing refinement edges, a *leaf element node* any (non-root) element node that has no incoming refinement edges, and an *internal element node* any other element node. (Hereafter we will usually drop the word “node", simply saying “refinement" for “refinement node", “element" for “element node", etc.)

Table 5.1: Summary of Goal Model Structure

Constructor	Textual Representation	Graphical Representation	Propositional Encoding
Goal refinement	$(E_1,\ldots,E_n) \xrightarrow{R} E$		$((\bigwedge_{j=1}^{n} E_j) \leftrightarrow R) \wedge$ $(R \rightarrow E)$
Closed world	—		$E \rightarrow (\bigvee_{R_i \in \mathrm{Ref}(G)} R_i)$
Contribution	$E_1 \xrightarrow{++} E_2$		$(E_1 \rightarrow E_2)$
Conflict	$E_1 \xleftrightarrow{--} E_2$		$\neg(E_1 \wedge E_2)$
Preferences	$E_1 \succeq E_2$		$(E_1 \vee (\neg E_2))$

Notice that, by construction, only elements can be roots and leaves of a goal graph. The sets of root, leaf and internal elements of a goal graph $\mathscr{D}$ are denoted as $\mathsf{R}oots(\mathscr{D})$, $\mathsf{L}eaves(\mathscr{D})$, $\mathsf{I}nternals(\mathscr{D})$ respectively. Given a refinement R with outgoing edge to the element E and incoming edges from the element s $E_1,\ldots,E_n$, we call $E_1,\ldots,E_n$ the *source elements* of R and E the *target element* of R, which are denoted by $\mathsf{S}ources(R)$ and $\mathsf{T}arget(R)$ respectively. We say that R is *a refinement of* E and that R *refines* E *into* $E_1,\ldots,E_n$, denoted "$(E_1,\ldots,E_n) \xrightarrow{R} E$". The set of refinements of an element E are denoted with $\mathsf{R}efinements(E)$.

Elements are *goals* or *domain assumptions*, subject to the following rules:

- a domain assumption cannot be a root element;
- if the target of a refinement R is a domain assumption, then it sources are only domain assumptions;
- if the target of a refinement R is a goal, then at least one of its sources is a goal.

We call root goals and leaf goals *requirements* and *tasks* respectively.

Notationally, we use the symbols R, R_j for labeling refinements, E, E_i for generic elements (without specifying if goals or domain assumptions), G, G_i for goals, A, A_i for domain assumptions. Graphically (see Figure 4.3) we collapse refinements nodes into one bullet, so that we see a refinement as an aggregation of edges from a set of other goals. (See Table 5.1.) Hence, in a goal graph we consider element nodes as the only nodes, and refinements as (aggregations of) edges from a group of source elements to a target element.

Definition 1 (Constrained Goal Model)**.** *A* Constrained Goal Model (CGM) *is a tuple* $\mathscr{M} \overset{def}{=} \langle \mathscr{B}, \mathscr{N}, \mathscr{D}, \Psi \rangle$*, s.t.*

- $\mathscr{B} \overset{def}{=} \mathscr{G} \cup \mathscr{R} \cup \mathscr{A}$ *is a set of atomic propositions, where* $\mathscr{G} \overset{def}{=} \{G_1, ..., G_N\}$*,* $\mathscr{R} \overset{def}{=} \{R_1, ..., R_K\}$*,* $\mathscr{A} \overset{def}{=} \{A_1, ..., A_M\}$ *are respectively sets of goal, refinement and domain-assumption labels. We denote with* $\mathscr{E}$ *the set of element labels:* $\mathscr{E} \overset{def}{=} \mathscr{G} \cup \mathscr{A}$*;*
- $\mathscr{N}$ *is a set of numerical variables in the rationals;*
- $\mathscr{D}$ *is a goal graph, s.t. all its goal nodes are univocally labeled by a goal label in* $\mathscr{G}$*, all its refinements are univocally labelled by a refinement label in* $\mathscr{R}$*, and all its domain assumption are univocally labeled by a assumption label in* $\mathscr{A}$*;*
- Ψ *is a SMT(*$\mathscr{LRA}$*) formula on* $\mathscr{B}$ *and* $\mathscr{N}$*.*

A CGM is thus a "backbone" goal graph $\mathscr{D}$ –i.e., an and-or directed acyclic graph (DAG) of *elements*, as nodes, and *refinements*, as (grouped) edges, which are labeled by atomic propositions in $\mathscr{B}$– which is augmented with an SMT($\mathscr{LRA}$) formula Ψ on the element and refinement labels in $\mathscr{B}$ and on the numerical variables in $\mathscr{N}$. The SMT($\mathscr{LRA}$) formula Ψ is a conjunction of smaller formulas encoding relation edges, global and local Boolean/SMT($\mathscr{LRA}$) constraints, user assertions, and the definition of numerical objectives, all of which will be described later in this section.

Intuitively, a CGM describes a (possibly complex) combination of alternative ways of realizing a set of requirements in terms of a set of tasks, under certain domain assumptions and constraints. A couple of remarks are in order.

Remark 5. *The fact that the goal graph* $\mathscr{D}$ *is an* and-or *graph can be deduced from the propositional encoding of Goal refinement and Closed World in Table 5.1: by combining the propositional encodings of goal refinement and Closed World in Table 5.1, we can infer the formulas:* [1]

[1] We recall that in Boolean logic the formula $\bigwedge_i (R_i \to E)$, which comes from the goal refinement encoding in Table 5.1, is equivalent to $E \leftarrow (\bigvee_i R_i)$. The latter, combined with the encoding of Closed World $E \to (\bigvee_i R_i)$, gives the left formula in (5.1). The right formula in (5.1) is the other part of the goal refinement encoding in Table 5.1.

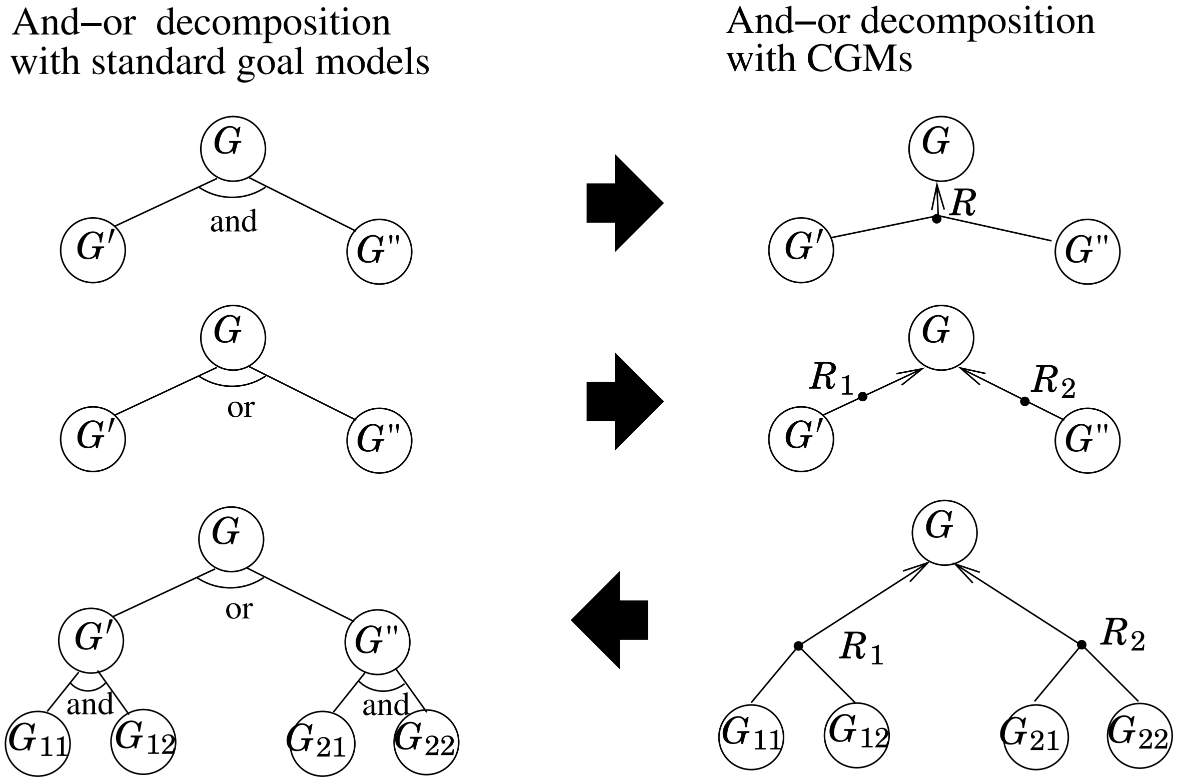

And–or decomposition with standard goal models
And–or decomposition with CGMs
G
G'
G"
and
or
R
R1
R2
G11
G12
G21
G22

Figure 5.1: Top: and-decomposition and its translation into CGM format as a single multi-source refinement. Middle: or-decomposition and its translation into CGM format as multiple single-source refinements. Bottom: a simple piece of CGM (right) and its translation into standard and-or goal model format (left): it is necessary to introduce two auxiliary goals G' and G'' to encode the refinements R_1 and R_2.

$$E \leftrightarrow \bigvee_i R_i \quad \text{and} \quad R \leftrightarrow \bigwedge_j E_j. \tag{5.1}$$

Thus, each non-leaf element E is or-decomposed into the set of its incoming refinements $\{R_i\}_i$*, and each refinement R is and-decomposed into the set of its source elements* $\{E_j\}_j$*.*

Remark 6. *CGMs are more succinct in terms of number of goals than standard and-or goal models. On the one hand, a standard n-ary and-decomposition of a goal can be represented straightforwardly in a CGM by one refinement with n sources (Figure 5.1, Top), and an or-decomposition by n one-source refinements (Figure 5.1, Middle), so that no extra goals are added. On the other hand, in order to represent a piece of CGM with n non-unary refinements by standard goal models, we need introducing n new auxiliary intermediate* *goals to encode refinements, which CGMs encode natively (Figure 5.1, Bottom). We recall from section 4.1 that refinements do not need to be explicitly labeled unless they need to be mentioned in other parts of the model.*

Stakeholders might not be at ease in defining a possibly-complex *global* SMT($\mathcal{LRA}$) formula Ψ to encode constraints among elements and refinements, plus numerical variables. To this extent, as mentioned in section 4.2 and section 4.3, apart from the possibility of defining global formulas, CGMs provide constructs allowing the user to encode *graphically* and *locally* desired constraints of frequent usage: *relation edges*, *prerequisite formulas* $\{\phi_G^+, \phi_G^-\}$ and $\{\phi_R^+, \phi_R^-\}$ and *user assertions*. Each is automatically converted into a simple SMT($\mathcal{LRA}$) formula as follows, and then conjoined to Ψ.

Element-contribution edges, $E_1 \xrightarrow{++} E_2$, meaning that satisfying E_1 forces E_2 to be satisfied (but not vice versa). They are encoded into the formula $(E_1 \rightarrow E_2)$. (The edge $E_1 \xleftrightarrow{++} E_2$ can be used to denote the merging of the two contribution edges $E_1 \xrightarrow{++} E_2$ and $E_2 \xrightarrow{++} E_1$ into one.)

Element-conflict edges, $E_1 \xleftrightarrow{--} E_2$, meaning that E_1 and E_2 cannot be both satisfied. They are encoded into the formula $\neg(E_1 \wedge E_2)$.

Refinement-binding edges, $R_1 \longleftrightarrow R_2$, meaning that, if both the target goals of R_1 and R_2 (namely E_1 and E_2 respectively) are satisfied, then R_1 refines E_1 if and only if R_2 refines E_2. They are encoded into the formula $(E_1 \wedge E_2) \rightarrow (R_1 \leftrightarrow R_2)$.

User assertions, $E_i := \top$ and $E_j := \bot$, are encoded into the formulas (E_i), $(\neg E_j)$ respectively.

Prerequisite formulas, $\{\phi_G^+, \phi_G^-\}$ [resp. $\{\phi_R^+, \phi_R^-\}$] are encoded into the formulas $(G \rightarrow \phi_G^+)$ and $(\neg G \rightarrow \phi_G^-)$ [resp. $(R \rightarrow \phi_R^+)$ and $(\neg R \rightarrow \phi_R^-)$].

The following are instead encoded into SMT($\mathcal{LRA}$) "soft" [2] constraints:

Preference edges, $E_1 \xrightarrow{\text{prefer}} E_2$ [resp. $R_1 \xrightarrow{\text{prefer}} R_2$], and their equivalent *binary preference relations* $E_1 \succeq E_2$ [resp. $R_1 \succeq R_2$], are implemented into the soft constraint $\phi_{E_1 \succeq E_2} \overset{\text{def}}{=} (E_1 \vee (\neg E_2))$ [resp. $\phi_{R_1 \succeq R_2} \overset{\text{def}}{=} (R_1 \vee (\neg R_2))$]. (See also Remark 4 in section 4.5.) Notice that E_1 and E_2 [resp. R_1 and R_2] must be of the same kind,

[2] In constraint programming and other related disciplines (e.g. MaxSAT, MaxSMT, OMT) constraints which must be satisfied are called "hard", whereas constraints which are preferably satisfied but which can be safely violated, although paying some penalty, are called "soft".

i.e. they must be both tasks, or both requirements, or both refinements, or both intermediate goals, or both domain assumptions.

Unlike with other constraints, these soft constraints are *not* added directly to Ψ. Rather, the following SMT($\mathcal{LRA}$) constraint, which defines a numeric Pseudo-Boolean *cost function*, is added to Ψ:

$$(5.2) \qquad (\texttt{numUnsatPrefs} \; = \sum_{\langle E_i E_j\rangle \in \mathcal{P}} \mathsf{i}te(\phi_{E_i \succeq E_j}, 0, 1) + \sum_{\langle R_i R_j\rangle \in \mathcal{P}} \mathsf{i}te(\phi_{R_i \succeq R_j}, 0, 1)),$$

where $\mathcal{P}$ is the list of binary preference relations, and "$\mathsf{i}te(\phi_*, 0, 1)$" denotes an if-then-else arithmetical term, which is evaluated to 0 if ϕ_* is evaluated to true, to 1 otherwise. Hence, `numUnsatPrefs` counts the number of unsatisfied preferences, that is, the number of binary preferences $P_i \succeq P_j$ s.t. P_i is false and P_j is true. [3]

Notice that, unlike refinements, relation edges and preference edges are allowed to create loops, possibly involving refinements. In fact, refinements are acyclic because they represent the and-or decomposition DAG or the CGM requirements. Other arcs (and formulas) represent relations and constraints among elements, and as such they are free to form loops, even with refinements.

Finally we provide the user of a list of syntactic-sugaring constructs, which allow for defining, both globally and locally, the most standard and intuitive constraints among assumption, goal and refinement labels, with no need of defining the corresponding complicate or less-intuitive propositional formulas. (In what follows, $P_1,...,P_n$ denote atomic propositions in $\mathcal{B}$.)

$\mathsf{Alt}(P_1,P_2)$ denotes the fact P_1 and P_2 are alternative, e.g., that one and only one of them is satisfied. This is encoded by the formula $(P_1 \leftrightarrow \neg P_2)$.

$\mathsf{Causes}(P_1,P_2)$ denotes the fact that satisfying P_1 causes P_2 to be satisfied. This is encoded by the formula $(P_1 \rightarrow P_2)$.

$\mathsf{Requires}(P_1,P_2)$ denotes the fact that satisfying P_1 requires P_2 to be satisfied. This is encoded by the formula $(P_1 \rightarrow P_2)$. [4]

[3] In practice, the OMT solver OptiMathSAT [ST15b] provides more efficient ad-hoc encodings for soft constraints like those in (5.2), which we have exploited in the implementation of CGM-Tool; we refer the reader to [ST15b] for details.

[4] Notice that the relation edge $P_1 \xrightarrow{++} P_2$, and the Boolean constraints $\mathsf{Causes}(P_1,P_2)$, $\mathsf{Requires}(P_1,P_2)$, and $(P_1 \rightarrow P_2)$ are equivalent from the perspective of Boolean semantics. Nevertheless, stakeholders may use them in different contexts: e.g., "$\mathsf{Causes}(P_1,P_2)$" is used when event P_1 occurs before P_2 and the former causes the latter, whereas "$\mathsf{Requires}(P_1,P_2)$" is used when P_1 occurs after P_2 and the former requires the latter as a prerequisite.

AtMostOneOf $(\{P_1,...,P_n\})$ denotes the fact that at most one of $\{P_1,...,P_n\}$ must be satisfied. This is encoded by the formula $\big(\bigwedge_{1\le i<j\le n}(\neg P_i \vee \neg P_j)\big)$.

AtLeastOneOf $(\{P_1,...,P_n\})$ denotes the fact that at least one of $\{P_1,...,P_n\}$ must be satisfied. This is encoded by the formula $\big(\bigvee_{1\le i\le n} P_i\big)$.

OneOf $(\{P_1,...,P_n\})$ denotes the fact that exactly one of $\{P_1,...,P_n\}$ must be satisfied. This is encoded by the conjunction of the previous two formulas.

5.2 Semantics

The semantics of CGMs is formally defined in terms of the semantics of simple Boolean expressions, as follows.

Definition 2 (Realization of a CGM)**.** *Let $\mathcal{M} \overset{def}{=} \langle \mathcal{B}, \mathcal{N}, \mathcal{D}, \Psi\rangle$ be a CGM. A* realization *of $\mathcal{M}$ is a $\mathcal{LRA}$-interpretation μ over $\mathcal{B} \cup \mathcal{N}$ such that:*

(a) $\mu \models ((\bigwedge_{i=1}^{n} E_i) \leftrightarrow R) \wedge (R \rightarrow E)$ *for each refinement* $\big(E_1,\dots,E_n\big) \xrightarrow{R} E$*;*

(b) $\mu \models \big(E \rightarrow (\bigvee_{R_i \in \mathsf{Ref}(E)} R_i)\big)$*, for each non-leaf element E;*

(c) $\mu \models \Psi$.

We say that $\mathcal{M}$ is realizable *if it has at least one realization,* unrealizable *otherwise.*

Alternatively and equivalently, (a) and (b) can be substituted by the conditions:

(a') $\mu \models ((\bigwedge_{i=1}^{n} E_i) \leftrightarrow R)$ for each refinement $\big(E_1,\dots,E_n\big) \xrightarrow{R} E$;

(b') $\mu \models \big(E \leftrightarrow (\bigvee_{R_i \in \mathsf{Ref}(E)} R_i)\big)$, for each non-leaf element E,

which reveal the and-or structure of $\mathcal{D}$. (Recall Remark 5 and Footnote 1.)

In a realization μ for a CGM $\mathcal{M} \overset{\text{def}}{=} \langle \mathcal{B}, \mathcal{N}, \mathcal{D}, \Psi\rangle$, each element E or refinement R can be either *satisfied* or *denied* (i.e., their label can be assigned true or false respectively by μ), and each numerical value is assigned a rational value. μ is represented graphically as the sub-graph of $\mathcal{D}$ which includes all the satisfied elements and refinements and does not include the denied elements and refinements. As an example, consider the realization highlighted in yellow in Figure 4.8, where $\texttt{cost} = 0$ and $\texttt{cost}_{\texttt{E}} = 0$ for every element E. From Definition 2, a realization μ represents a sub-graph of the CGM, such that:

(a) A refinement R is part of μ if and only if all its source elements E_i are also included. Moreover, if R is part of μ, then also its target element E is part of it. (See, e.g., refinement R_1 for `ScheduleMeeting`, with all its source goals.)

(b) If a non-leaf goal is in a realization sub-graph, then at least one of its refinements is included in the realization. (See, e.g., refinement R_5 for `FindASuitableRoom`.)

(c) A realization complies with all Boolean and SMT($\mathscr{LRA}$) constraints of the CGM, including relational edges, global and local formulas, user assertions, and the definitions of the numerical attributes and objectives. In particular:

$E_1 \xrightarrow{++} E_2$: If E_1 is in μ, then E_2 is in μ. (See, e.g., the contribution edge `BySystem` $\xrightarrow{++}$ `CollectionEffort`.)

$E_1 \xleftrightarrow{--} E_2$: E_1 and E_1 cannot be both part of μ. (See, e.g., the conflict edge `Byperson` $\xleftrightarrow{--}$ `CollectionEffort`.)

$R_1 \longleftrightarrow R_2$: if both the target goals of R_1 and R_2 are part of the realization μ, then R_1 is in μ if and only if R_2 is there. (See, e.g., the binding $R_{16} \longleftrightarrow R_{17}$.)

User assertions: If E_i is marked satisfied [resp. denied], then E_i is [resp. is not] part of a realization μ. (See, e.g., the requirement `ScheduleMeeting`, which is mandatory, i.e., it is marked satisfied.)

ϕ_G^+: if G is part of a realization μ, then ϕ_G^+ must be satisfied in μ. (E.g., `LowCost` is part of μ, so that $\phi_G^+ \overset{\text{def}}{=} ... \wedge (\texttt{cost} < 100)$ is satisfied, in compliance with the fact that μ sets $\texttt{cost} = 0$.)

ϕ_G^-: if G is *not* part of a realization μ, then ϕ_G^- must be satisfied in μ. (E.g., `UsePartnerInstitutions` is not part of μ, so that $\phi^-_{\texttt{UsePartnerInstitutions}}$ –which includes $(\texttt{cost}_{\texttt{UsePartnerInstitutions}} = 0)$ by (4.6)– is satisfied, in compliance with the fact that μ sets $\texttt{cost}_{\texttt{E}} = 0$ for every E.)

Global formulas and attribute definitions: The realization complies with all global formulas and attribute definitions. (E.g., the global formula $(\texttt{cost} = \sum_E \texttt{cost}_{\texttt{E}})$, which defines the attribute `cost`, is satisfied by μ because $\texttt{cost} = 0$ and $\texttt{cost}_{\texttt{E}} = 0$ for every element E.)

In a realization, each element E or refinement R can be either *satisfied* or *denied* (i.e., their label can be assigned to $\top$ or $\bot$ respectively by μ). If an element E is not a leaf, then it can be satisfied only by satisfying the set of source elements $E_1,\ldots,E_n$ of one of its refinements $(E_1,\ldots,E_n) \xrightarrow{R} E$. If μ satisfies a refinement R of an element E, i.e., it satisfies all the source elements $E_1,\ldots,E_n$, then it satisfies the element E, but not vice versa (condition (a)). For a non-leaf element to be satisfied, at least one of its refinements must be satisfied (condition (b)). We call this fact *Closed World Assumption (CWA)*. The satisfiability or deniability of each element or refinement can be further constrained by all the constraints defined inside the formula Ψ: every realization μ must satisfy such constraints (condition (c)). Notice that, by fulfilling condition (c), a realization must implicitly comply also with all the relation edges, with the user assertions and with the local pre-requisite constraints $\{\phi_E^+, \phi_E^-\}$ and $\{\phi_R^+, \phi_R^-\}$, because the corresponding formulas are conjuncts of Ψ. Thus Ψ contains also the global and local SMT($\mathscr{LRA}$) constraints over global and local numerical attributes (e.g. $\texttt{LowCost} \rightarrow (\texttt{cost} \leq 100)$, $\texttt{UsePartnerInstitutions} \rightarrow (\texttt{cost}_{\texttt{UsePartnerInstitutions}} = 80)$, and the definitions of objectives (e.g., $(\texttt{cost} = \sum_{E \in \mathscr{E}} \texttt{cost}_{\texttt{E}})$.

Remark 7. *Importantly, in the definition of objectives only non-zero terms of the sums need to be considered. (E.g., the sum in* $(\texttt{cost} = \sum_{E \in \mathscr{E}} \texttt{cost}_{\texttt{E}})$ *can be safely restricted to the elements* UsePartnerInstitutions *and* UseHotelsAndConventionCenters.*) This allows for reducing drastically the number of rational variables involved in the encoding. In the implementation of CGM-Tool we have exploited this fact.*

CHAPTER 6

Automated Reasoning with Constrained Goal Models

This chapter describes the automated reasoning functionalities on CGMs, which we support by encoding them into SMT and OMT. We first show how to encode a CGM $\mathcal{M}$ into a SMT($\mathcal{LRA}$) formula $\Psi_{\mathcal{M}}$, so that the search for an optimum realization of $\mathcal{M}$ reduces to an OMT($\mathcal{LRA}$) problem over the formula $\Psi_{\mathcal{M}}$, which is then fed to an OMT solver. Then we present the reasoning functionalities over CGMs we have implemented on top of our OMT solver.

* * ★ * *

This chapter presents the encoding of CGMs and automated reasoning functionalities on CGM.

6.1 Encoding of CGMs

Definition 3 (SMT($\mathscr{LRA}$) Encoding of a CGM)**.** *Let $\mathscr{M} \overset{def}{=} \langle \mathscr{B}, \mathscr{N}, \mathscr{D}, \Psi \rangle$ be a CGM. The* SMT($\mathscr{LRA}$) encoding *of $\mathscr{M}$ is the formula $\Psi_{\mathscr{M}} \overset{def}{=} \Psi \wedge \Psi_{\mathscr{R}} \wedge \Psi_{\mathscr{E}}$, where:*

$$\Psi_{\mathscr{R}} \overset{def}{=} \bigwedge_{\left(E_1,\dots,E_n\right) \xrightarrow{R} E,\ R \in \mathscr{R}} \left(\left(\bigwedge_{i=1}^{n} E_i \leftrightarrow R \right) \wedge (R \to E) \right), \tag{6.1}$$

$$\Psi_{\mathscr{E}} \overset{def}{=} \bigwedge_{E \in \mathsf{R}oots(\mathscr{D}) \cup \mathsf{I}nternals(\mathscr{D})} \left(E \to \left(\bigvee_{R_i \in \mathsf{Refinements}(E)} R_i \right) \right). \tag{6.2}$$

$\mathsf{R}oots(\mathscr{D})$ and $\mathsf{I}nternals(\mathscr{D})$ being the root and internal elements of $\mathscr{D}$ respectively. We call $\Psi_{\mathscr{M}}$ the SMT($\mathscr{LRA}$) Encoding *of the CGM $\mathscr{M}$.*

Notice that the formulas $\Psi_{\mathscr{R}}$ and $\Psi_{\mathscr{E}}$ in (6.1) and (6.2) encode directly points (a) and (b) in Definition 2, for every element and refinement in the CGM. In short, the $\Psi_{\mathscr{R}} \wedge \Psi_{\mathscr{E}}$ component of $\Psi_{\mathscr{M}}$ encodes the relation induced by the and-or goal graph $\mathscr{D}$ in $\mathscr{M}$. The component Ψ is the formula described in point (c) in Definition 2, which encodes all Boolean and SMT($\mathscr{LRA}$) constraints of the CGM, including relational edges, global and local formulas, user assertions, and the definitions of the numerical attributes and objectives.

Therefore, the following facts are straightforward consequences of Definitions 2 and 3 and of the definition and OMT($\mathscr{LRA}$).

Proposition 1. *Let $\mathscr{M} \overset{def}{=} \langle \mathscr{B}, \mathscr{N}, \mathscr{D}, \Psi \rangle$ be a CGM; let $\Psi_{\mathscr{M}}$ its SMT($\mathscr{LRA}$) encoding as in Definition 3; let μ a $\mathscr{LRA}$-interpretation over $\mathscr{B} \cup \mathscr{N}$. Then μ is a realization of $\mathscr{M}$ if and only if $\mu \models \Psi_{\mathscr{M}}$.*

In short, Proposition 1 says that μ is a realization for the CGM $\mathscr{M}$ if and only if μ is a model in SMT($\mathscr{LRA}$) for the formula $\Psi_{\mathscr{M}}$. Therefore, a realization μ for $\mathscr{M}$ can be found by invoking a SMT($\mathscr{LRA}$) solver on the CGM encoding $\Psi_{\mathscr{M}}$.

Proposition 2. *Let $\mathscr{M}$ and $\Psi_{\mathscr{M}}$ be as in Proposition 1, and let μ be a realization of $\mathscr{M}$. Let $\{obj_1, \dots, obj_k\}$ be numerical objectives occurring in $\Psi_{\mathscr{M}}$. Then we have that:*

(i) for every i in $1, \dots, k$, μ minimizes [resp. maximizes] obj_i if and only if μ is a solution of the OMT($\mathscr{LRA}$) minimization [resp. maximization] problem $\langle \Psi_{\mathscr{M}}, \langle obj_i \rangle \rangle$;

(ii) μ lexicographically minimizes [resp. maximizes] $\langle obj_1, \dots, obj_k \rangle$ if and only if μ is a solution of the OMT($\mathscr{LRA}$) lexicographic minimization [resp. maximization] problem $\langle \Psi_{\mathscr{M}}, \langle obj_1, \dots, obj_k \rangle \rangle$.

In short, Proposition 2 says that μ is a realization for the CGM $\mathcal{M}$ which optimizes lexicographically $\langle obj_1,...,obj_k\rangle$ if and only if μ is a model in SMT($\mathcal{LRA}$) for the formula $\Psi_{\mathcal{M}}$ which optimizes lexicographically $\langle obj_1,...,obj_k\rangle$. Therefore, one such realization can be found by invoking a OMT($\mathcal{LRA}$) solver on $\Psi_{\mathcal{M}}$ and $\langle obj_1,...,obj_k\rangle$. Notice that we are always looking for *one* realization at a time. Multiple realizations require multiple calls to the OMT solver.

6.2 Automated Reasoning on CGMs

Propositions 1 and 2 suggest that realizations of a CGM $\mathcal{M}$ can be produced by applying SMT($\mathcal{LRA}$) solving to the encoding $\Psi_{\mathcal{M}}$, and that *optimal* realizations can be produced by applying OMT($\mathcal{LRA}$) to $\Psi_{\mathcal{M}}$ and a list of defined objectives $obj_1,...,obj_k$. (Notice that such list may include also the pre-defined objectives `numUnsatRequirements`, `numSatTasks` and `numUnsatPrefs` of chapter 4 and (5.2) to be minimized.) This allowed us to implement straightforwardly the following reasoning functionalities on CGMs by interfacing with a SMT/OMT tool.

Search/enumerate realizations. Stakeholders can automatically check the realizability of a CGM $\mathcal{M}$ –or to enumerate one or more of its possible realizations– under a group of user assertions and of user-defined Boolean and SMT($\mathcal{LRA}$) constraints; the tool performs this task by invoking the SMT solver on the formula $\Psi_{\mathcal{M}}$ of Definition 3. The realization of the goal model presented in Figure 4.6 is found using this functionality of the tool.

Search/enumerate minimum-penalty/maximum reward realizations. Stakeholders can assert the desired requirements and set penalties of tasks; then the tool finds automatically realizations achieving the former while minimizing the latter, by invoking the OMT solver on $\Psi_{\mathcal{M}}$ with the pre-defined `Weight` objective. The vice versa is obtained by negating undesired tasks and setting the rewards of nice-to-have requirements. Every intermediate situations can be also be obtained. The realization of the goal model presented in Figure 4.7 is found using this functionality of the tool.

Search/enumerate optimal realizations wrt. pre-defined/user-defined objectives. Stakeholders can define their own objective functions $obj_1,...,obj_k$ over goals, refinements and their numerical attributes; then the tool finds automatically realizations optimizing them, either independently or lexicographically, by invoking the

OMT solver on $\Psi_{\mathcal{M}}$ and $obj_1,...,obj_k$. User-defined objectives can also be combined with the pre-defined ones, like `Weight`, `numUnsatRequirements`, `numSatTasks` and `numUnsatPrefs`. The realization of the goal model presented in Figure 4.8 is found using this functionality of the tool.

In particular, notice that `numUnsatPrefs`allows for addressing the fulfillment of the maximum number of binary preferences as the optimization of a pre-defined objective.

Example 1. *As a potentially frequent scenario, stakeholders may want to find a realization which minimizes, in order of preference, the number of unsatisfied non-mandatory requirements, the number of unsatisfied binary preferences, and the number of satisfied tasks. This can be achieved by setting the following ordered list of pre-defined objectives to minimize lexicographically:*

$$\langle \texttt{numUnsatRequirements}, \texttt{numUnsatPrefs}, \texttt{numSatTasks} \rangle.$$

Notice that all the above actions can be performed *interactively* by marking an unmarking (nice-to-have) requirements, tasks and domain assumptions, each time searching for a suitable or optimal realization.

Importantly, when a CGM is found un-realizable under a group of user assertions and of user-defined Boolean and SMT($\mathcal{LRA}$) constraints, it highlights the subparts of the CGM and the subset of assertions causing the problem. This is implemented by asking the SMT/OMT solver to identify the *unsatisfiable core* of the input formula —i.e. the subset of sub-formulas which caused the inconsistency, see e.g. [CGS11]— and mapping them back into the corresponding information.

CHAPTER 7

Requirements Evolution and Evolution Requirements with Constrained Goal Models

We are living in an ever-changing world. Changes need to be accommodated by any system that lives and operates in the world. In this chapter, we propose to study how to model and reason with requirements evolution *as well as model and reason with* evolution requirements.

* * ★ * *

This chapter includes a proposal for modelling changing requirements in terms of changes to a CGM model, the identification of a new class of evolution requirements, expressed as optimization goals in CGM. In addition, we show how to support reasoning with changed goal models and evolution requirements in order to derive optimal solutions.

7.1 Motivation

We have come to live in a world where the only constant is change. Changes need to be accommodated by any system that lives and operates in that world, biological and/or engineered. For software systems, this is a well-known problem referred to as software evolution. There has been much work and interest on this problem since Lehman's seminal proposal for laws of software evolution [Leh80]. However, the problem of effectively supporting software evolution through suitable concepts, tools and techniques is still largely open. And software evolution still accounts for more than 50% of total costs in a software system's lifecycle.

We are interested in supporting software evolution caused by changing requirements and/or environmental conditions. Specifically, we are interested in models that capture such changes, also in reasoning techniques that derive optimal new specifications for a system whose requirements and/or environment have changed. Moreover, we are interested in discovering new classes of evolution requirements, in the spirit of [Sou12] who proposed such a class for adaptive software systems. We propose to model requirements changes through changes to a goal model, and evolution requirements as optimization goals, such as "Minimize costs while implementing new functionality".

7.2 Requirements Evolution

Constrained goal models may evolve in time: goals, requirements and assumptions can be added, removed, or simply modified; Boolean and SMT constraints may be added, removed, or modified as well; assumptions which were assumed true can be assumed false, or vice versa.

Some modifications *strengthen* the CGMs, in the sense that they reduce the set of candidate realizations. For instance, dropping one of the refinements of an element (if at least one is left) reduces the alternatives in realizations; adding source elements to a refinement makes it harder to satisfy; adding Boolean or SMT constraints, or making some such constraint strictly stronger, restricts the set of candidate solutions; changing the value of an assumption from true to false may drop some alternative solutions. Vice versa, some modifications *weaken* the CGMs, augmenting the set of candidate realizations: for instance, adding one of refinement to an element, dropping source elements to a refinement, dropping Boolean or SMT constraints, or making some such constraint strictly weaker, changing the value of an assumption from false to true. In general, however, since in a CGM the goal and/or decomposition graph is a DAG and not a tree, and the and/or decomposition is augmented with relational edges and constraints, modifications may produce combinations of the above effects, possibly propagating unexpected side effects which are sometimes hard to predict.

We consider the CGM of a Schedule Meeting described in Figure 7.1 (namely, $\mathcal{M}_1$) as our starting model, and we assume that for some reasons it has been modified into the CGM $\mathcal{M}_2$ in Figure 7.2. $\mathcal{M}_2$ differs from $\mathcal{M}_1$ for the following modifications:

(a) two new tasks, `SetSystemCalendar` and `ParticipantsFillSystemCalendar`, are added to the sub-goal sources of the refinement R_{13};

(b) a new source task `RegisterMeeting` is added to R_{17}, and the binding between R_{16} and R_{17} is removed; the refinement R_{18} of the goal `BookRoom` and its source task `CancelLessImportantMeeting` are removed;

(c) the alternative refinements R_9 and $R_1 0$ of `ManageMeeting` are also modified: two new internal goals `ByUser` and `ByAgent` are added and become the single source of the two refinements R_9 and $R_1 0$ respectively, and the two tasks `ConfirmOccurrence` and `CancelMeeting` become respectively the sources of two new refinements R_{21} and R_{22}, which are the alternative refinements of the goal `ByUser`; the new goal `ByAgent` is refined by the new refinement R_{23} with source task `SendDecision`.

7.3 Evolution Requirements

We consider the generic scenario in which a previous version of a CGM $\mathscr{M}_1$ with an available realization μ_1 is modified into a new CGM $\mathscr{M}_2$. We call the *restriction of* μ_1 *to* $\mathscr{M}_2$, denoted with μ_1, the restriction of μ_1 to the atomic propositions and rational values which are common to $\mathscr{M}_1$ and $\mathscr{M}_2$. (That is, μ_1 is the part of μ_1 which is still of interest for $\mathscr{M}_2$.) In Figure 7.2, we highlight the restriction μ_1 of μ_1 to the novel CGM $\mathscr{M}_2$.

As a consequence of modifying a CGM $\mathscr{M}_1$ into a new version $\mathscr{M}_2$, μ_1 typically is no more a valid realization of $\mathscr{M}_2$.[1] E.g., we notice that μ_1 in Figure 7.2 does not represent a valid realization of $\mathscr{M}_2$: not all source tasks of R_{13} are satisfied, `BookRoom` has no satisfied refinement, and the new goal `ByUser` and refinement R_{21} are not satisfied. It is thus necessary to produce a new realization μ_2 for $\mathscr{M}_2$.

In general, when one has a sequence $\mathscr{M}_1, \mathscr{M}_2, ..., \mathscr{M}_i, ...$ of CGMs and must produce a corresponding sequence $\mu_1, \mu_2, ..., \mu_i, ...$ of realizations, it is necessary to decide some criteria by which the realizations μ_i evolve in terms of the evolution of the CGMs $\mathscr{M}_i$. We call these criteria, *evolution requirements*. We describe some possible criteria.

7.3.1 Recomputing realizations

One possible evolution requirement is that of always having the "best" realization μ_i for each $\mathscr{M}_i$, according to some objective (or lexicographic combination of objectives). Let $\mathscr{M}_1$,

[1] More precisely, rather than "μ_1", here we should say "the restriction of μ_1 to the elements and variables which are still in $\mathscr{M}_2$." We will keep this distinction implicit in the rest of the paper.

$\mathscr{M}_2$, and μ_1 be as above. One possible choice for the user is to compute a new optimal realization μ_2 from scratch, using the same criteria used in computing μ_1 from $\mathscr{M}_1$. In general, however, it may be the case that the new realization μ_2 is very different from μ_1, which may displease the stakeholders.

We consider now the realization μ_1 of the CGM $\mathscr{M}_1$ highlighted in Figure 7.1 and the modified model $\mathscr{M}_2$ of Figure 7.2. If we run CGM-Tool over $\mathscr{M}_2$ with the same optimization criteria as for μ_1 –i.e., minimize lexicographically, in order, the difference `Penalty-Reward`, `workTime`, and `cost`– we obtain a novel realization μ_2^{lex} depicted in Figure 7.3. The new realization μ_2^{lex} satisfies all the requirements (both "nice to have" and mandatory). It includes the following tasks: `ChirateriseMeeting`, `EmailParticipants`, `GetRoomSuggestions`, `UseAvailableRoom`, `RegisterMeetingRoom`, `ScheduleManually`, `ConfirmOccurrence`, `GoodParticipation`, and `MinimalConflicts`, and it requires one domain assumption: `LocalRoomAvailable`. This realization was found automatically by our CGM-Tool in 0.059 seconds on an Apple MacBook Air laptop.

Unfortunately, μ_2^{lex} turns out to be extremely different from μ_1. This is due to the fact that the novel tasks `SetSystemCalendar` and `ParticipantsFillSystemCalendar` raise significantly the penalty for R_{13} and thus for R_2; hence, in terms of the `Penalty-Reward` objective, it is now better to choose R_{10} and R_6 instead of R_2 and R_7, even though this forces `ByPerson` to be satisfied, which is incompatible with `CollectionEffort`, so that `MinimalEffort` is no more achieved. Overall, for μ_2 we have $\texttt{Penalty} - \texttt{Reward} = -65$, $\texttt{workTime} = 4h$ and $\texttt{cost} = 0e$.

In many contexts, in particular if μ_1 is well-established or is already implemented, one may want to find a realization μ_2 of the modified CGM $\mathscr{M}_2$ which is as similar as possible to the previous realization $\mathscr{M}_1$. The suitable notion of "similarity", however, may depend on stakeholder's needs. In what follows, we discuss two notions of "similarity" from [EBMJ12], *familiarity* and *change effort*, adapting and extending them to CGMs.

7.3.2 Maximizing familiarity

In our approach, in its simplest form, the *familiarity* of μ_2 wrt. μ_1 is given by the number of elements of interest which are common to $\mathscr{M}_1$ and $\mathscr{M}_2$ and which either are in both μ_1 and μ_2 or are out of both of them; this can be augmented also by the number of new elements in $\mathscr{M}_2$ of interest (e.g., tasks) which are denied. In a more sophisticate form, the contribution of each element of interest can be weighted by some numerical value (e.g., `Penalty`, `cost`, `WorkTime`,...). This is formalized in section 7.4, and a functionality for maximizing familiarity is implemented in CGM-Tool.

For example, if we ask CGM-Tool to find a realization which maximizes our notion of familiarity (see section 7.4), we obtain the novel realization μ_2^{fam} depicted in Figure 7.4. μ_2^{fam} satisfies all the requirements (both "nice to have" and mandatory ones), and includes the following tasks:

`CharacteriseMeeitng`,

`SetSystemCalendar`,

`ParticipantsFillSystemCalendar`,

`CollectFromSystemCalendar`,

`GetRoomSuggestions`,

`UseAvailableRoom`,

`RegisterMeetingRoom`,

`ScheduleAutomatically`,

`ConfirmOccurrence`,

`GoodParticipation`,

`MinimalConflicts`,

`CollectionEffort`, and

`MatchingEffort`;

μ_2^{fam} also requires two domain assumptions:

`ParticipantsUseSystemCalendar` and

`LocalRoomAvailable`.

Notice that all the tasks which are satisfied in μ_1 are satisfied also in μ_2^{fam}, and only the intermediate goal `ByUser`, the refinement R_{21} and the four tasks:

`SetSystemCalendar`,

`ParticipantsFillSystemCalendar`,

`UseAvailableRoom`, and

`RegisterMeetingRoom` are added to μ_2^{fam}, three of which are newly-added tasks. Thus, on common elements, μ_2^{fam} and μ_1 differ only on the task `UseAvailableRoom`, which must be mandatorily be satisfied to complete the realization. Overall, wrt. μ_2^{lex}, we pay familiarity with some loss in the “quality” of the realization, since for μ_2^{fam} we have $\texttt{Penalty} - \texttt{Reward} = -50$, $\texttt{workTime} = 3.5h$ and $\texttt{cost} = 0e$. This realization was found automatically by our CGM-Tool in 0.067 seconds on an Apple MacBook Air laptop.

7.3.3 Minimizing change effort

In our approach, in its simplest form, the *change effort* of μ_2 wrt. μ_1 is given by the number of newly-satisfied tasks, i.e., the amount of the new tasks which are satisfied in μ_2 plus that of common tasks which were not satisfied in μ_1 but are satisfied in μ_2. In a more sophisticate form, the contribution of each task of interest can be weighted by some numerical value (e.g., `Penalty`, `cost`, `WorkTime`,...). Intuitively, since satisfying a task requires effort, this value considers the extra effort required to implement μ_2. (Notice that tasks which pass from satisfied to denied do not reduce the effort, because we assume they have been implemented anyway.) This is formalized in section 7.4, and a functionality for minimizing change effort is implemented in CGM-Tool.

For example, if we ask CGM-Tool to find a realization which minimizes the number of newly-satisfied tasks, we obtain the realization μ_2^{eff} depicted in Figure 7.5. The realization satisfies all the requirements (both "nice to have" and mandatory), and includes the following tasks:

`CharacteriseMeeitng`,

`SetSystemCalendar`,

`ParticipantsFillSystemCalendar`,

`CollectFromSystemCalendar`,

`UsePartnerInstitutions`,

`ScheduleAutomatically`,

`ConfirmOccurrence`,

`GoodParticipation`,

`MinimalConflicts`,

`CollectionEffort`, and

`MatchingEffort`;

μ_2^{eff} also requires one domain assumption

`ParticipantsUseSystemCalendar`.

Notice that, in order to minimize the number of new tasks needed to be achieved, in μ_2^{eff}, `FindASuitableRoom` is refined by R_3 instead of R_5. In fact, in order to achieve R_5, we would need to satisfy two extra tasks (`UseAvailableRoom` and `RegisterMeetingRoom`) wrt. μ_1, whilst for satisfying R_3 we only need to satisfy one task (`UsePartnerInstitutions`). Besides, two newly added tasks `SetSystemCalendar` and `ParticipantsFillSystemCalendar` are also included in μ_2^{eff}. Thus the total effort of evolving from μ_1 to μ_2^{eff} is to implement three new tasks. Overall, for μ_2^{eff} we have $\texttt{Penalty}-\texttt{Reward} = -50$, $\texttt{workTime} = 3.5h$ and $\texttt{cost} = 80e$. This realization was found automatically by our CGM-Tool in 0.085 seconds on an Apple MacBook Air laptop.

7.3.4 Combining familiarity or change effort with other objectives

In our approach, familiarity and change effort are numerical objectives like others, and as such they can be combined lexicographically with other objectives, so that stakeholders can decide which objectives to prioritize.

7.4 Automated Reasoning with Evolution Requirements

7.4.1 Evolution Requirements

Here we formalize the notions described in section 7.3. Let $\mathcal{M}_1 \overset{\text{def}}{=} \langle \mathcal{B}_1, \mathcal{N}_1, \mathcal{D}_1, \Psi_1 \rangle$ be the original model, μ_1 be some realization of $\mathcal{M}_1$ and $\mathcal{M}_2 \overset{\text{def}}{=} \langle \mathcal{B}_2, \mathcal{N}_2, \mathcal{D}_2, \Psi_2 \rangle$ be a new version of $\mathcal{M}_1$. We look for a novel realization μ_2 for $\mathcal{M}_2$.

We assume that the system of the original model $\mathcal{M}_1$ is already built based on the realization μ_1. Over the time, the model $\mathcal{M}_1$ evolves and becomes a new model $\mathcal{M}_2 \overset{\text{def}}{=} \langle \mathcal{B}_2, \mathcal{N}_2, \mathcal{D}_2, \Psi_2 \rangle$. In the evolution requirement engineering problem, we want to find a realization μ_2 for $\mathcal{M}_2$ such that we can make the most use out of μ_1. The definition for "most use" can be varied, it can be the familiarity between the two realization μ_1 and μ_2, or the effort needed to implement the system from μ_2 provided that we already have μ_1 implemented.

Stakeholders can select a subset of the elements, called *elements of interest*, on which to focus, which can be requirements, tasks, domain assumptions, and intermediate goals. (When not specified otherwise, we will assume by default that all elements are of interest.) Let $\mathcal{E}^* \subseteq \mathcal{E}_1 \cup \mathcal{E}_2$ be the subset of the elements of interest, and let $\mathcal{E}_1^* \overset{\text{def}}{=} \mathcal{E}^* \cap \mathcal{E}_1$ and $\mathcal{E}_2^* \overset{\text{def}}{=} \mathcal{E}^* \cap \mathcal{E}_2$ be the respective subsets of $\mathcal{M}_1$ and $\mathcal{M}_2$. We define $\mathcal{E}_{common}^* \overset{\text{def}}{=} \{E_i \in \mathcal{E}_2^* \cap \mathcal{E}_1^*\}$ as the set of elements of interest occurring in both $\mathcal{M}_1$ and $\mathcal{M}_2$, and $\mathcal{E}_{new}^* \overset{\text{def}}{=} \{E_i \in \mathcal{E}_2^* \setminus \mathcal{E}_1^*\}$ as the set of new elements of interest in $\mathcal{M}_2$.

Familiarity. In its simplest form, the cost of familiarity can be defined as follows:

$$\begin{aligned} &(7.1) \qquad \mathsf{F}amiliarityCost(\mu_2|\mu_1) &\overset{\text{def}}{=}& \mid \{E_i \in \mathcal{E}_{common}^* \mid \mu_2(E_i) \neq \mu_1(E_i)\} \mid \\ &(7.2) & + & \mid \{E_i \in \mathcal{E}_{new}^* \mid \mu_2(E_i) = \top\} \mid, \end{aligned}$$

where $\mid S \mid$ denotes the number of elements of a set S. $\mathsf{F}amiliarityCost(\mu_2|\mu_1)$ is the sum of two components:

(7.1) the number of common elements of interest (e.g., tasks) which were in μ_1 and are no more in μ_2, plus the number of these which were not in μ_1 and now are in μ_2,

(7.2) the number of new elements of interest which are in μ_2.

In a more sophisticate form, each element of interest E_i can be given some rational weight value w_i [2], so that the cost of familiarity can be defined as follows:

$$\begin{aligned}
&(7.3) \quad \mathsf{W}eightFamiliarityCost(\mu_2|\mu_1) \overset{\text{def}}{=} \sum_{E_i \in \mathscr{E}^*_{common}} w_i \cdot \mathsf{I}nt(\mu_2(E_i) \neq \mu_1(E_i))\\
&(7.4) \quad + \sum_{E_i \in \mathscr{E}^*_{new}} w_i \cdot \mathsf{I}nt(\mu_2(E_i) = \top),
\end{aligned}$$

where Int() converts true and false into the values 1 and 0 respectively.

Both forms are implemented in CGM-Tool. (Notice that (7.1) and (7.2), or even (7.3) and (7.4), can also be set as distinct objectives in CGM-Tool.) Consequently, a realization μ_2 maximizing familiarity is produced by invoking the OMT solver on the formula $\Psi_{\mathcal{M}_2}$ and the objective $\mathsf{F}amiliarityCost(\mu_2|\mu_1)$ or $\mathsf{W}eightFamiliarityCost(\mu_2|\mu_1)$ to minimize.

Change effort. We restrict the elements of interest to tasks only. In its simplest form, the change effort can be defined as follows:

$$\begin{aligned}
&(7.5) \quad \mathsf{C}hangeEffort(\mu_2|\mu_1) \overset{\text{def}}{=} \mid \{T_i \in \mathscr{E}^*_{common} \mid \mu_2(T_i) = \top, \text{ and } \mu_1(T_i) = \bot\} \mid\\
&(7.6) \quad + \mid \{T_i \in \mathscr{E}^*_{new} \mid \mu_2(T_i) = \top\} \mid .
\end{aligned}$$

$\mathsf{C}hangeEffort(\mu_2|\mu_1)$ is the sum of two components:

(7.5) is the number of common tasks which were not in μ_1 and which are now in μ_2,

(7.6) is the number of new tasks which are in μ_2.

As above, in a more sophisticate form, each task of interest T_i can be given some rational weight value w_i, so that the change effort can be defined as follows:

$$\begin{aligned}
\mathsf{W}eightChangeEffort(\mu_2|\mu_1) \overset{\text{def}}{=} & \sum_{T_i \in \mathscr{E}^*_{common}} w_i \cdot \mathsf{I}nt(\mu_2(T_i) = \top) \cdot \mathsf{I}nt(\mu_1(T_i) = \bot)\\
& + \sum_{T_i \in \mathscr{E}^*_{new}} w_i \cdot \mathsf{I}nt(\mu_2(T_i) = \top).
\end{aligned}$$

[2] Like `Penalty`, `Cost` and `WorkTime` in Figure 7.3.

Both forms are implemented in CGM-Tool. Consequently, a novel realization μ_2 minimizing change effort is produced by invoking the OMT solver on the formula $\Psi_{\mathcal{M}_2}$ and the objective $ChangeEffort(\mu_2|\mu_1)$ or $WeightChangeEffort(\mu_2|\mu_1)$.

Notice an important difference between (7.1) and (7.5), even if the former is restricted to tasks only: a task which is satisfied in μ_1 and is no more in μ_2 worsens the familiarity of μ_2 wrt. μ_1 (7.1), but it does not affect its change effort (7.5), because it does not require implementing one more task.

7.4.2 Comparison wrt. previous approaches

Importantly, Ernst et al. [EBMJ12] proposed two similar notion of familiarity and change effort for (un-)constrained goal graphs:

familiarity: maximize (the cardinality of) the set of tasks used in the previous solution;

change effort: (i) minimize (the cardinality of) the set of new tasks in the novel realization –or, alternatively, (ii) minimize also the number of tasks.

We notice remarkable differences of our approach wrt. the one in [EBMJ12].

First, our notion of familiarity presents the following novelties:

(i) it uses all kinds of elements, on stakeholders' demand, rather than only tasks;

(ii) it is (optionally) enriched also with (7.2);

(iii) (7.1) is sensitive also to tasks which were in the previous realization and which are not in the novel one, since we believe that also these elements affect familiarity.

Also, in our approach both familiarity and change effort allow for adding *weights* to tasks/elements, and to combine familiarity and change-effort objectives lexicographically with other user-defined objectives.

Second, unlike with [EBMJ12], in which the optimization procedure is hardwired, we rely on logical encodings of novel objectives into OMT($\mathcal{LRA}$) objectives, using OPTIMATHSAT as workhorse reasoning engine. Therefore, new objectives require implementing no new reasoning procedure, only new OMT($\mathcal{LRA}$) encodings. For instance, we could easily implement also the notion of familiarity of [EBMJ12] by asking OPTIMATHSAT to minimize the objective: $|\{T_i \in \mathcal{E}^*_{common} \mid \mu_2(T_i) = \bot, \text{ and } \mu_1(T_i) = \top\}|$.

Third, our approach deals with CGMs, which are very expressive formalisms, are enriched by Boolean and numerical constraints, and are supported by a tool (CGM-Tool) with efficient search functionalities for optimum realizations. These functionalities, which are enabled by state-of-the-art SMT and OMT technologies [ST15a, ST15b], scale very well, up to thousands of elements, as shown in the empirical evaluation of chapter 9

Fourth, unlike with [EBMJ12], where realizations are intrinsically supposed to be *minimal*, in our approach minimality is an objective stakeholders can set and obtain as a byproduct of *minimum* solutions, but it is not mandatory. This fact is relevant when dealing with familiarity evolution requirements, because objective (7.1) can conflict with minimality, because it may force the presence of tasks from the previous solution which have become redundant in the new model. Thus, sometimes CGM-tool may return a non-minimal model if the stakeholder prioritizes familiarity above all other objectives.

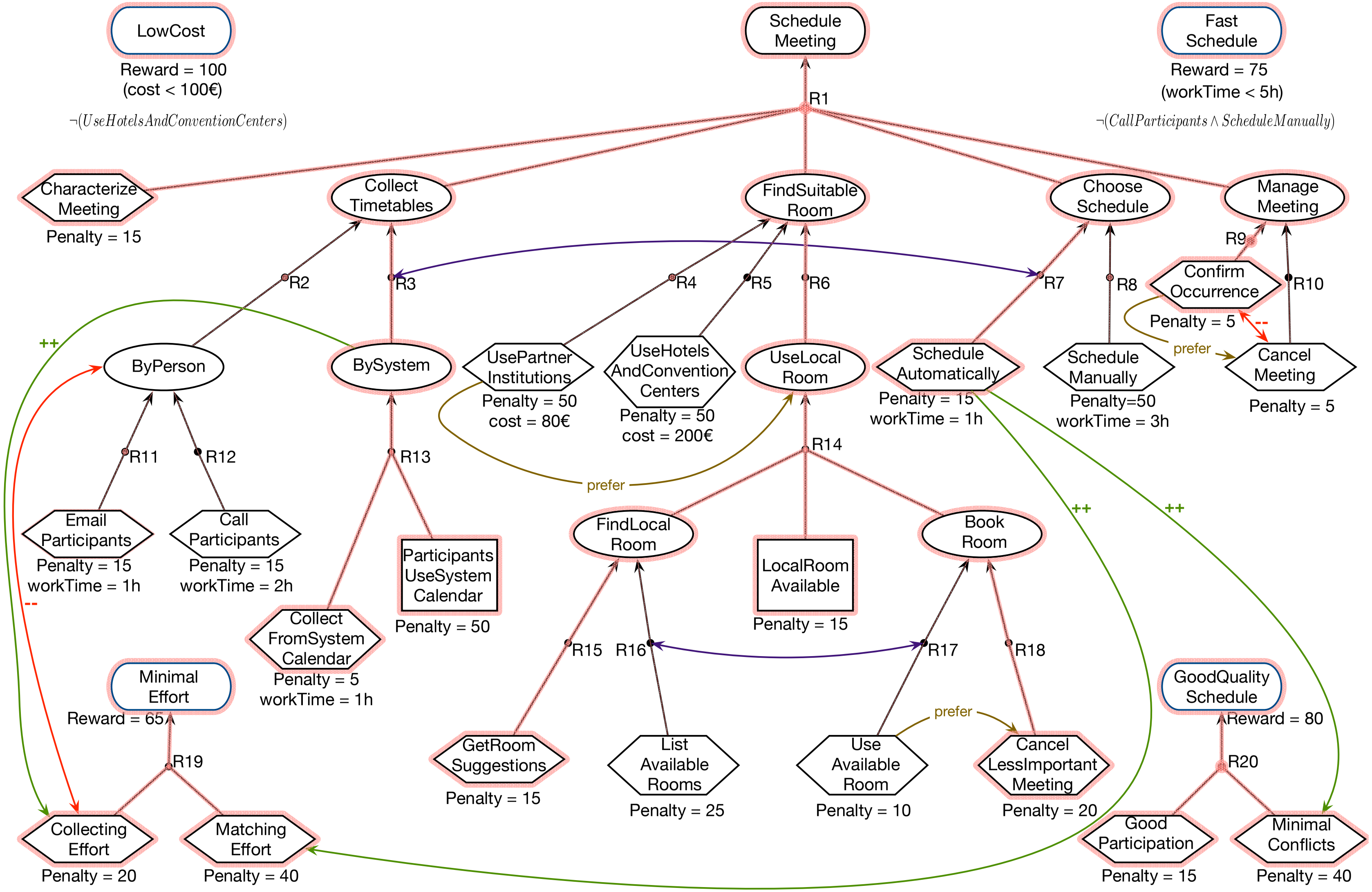


Figure 7.1: The original CGM and its already implemented realization. The realization is highlighted in hue, the denied elements are visible but not highlighted.

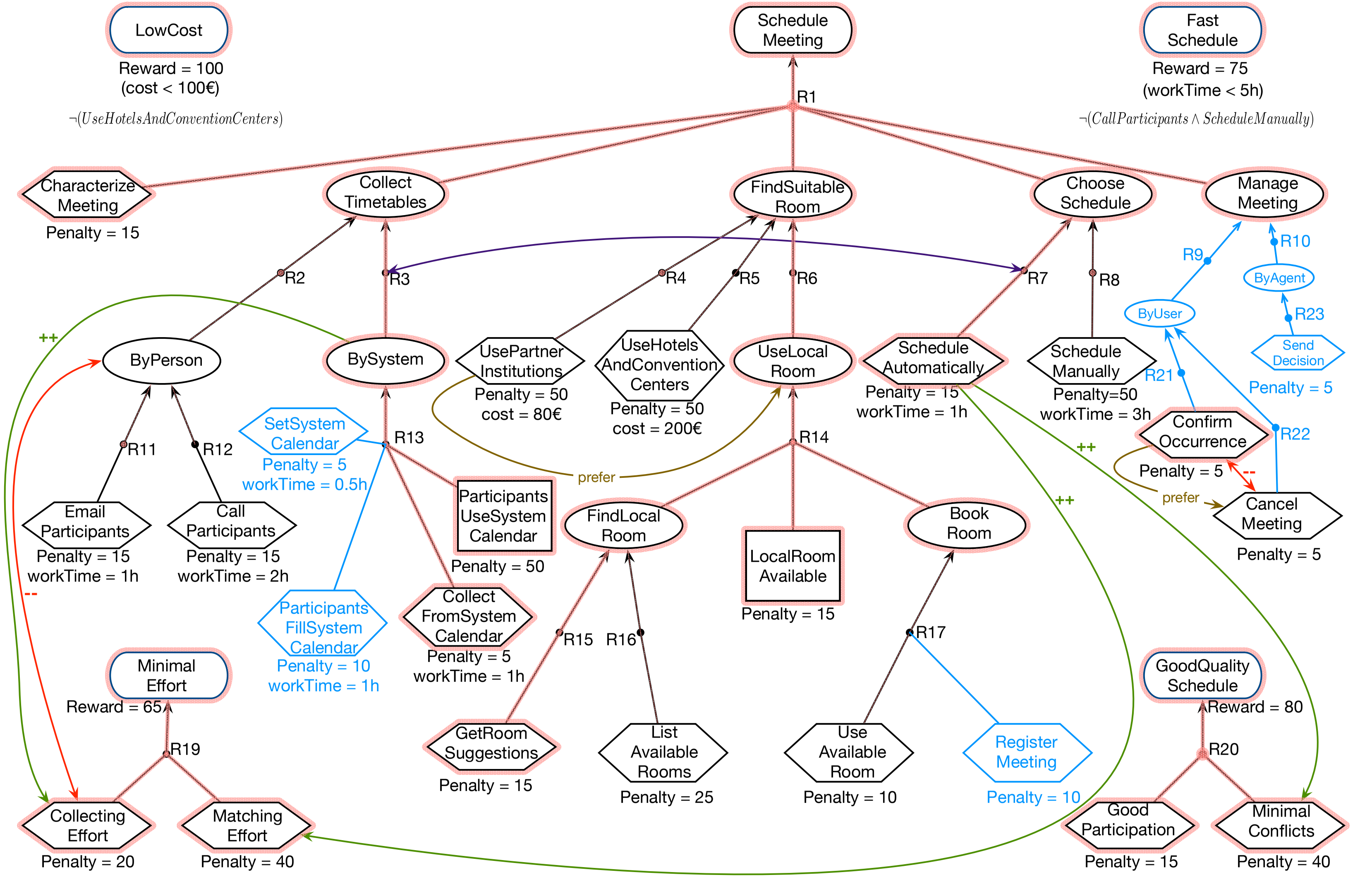


Figure 7.2: An example of requirements evolution in a CGM. The highlighted part of the CGM is the part of its original realization (i.e., already implemented solution), while the light blue part shows the changes (modified or newly added).

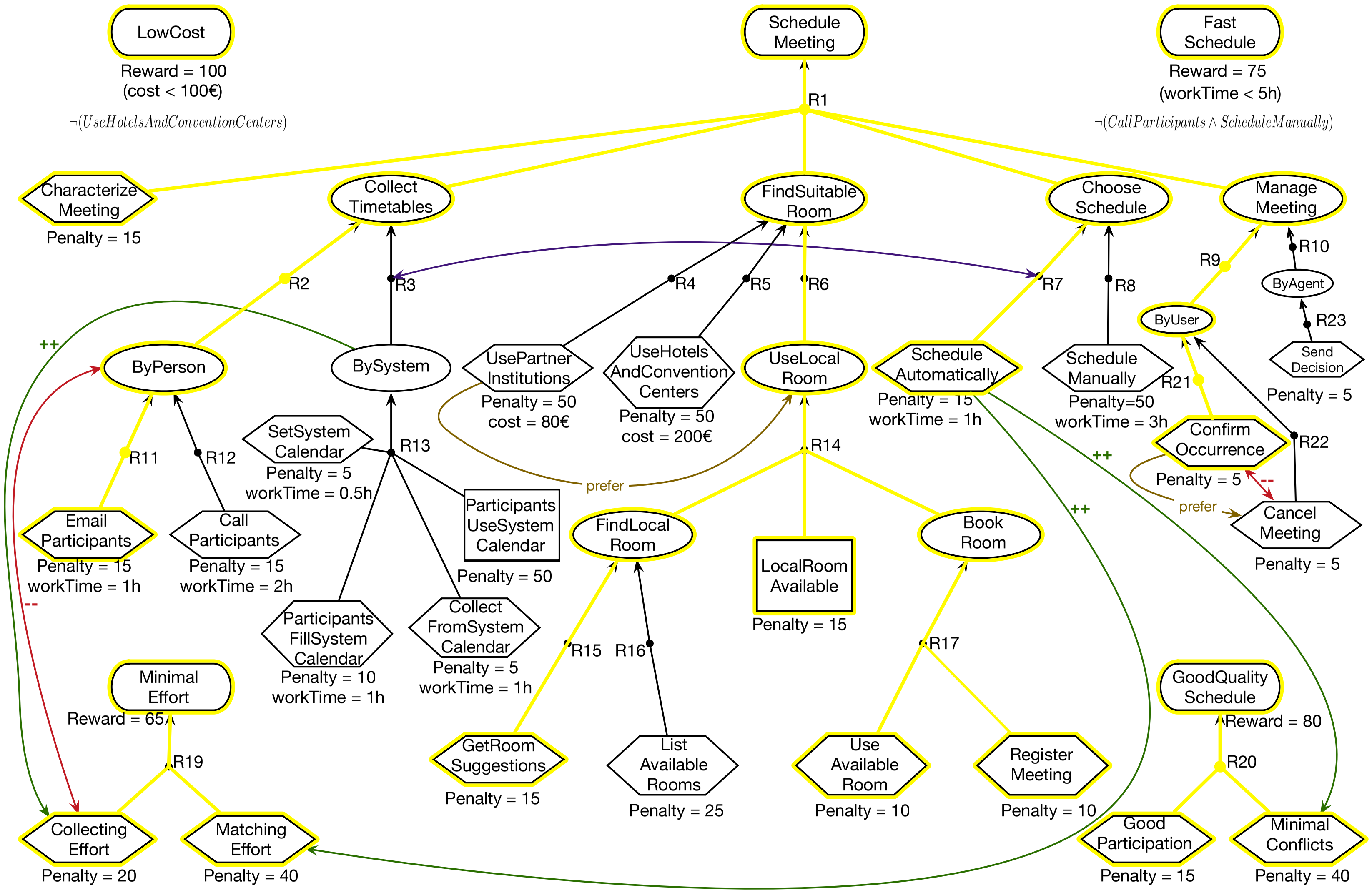


Figure 7.3: An example of evolved CGM and its new realization. This is a realization which optimizes the three objectives [(Penalty − Reward), workTime, cost] in lexicographic order.

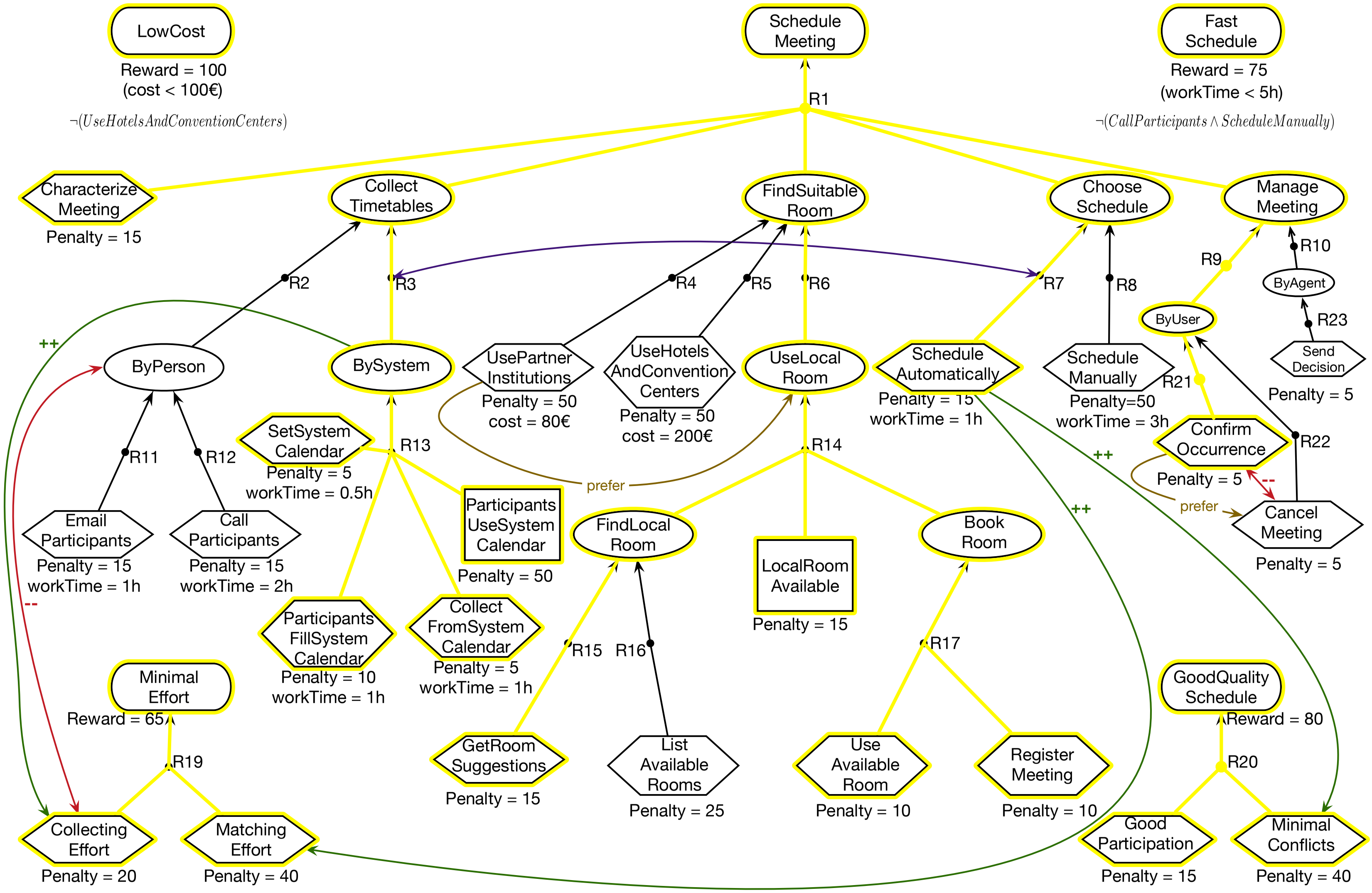


Figure 7.4: An example of evolved CGM and its new realization. This is a realization which maximizes the familiarity between the new realization and the implemented realization of the original CGM.

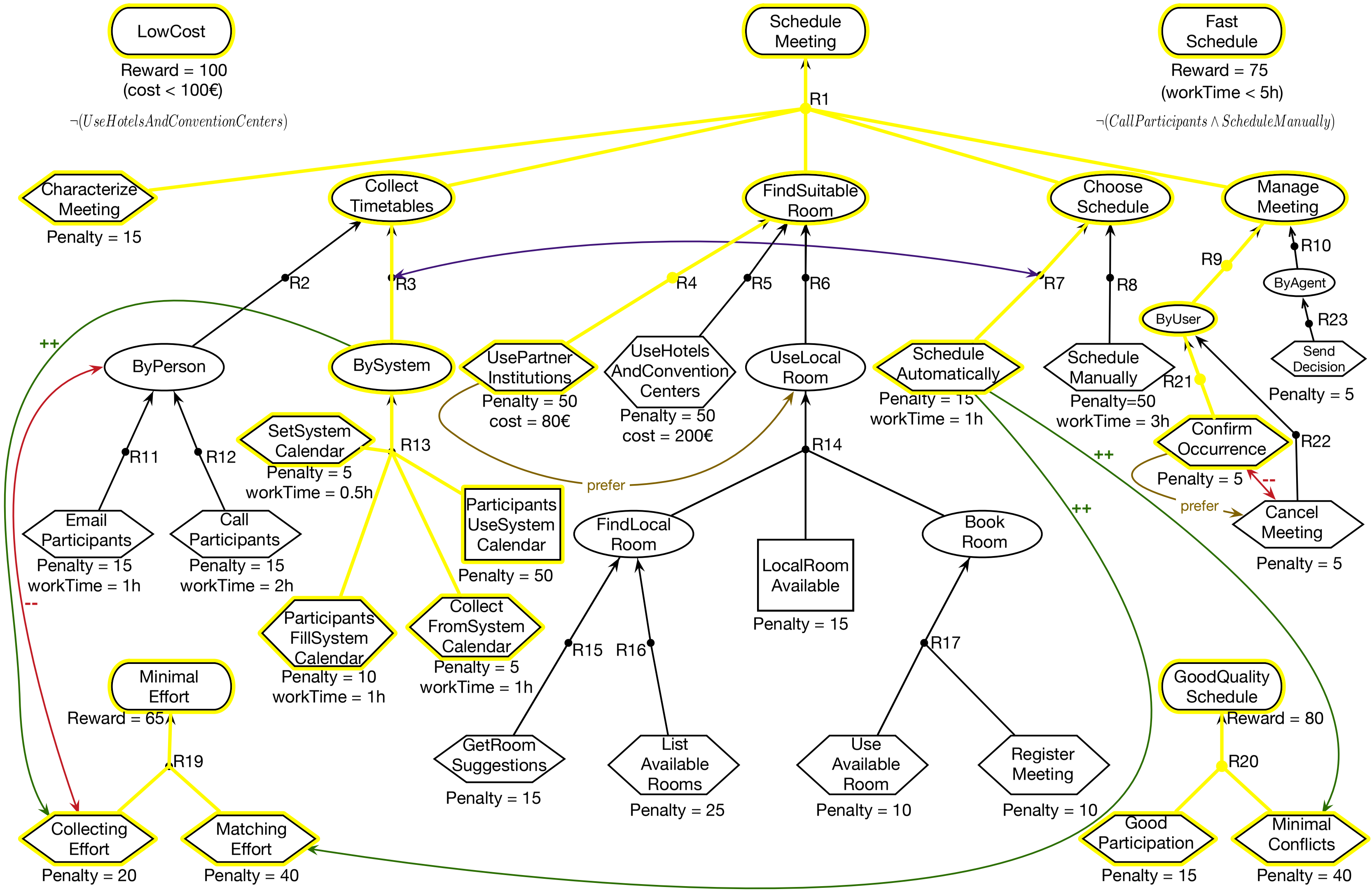


Figure 7.5: An example of evolved CGM and its new realization. This is a realization which minimizes the effort of implementing the new realization with respect to the implemented realization of the original CGM (i.e., minimizes the number of newly satisfied tasks).

Part III

Implementation and Evaluation

CHAPTER 8

IMPLEMENTATION

This chapter presents CGM-Tool, which was implemented based on the CGM framework. CGM-Tool provides graphical constructs that support for modelling and reasoning on CGMs. The tool also supports the evolution analysis based on the requirements evolution and evolution requirements discussed in chapter 7. The tool (and its manual) is available online (www. cgm-tool. eu).

* * ★ * *

The chapter is organized as follows: section 8.1 introduces CGM-Tool, presents an overview of its features, and describes its modular architechture; section 8.2 shows how to use the tool through examples and screenshots.

8.1 CGM-Tool

CGM-Tool provides support for modelling and reasoning on CGMs. Technically, CGM-Tool is a standalone application written in Java and its core is based on Eclipse RCP engine. Under the hood, it encodes CGMs and invokes the OptiMathSAT [1] SMT/OMT solver [ST15b] to support reasoning on goal models. It is freely distributed as a compressed archive file for multiple platforms [2]. CGM-Tool supports:

[1] http://optimathsat.disi.unitn.it

[2] http://www.cgm-tool.eu/

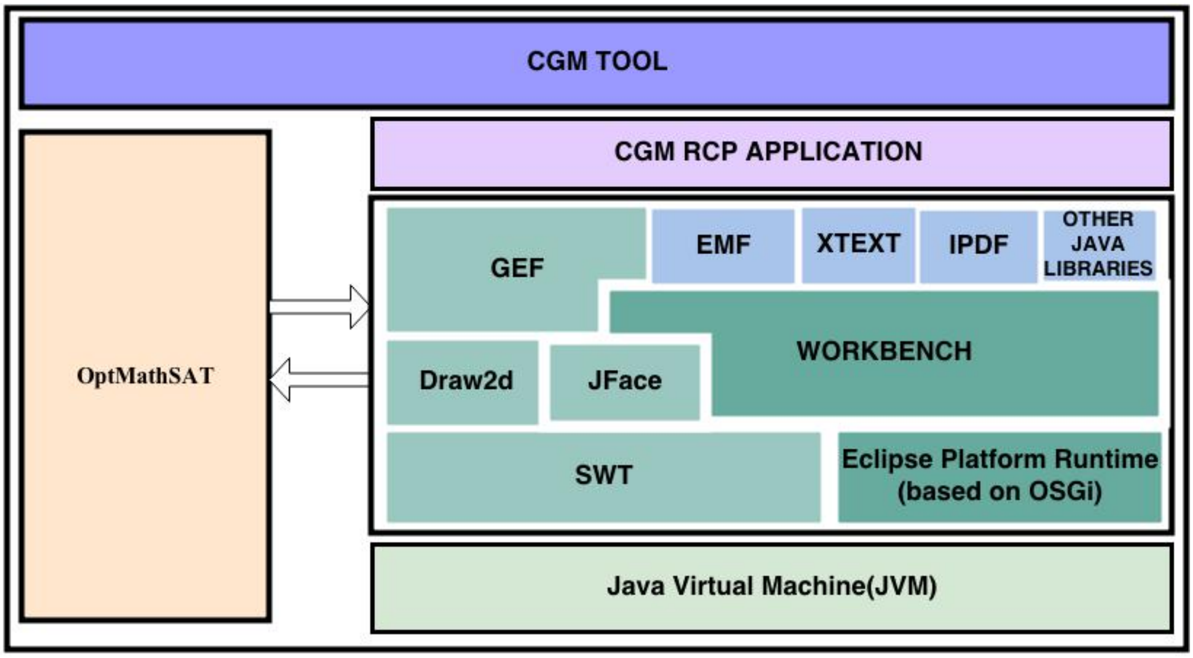


Figure 8.1: CGM-Tool: Component View

Specification of projects: CGMs are created within the scope of project containers. A project contains a set of CGMs that can be used to generate reasoning sessions with OptiMathSAT (i.e., scenarios);

Diagrammatic modelling: the tool enables the creation (drawing) of CGMs in terms of diagrams; furthermore it enhances the modelling process by providing real-time check for refinement cycles and by reporting invalid refinement, contribution and binding links;

Consistency/well-formedness check: CGM-Tool allows for the creation of diagrams conform with the semantics of the modelling language by providing the ability to run consistency analysis on the model;

Automated Reasoning: CGM-Tool provides the automated reasoning functionalities of section 6.2 by encoding the model into an SMT formula. The results of OptiMathSAT are shown directly on the model as well as in a tabular form.

Evolution Requirements Modelling and Automated Reasoning: by means of scenarios, stakeholders can generate *evolution sessions*, which allows for (i) defining the first model and finding the first optimal realization, (ii) modifying the model to obtain the new models, and (iii) generating automatically the "similar" realization (as discussed in section 7.3).

One essential feature of the tool is that expressive constructs (which may be more complex and difficult to use) are only available on demand: there are easy-to-use default settings for everything, so that the user can decide the level of expressiveness he/she feels at ease with.

CGM-Tool extends the STS-Tool [PDP$^+$12] as an RCP application by using the major frameworks shown in Figure 8.1: *Rich Client Platform (RCP)*, a platform for building rich client applications, made up of a collection of low level frameworks such as OSGi, SWT, JFace and Equnix, which provide us a workbench where to get things like menus, editors and views; *Graphical Editing Framework (GEF)*, a framework used to create graphical editors for graphical modelling tools (e.g., tool palette and figures which can be used to graphically represent the underlying data model concepts); *Eclipse Modelling Framework (EMF)*, a modelling framework and a code generation facility for building tools and applications based on a structured data model. With CGM-Tool, a CGM is built progressively as a sequence of *scenarios*, which are versions of the CGM to which the automated reasoning functionalities of the CGM-Tool can be applied.

8.2 An example

Figure 8.2 shows how to create a new diagram in the tool. Figure 8.3 shows the graphical user interface (GUI) of the tool.

In order to create a CGM, the user will need first to identify the elements and draw the goal graph. Figure 8.4 shows the graphical presentation of CGM in CGM-Tool.

Notice that the element name must contain only alphanumeric characters [A..Z], [a...z], [0..9] and underscore [_] (no space or special characters). The user can use the description properties tab to better describe the elements, as showed in Figure 8.5.

As mentioned in chapter 4 and chapter 5, refinement edges must not form a cycle with each other, domain assumption can only be refined into sub-domain assumption(s), goal refinement must contain at least one sub-goal, all root goals must be requirements, and all leaf goals must be tasks.

Figure 8.6 and Figure 8.7 show respectively how to define a numerical attribute of an element and how to set its value. Figure 8.8 shows how to set objective functions from the numerical attributes (e.g., set the priorities, choose the form of optimization (maximize/minimize), . . .). Notice that:

- Each element has its own local numerical attributes (i.e. local variables), which are showed in the Node Variable tab below the model.
- The values of the "local" numerical attributes that are associated with an element can be set (default value is 0).
- The total value of the "global" numerical attribute is the sum of all the "local" numerical attributes.
- When an element is satisfied, its local variables is set to its *positive value*, otherwise, the local variables is set to its *negative value*.

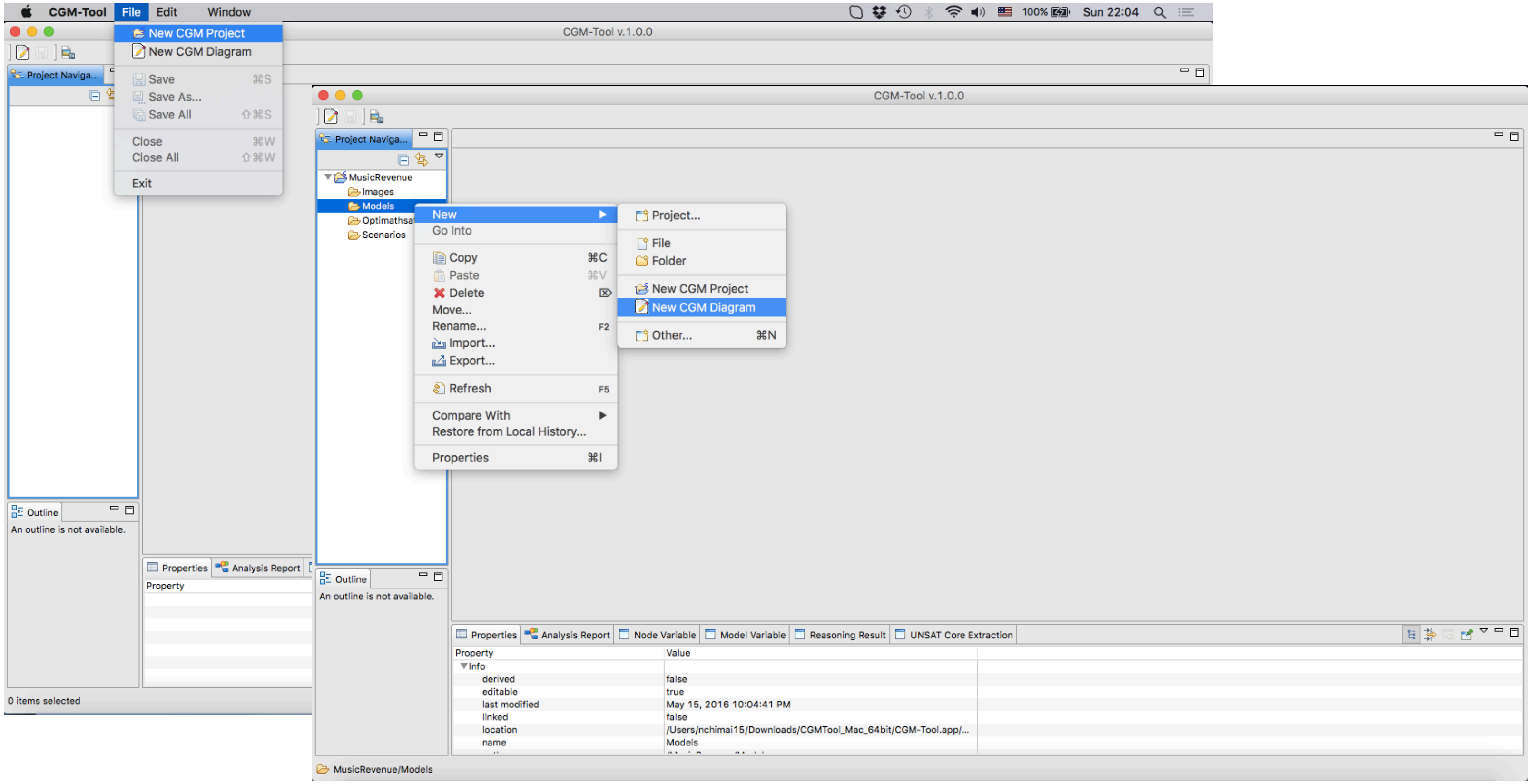
Figure 8.9 shows how to define the global constraints in the model. The constraints can be *Boolean* or *SMT*($\mathscr{LRA}$) formulas.

Figure 8.10 shows how to define the optimization priorities of the numerical attributes (the lower the priority value of a attribute, the higher its optimization priority); it also shows how to define the optimization choices: either to maximize or minimize. Figure 8.11 shows how to check for the well-formedness of the model.

In order to do automated requirements analysis, a scenario of the CGM must be created. Figure 8.12 and Figure 8.13 show how to create and open a scenario. Figure 8.14 shows how the user assertions can be added by using the option “Force True" (element that must be included in the realization) and “Force False" (element that must not be included in the realization). Mandatory requirements must be set by “Force True”. Figure 8.15 shows how to automatically generate a realization for the current scenario by invoking the automated-reasoning functionalities. Figure 8.16 shows an automatically generated candidate solution of a CGM. The candidate solution is coloured in blue. Figure 8.17 shows how to check for the result analysis. Notice that:

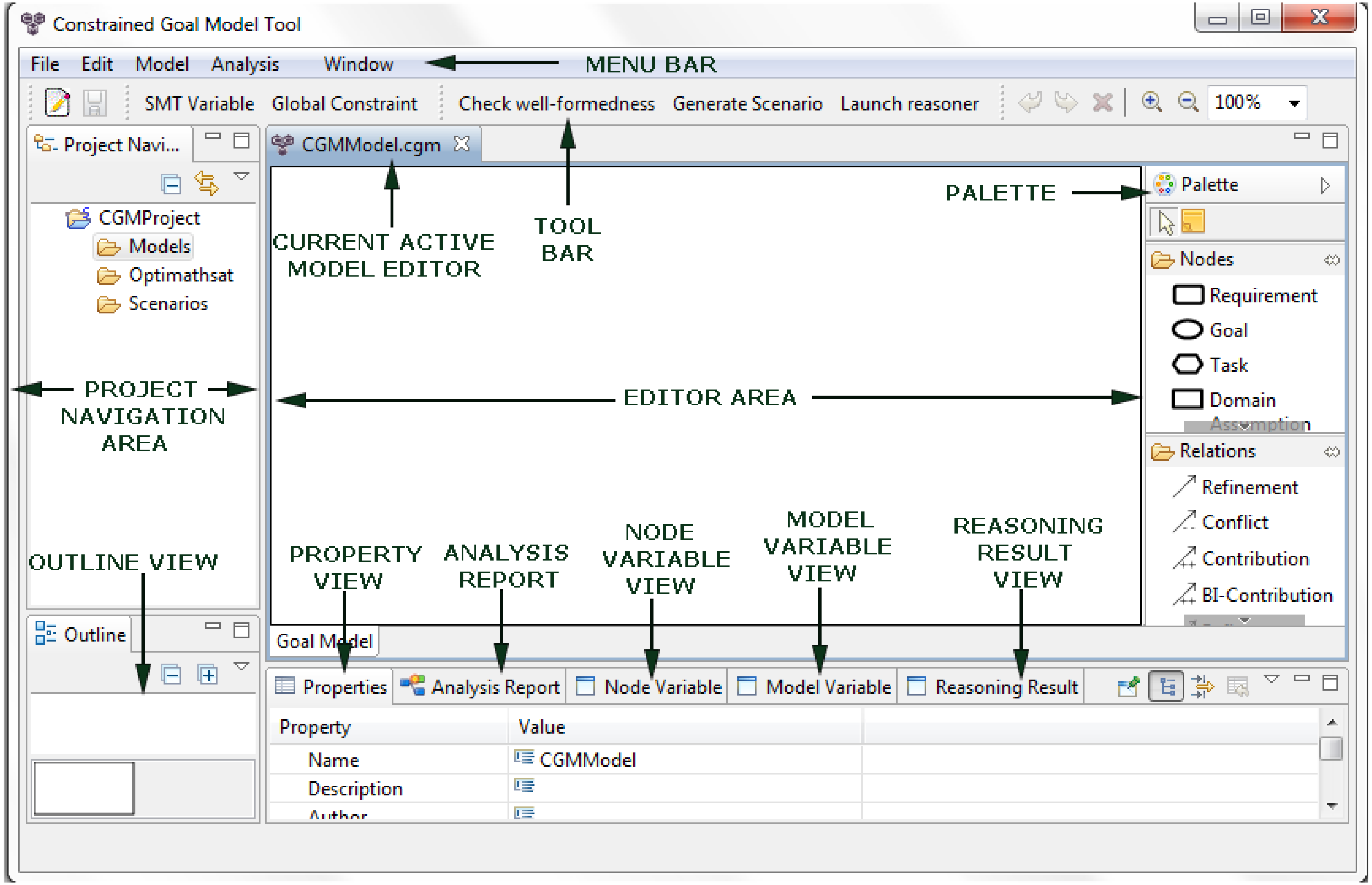
- The tool will automatically check for the well-formedness for once last time after the user chooses "Launch the reasoner".
- The user will also be asked for if he/she wants to generate a report before launching the automatic reasoning.

- The user will get the message if the model is SAT after the reasoning functionalies have finished running.
- "Launch the reasoner" can only be used in a scenario diagram.



Property	Value
▾Info	
derived	false
editable	true
last modified	May 15, 2016 10:04:41 PM
linked	false
location	/Users/nchimai15/Downloads/CGMTool_Mac_64bit/CGM-Tool.app/...
name	Models

Figure 8.2: CGM-Tool: How to create a new diagram.



Property	Value
Name	CGMModel
Description	
Author	

Figure 8.3: CGM-Tool: Graphical User Interface as in the tool manual [Mek] (green notes are for description).

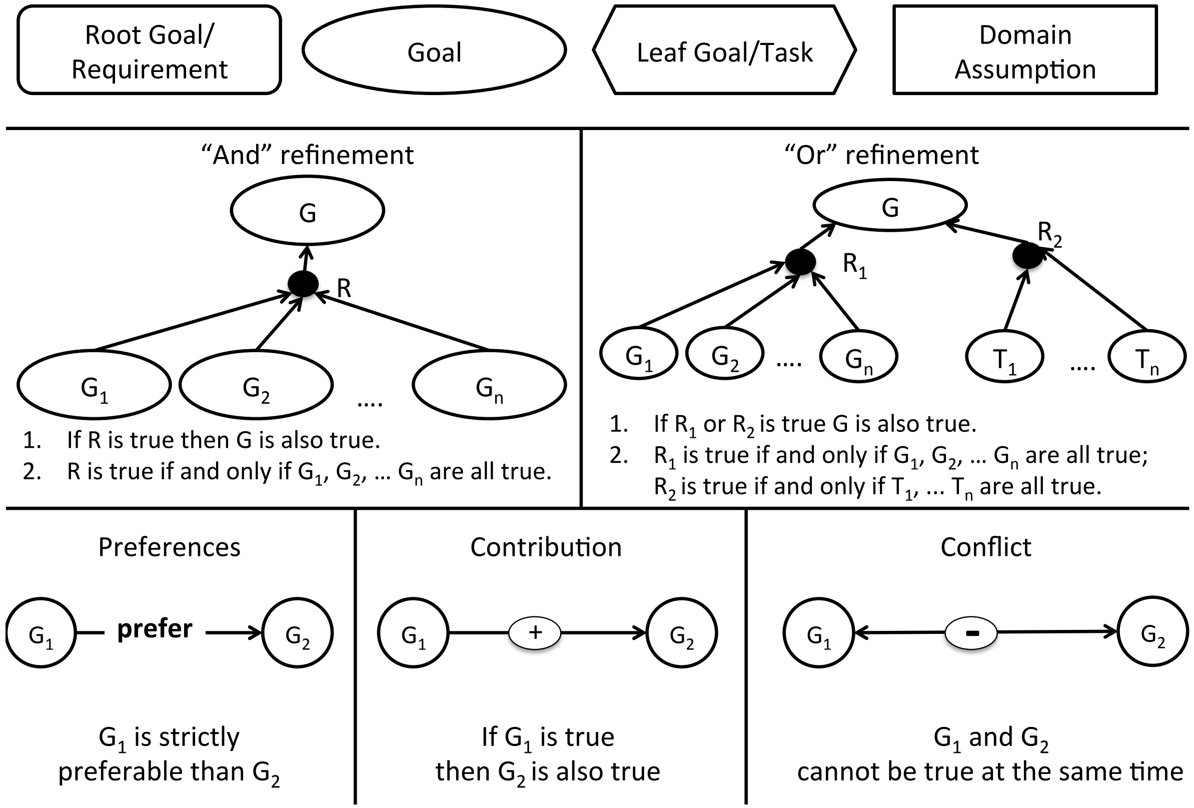
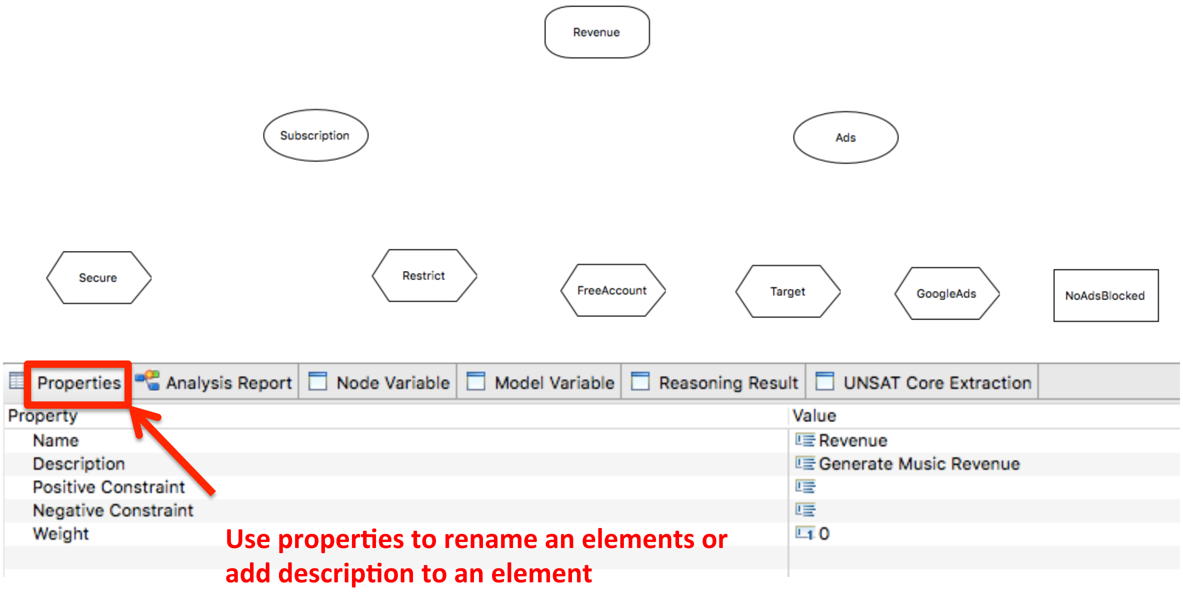


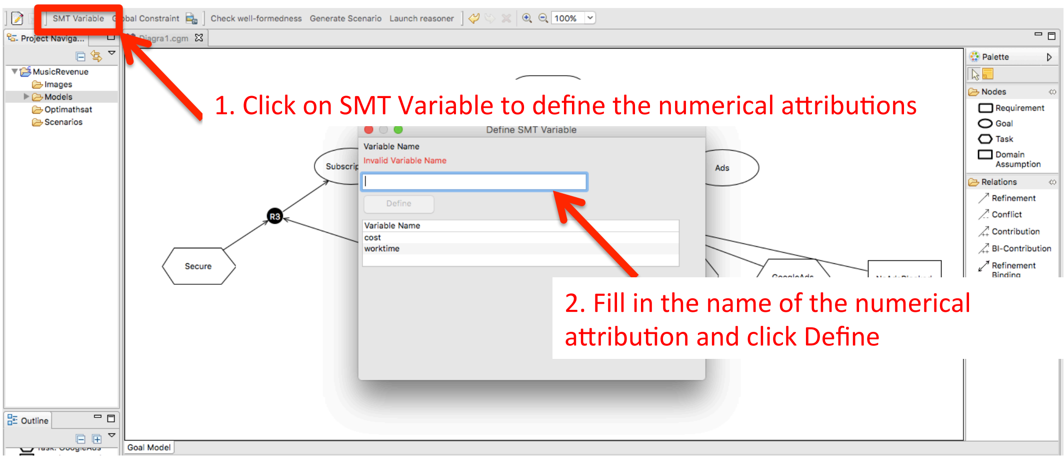
Figure 8.4: CGM-Tool: Graphical Presentation of Elements, Refinements, Relations of a Goal Graph.



Property	Value
Name	Revenue
Description	Generate Music Revenue
Positive Constraint	
Negative Constraint	
Weight	0

Use properties to rename an elements or add description to an element

Figure 8.5: CGM-Tool: How to add description to elements (instructions in red).



1. Click on SMT Variable to define the numerical attributions

Define SMT Variable

Variable Name

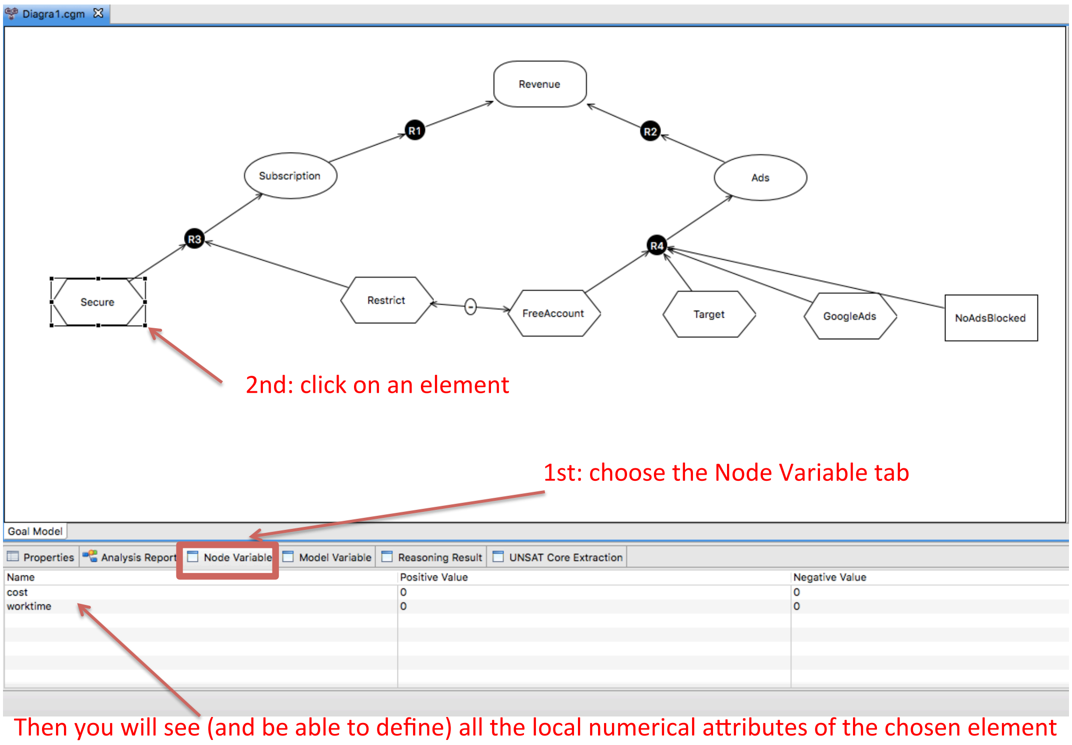
Invalid Variable Name

Define

Variable Name
cost
worktime

2. Fill in the name of the numerical attribution and click Define

Figure 8.6: CGM-Tool: How to Define Numerical Attributes (instructions in red).



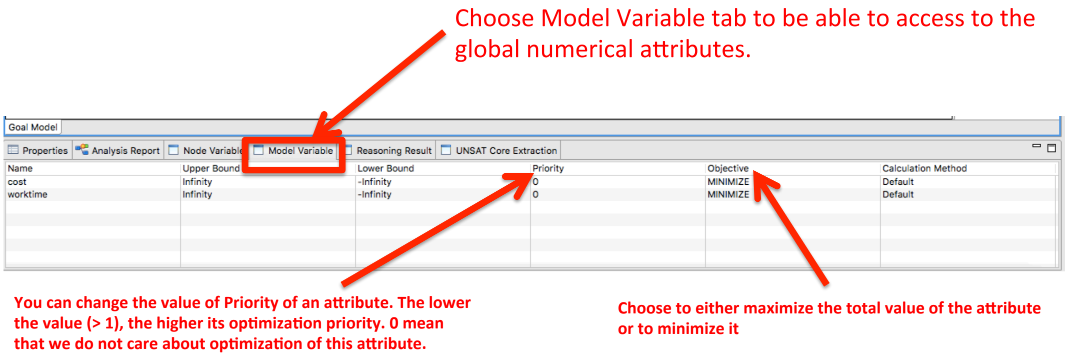
2nd: click on an element

1st: choose the Node Variable tab

Name	Positive Value	Negative Value
cost	0	0
worktime	0	0

Then you will see (and be able to define) all the local numerical attributes of the chosen element

Figure 8.7: CGM-Tool: How to Define the Value of the Numerical Attributes Associated with Elements (instructions in red).



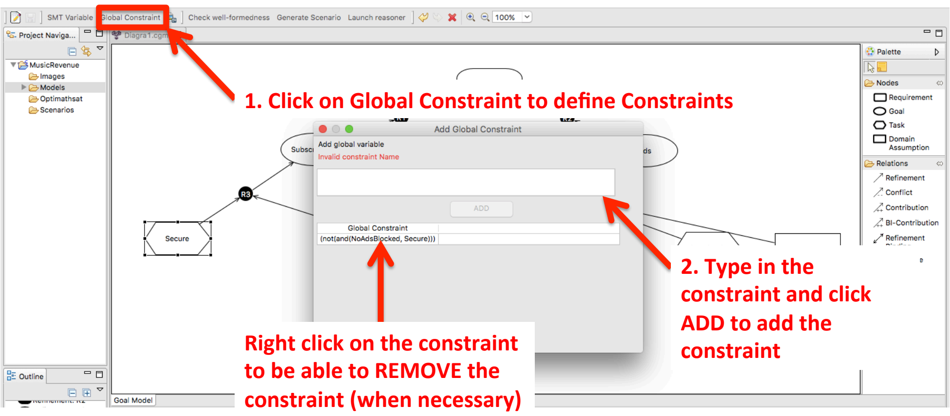
Choose Model Variable tab to be able to access to the global numerical attributes.

Name	Upper Bound	Lower Bound	Priority	Objective	Calculation Method
cost	Infinity	-Infinity	0	MINIMIZE	Default
worktime	Infinity	-Infinity	0	MINIMIZE	Default

You can change the value of Priority of an attribute. The lower the value (> 1), the higher its optimization priority. 0 mean that we do not care about optimization of this attribute.

Choose to either maximize the total value of the attribute or to minimize it

Figure 8.8: CGM-Tool: How to define objectives from Numerical Attributes (instructions in red).



Add Global Constraint

Add global variable

Invalid constraint Name

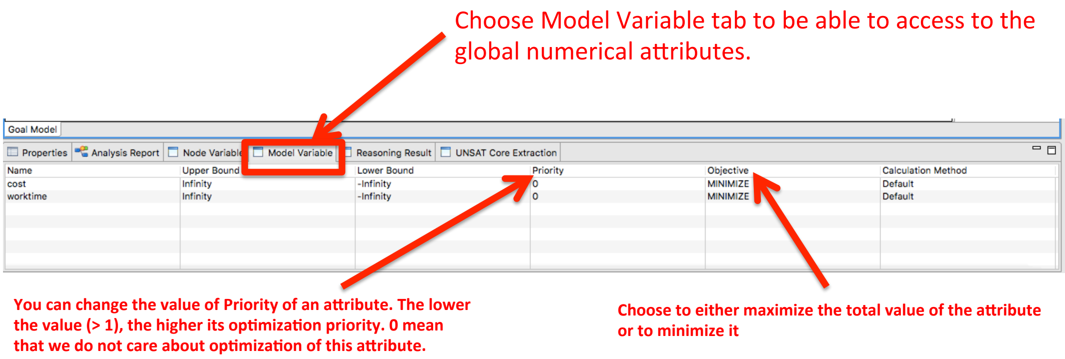
Global Constraint
(not(and(NoAdsBlocked, Secure)))

1. Click on Global Constraint to define Constraints

2. Type in the constraint and click ADD to add the constraint

Right click on the constraint to be able to REMOVE the constraint (when necessary)

Figure 8.9: CGM-Tool: How to Define Global Constraints (instructions in red).



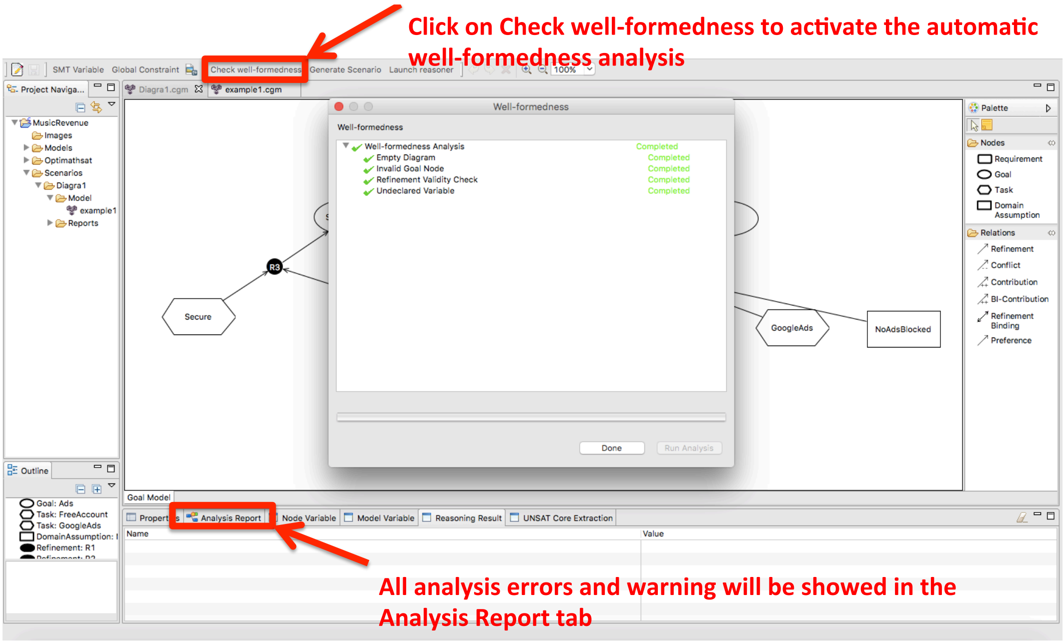
Choose Model Variable tab to be able to access to the global numerical attributes.

Name	Upper Bound	Lower Bound	Priority	Objective	Calculation Method
cost	Infinity	-Infinity	0	MINIMIZE	Default
worktime	Infinity	-Infinity	0	MINIMIZE	Default

You can change the value of Priority of an attribute. The lower the value (> 1), the higher its optimization priority. 0 mean that we do not care about optimization of this attribute.

Choose to either maximize the total value of the attribute or to minimize it

Figure 8.10: CGM-Tool: How to Define Optimization Priorities and Choices (instructions in red).



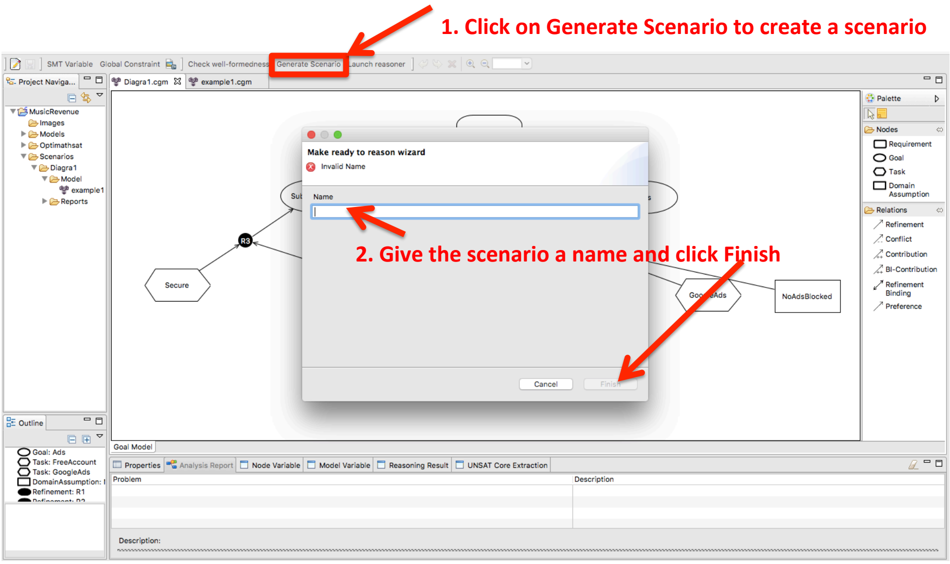
Click on Check well-formedness to activate the automatic well-formedness analysis

Well-formedness

- Well-formedness Analysis — Completed
  - Empty Diagram — Completed
  - Invalid Goal Node — Completed
  - Refinement Validity Check — Completed
  - Undeclared Variable — Completed

All analysis errors and warning will be showed in the Analysis Report tab

Figure 8.11: CGM-Tool: How to Check the Well-formedness of the CGM (instructions in red).



1. Click on Generate Scenario to create a scenario

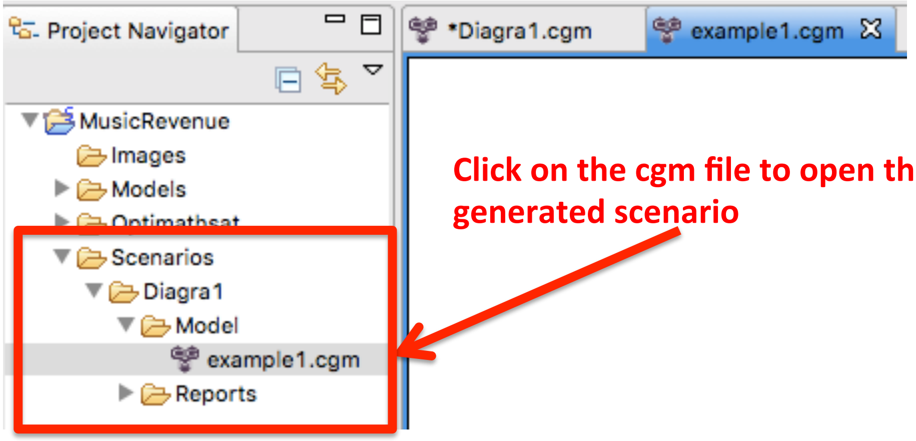
Make ready to reason wizard

Invalid Name

Name

2. Give the scenario a name and click Finish

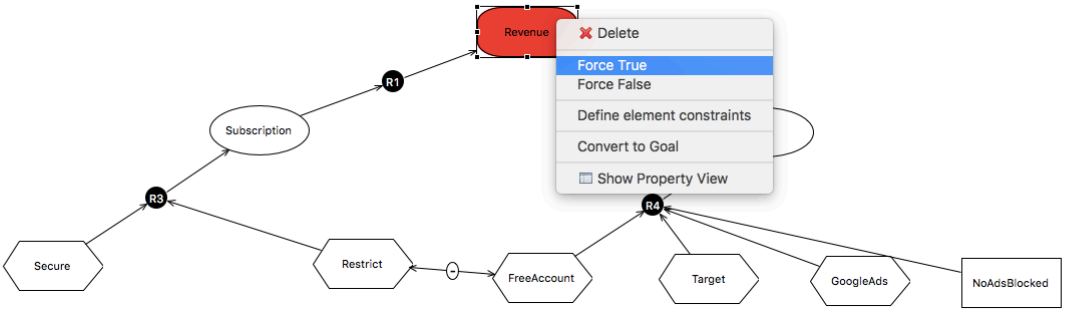
Figure 8.12: CGM-Tool: How to Create a Scenario (instructions in red).



Click on the cgm file to open the generated scenario

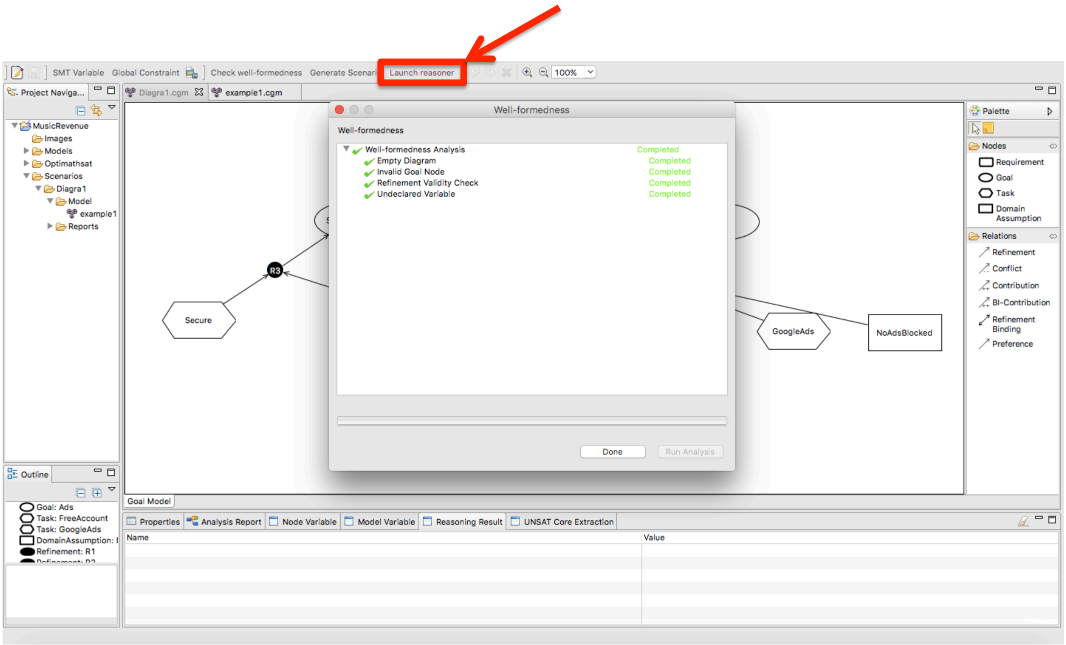
Figure 8.13: CGM-Tool: How to Open the created Scenario (instructions in red).

Right-click on an element to force it as true or false.”



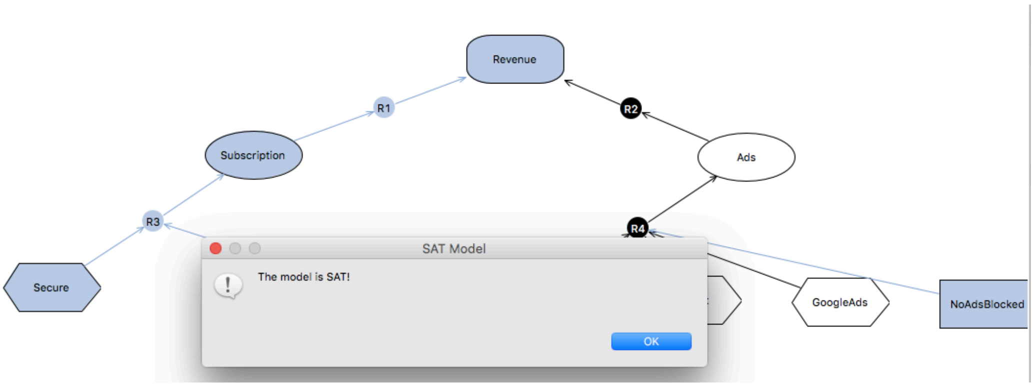
Note that: Mandatory requirements must be “Force True”
“Force True” elements are marked as red, while “Force False” elements are marked as green.

Figure 8.14: CGM-Tool How to Add User’s Assertions (instructions in red).



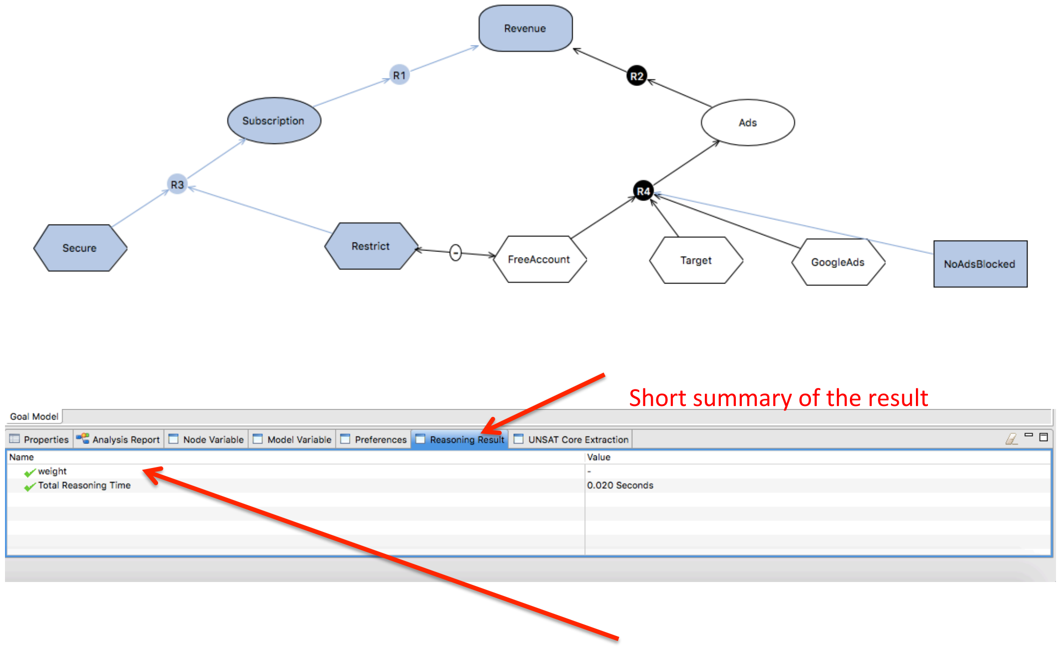
```
Well-formedness
▼ ✓ Well-formedness Analysis      Completed
    ✓ Empty Diagram               Completed
    ✓ Invalid Goal Node           Completed
    ✓ Refinement Validity Check   Completed
    ✓ Undeclared Variable         Completed
```

Figure 8.15: CGM-Tool How to Automatically Generate a Realization: click on *Launch Reasoner* in the menu (instructions in red).



The model is SAT!

Figure 8.16: CGM-Tool: Automated Generated Candidate Solution.



Name	Value
✓ weight	-
✓ Total Reasoning Time	0.020 Seconds

Short summary of the result

Optimal value of the optimization objectives (if set)

Figure 8.17: CGM-Tool: Results Analysis. (instructions in red).

CHAPTER 9

Empirical Experiments

An empirical experiment of CGM-Tool on a large example.

∗ ∗ ⋆ ∗ ∗

This chapter presents the empirical experiments and evaluation of CGM-tool focusing on scalability of the reasoning tool.

9.1 Empirical Experiment

We address the issue of the scalability of the automated-reasoning functionalities of chapter 4 wrt. the size of CGMs, by providing an empirical evaluation of the performance of CGM-Tool on increasingly-large CGMs. (For the sake of readability, here we provide only a qualitative description, whereas the data and plots are reported in an Appendix.) As in chapter 4, all experiments have been run on a MacBook Air laptop, Intel Core i5 1.8 GHz, 2 cores, 256 KB L2 Cache per Core, 3 MB L3 Cache, 4GB RAM.

For the readers' convenience, a compressed directory containing all the material to reproduce these experiments (models, tools, scripts, etc.) is available at `http://www.cgm-tool.eu/experiment-version/`.

We consider first the schedule-meeting CGM of chapter 4 as a seed model. The model consists in 32 goals –among which there are 1 mandatory requirement, 4 nice-to-have requirements, and 18 tasks– plus 20 refinements and 2 domain assumptions, totalling

54 nodes. The CGM contains also 3 numerical objectives: `cost`, `workTime`, and `Weight`. The user-defined objectives `cost` and `workTime` involve respectively 2 and 5 tasks and no requirement, whilst the pre-defined attributes `Weight` involves 16 tasks plus all 4 non-mandatory requirements. This involves $3+2+5+0+0+16+4 = 30$ rational variables (recall Remark 7). There are also three binary preference relations (4.15).

In the example reported in chapter 4 with different configurations, the tool returned the optimal solutions in negligible time (all took less than 0.02 seconds). This is not surprising: as mentioned in chapter 3, in previous empirical evaluation of OMT-encoded problems from formal verification, OptiMathSAT successfully handled optimization problems with up to thousands Boolean/rational variables [ST15a], so that hand-made CGMs resulting into SMT formulas with few tens of Boolean and rational variables, like that in chapter 4, are not a computational challenge.

In perspective, since CGM-Tool is supposed to be used to design CGMs representing possibly-large projects, we wonder how its automated-reasoning functionalities will scale on large models. To do this, we choose to build benchmark CGMs of increasing size, by combining different instances of the schedule-meeting CGM of chapter 4 in various ways, and testing them with different combination of objectives.

9.2 Experiment Setup

In all our experiments CGMs were produced as follows, according to three positive integer parameters N, k, and p, and some choices of objectives.

Given N and k, we pick N distinct instances of the schedule-meeting CGM of chapter 4, each with a fresh set of Boolean labels and rational variables, we create an artificial root goal G with only one refinement R whose source goals are the N mandatory requirements "$\texttt{ScheduleMeeting}_i$" of each CGM instance. Hence, the resulting CGM has $54 \cdot N + 2$ nodes and $30 \cdot N$ rational variables (see Table 9.2). In another group of experiments (see Table 9.1) we dropped the non-mandatory requirements and their 4 direct sub-tasks, so that each instance contains 24 goals, 2 domain assumptions and 18 refinements, and the resulting CGM has $44 \cdot N + 2$ nodes and $26 \cdot N$ rational variables.

Then we randomly add $(k-1) \cdot N$ contribution relations "$\xrightarrow{++}$" and N conflict relations "$\xleftrightarrow{--}$" between tasks belonging to different instances. When binary preference relations are involved (see below), we also randomly add $p \cdot N$ binary preference relations, each involving two refinements of one same goal.

In each group of experiments we fix the definition of the objectives and we set the value of k (and p when it applies), and increase the values of N. For every choice of N, we automatically [1] generate 100 instances of random problems as in the above schema, which we feed to our tool, and collect the median CPU times over the solved instances –including both encoding and solving times– as well as the number of unrealizable instances as well as the number of instances which OptiMathSAT could not solve within a timeout of 1000 seconds.

Notice that, following some ideas from a different context [HPSS00, PSS03], the parameters N, k and p have been chosen so that to allow us to *increase monotonically* and *tune* some essential features of the CGMs under test, which may significantly influence the performances. E.g.,

- N increases linearly the number of Boolean and rational variables,
- k (and, to some extent, p) increases the connectivity of the graph and the ratio between unrealizable and realizable CGMs.
- Importantly, k and p also play an essential role in drastically reducing the *symmetry* of the resulting CGMs, and insert some degree of randomness.

Another important parameter, which we borrowed from the schedule-meeting CGM, is the number of Boolean atoms per objective.

Table 9.3, Table 9.4, Table 9.5, and Table 9.6 show the detailed experiment data of first group of experiment. While, Table 9.7, Table 9.8, and Table 9.9 show the detailed experiment data of second group of experiment.

Remark 8. *We are aware that the CGMs produced with this approach may not represent* realistic *problems. However, we stress the fact that here we focus only on providing a test on the* scalability *of our automated-reasoning functionalities.*

9.3 Experiment Evaluation

We run two groups of experiments in which we focus on optimizing, respectively:

- *numerical attributes*, like cost, work-time, penalty/rewards;
- *discrete features*, like the number of binary preferences, of want-to-have requirements and of tasks to accomplish.

[1] To perform this test automatically, we developed an automated problem generator/manipulator which interfaces directly with the internal data structure representing the CGMs inside CGM-Tool.

Experiment	Number of Instances	Number of Replicas (N)	Number of Goals	Number of Refinements	Number of Domain Assumptions	Total Number of Nodes	Number of Rational Variables
1	100	2	49	37	4	90	52
2	100	3	73	55	6	134	78
3	100	4	97	73	8	178	104
4	100	5	121	91	10	222	130
5	100	6	145	109	12	266	156
6	100	7	169	127	14	310	182
7	100	9	217	163	18	398	234
8	100	11	265	199	22	486	286
9	100	13	313	235	26	574	338
10	100	15	361	271	30	662	390
11	100	17	409	307	34	750	442
12	100	21	505	379	42	926	546
13	100	26	625	469	52	1146	676
14	100	31	745	559	62	1366	806
15	100	36	865	649	72	1586	936
16	100	41	985	739	82	1806	1066
17	100	46	1105	829	92	2026	1196
18	100	51	1225	919	102	2246	1326
19	100	101	2425	1819	202	4446	2626
20	100	151	3625	2719	302	6646	3926
21	100	201	4825	3619	402	8846	5226

Table 9.1: First group of experiments, summary of experimental data.

Experiment	Number of Instances	Number of Replicas (N)	Number of Goals	Number of Refinements	Number of Domain Assumptions	Total Number of Nodes	Number of Rational Variables
1	100	2	65	41	4	110	60
2	100	3	97	61	6	164	90
3	100	4	129	81	8	218	120
4	100	5	161	101	10	272	150
5	100	6	193	121	12	326	180
6	100	7	225	141	14	380	210
7	100	9	289	181	18	488	270
8	100	11	353	221	22	596	330
9	100	13	417	261	26	704	390
10	100	15	481	301	30	812	450
11	100	17	545	341	34	920	510
12	100	21	673	421	42	1136	630
13	100	26	833	521	52	1406	780
14	100	31	993	621	62	1676	930
15	100	36	1151	721	72	1946	1080
16	100	41	1313	821	82	2216	1230
17	100	46	1473	921	92	2486	1380
18	100	51	1633	1021	102	2756	1530
19	100	101	3233	2021	202	5456	3030
20	100	151	4833	3021	302	8156	4530
21	100	201	6433	4021	402	10856	6030

Table 9.2: Second group of experiments, summary of experimental data.

Table 9.3: First group of experiments, $k = 2$: median time over solved instances.

								Optimum cost (2N terms)		Optimum time (5N terms)		Optimum weight (16N terms)		Lexic. Order cost time weight		Lexic. Order weight time cost	
Experiment	Number of Instances	Number of Replicas (N)	Total Number of Nodes	Number of Rational Variables	% Unrealizable	Solving Time	Time for Proving Unrealizable	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout
1	100	2	90	52	1	0.00	0.00	0.00	0	0.01	0	0.02	0	0.01	0	0.02	0
2	100	3	134	78	1	0.00	0.00	0.01	0	0.01	0	0.03	0	0.01	0	0.25	0
3	100	4	178	104	3	0.00	0.00	0.01	0	0.01	0	0.41	0	0.01	0	1.53	0
4	100	5	222	130	3	0.00	0.00	0.02	0	0.03	0	5.51	0	0.02	0	7.48	0
5	100	6	266	156	2	0.00	0.00	0.01	0	0.02	0	24.74	0	0.18	0	29.74	2
6	100	7	310	182	5	0.00	0.00	0.01	0	0.04	0	533.90	19	0.02	0	329.34	30
7	100	9	398	234	7	0.00	0.00	0.02	0	0.09	0	185.23	84	0.02	4	494.33	87
8	100	11	486	286	4	0.00	0.00	0.02	0	0.11	0	—	—	7.29	30	—	—
9	100	13	574	338	7	0.00	0.00	0.05	0	0.30	0	—	—	17.98	83	—	—
10	100	15	662	390	13	0.00	0.00	0.04	0	18.13	0	—	—	—	—	—	—
11	100	17	750	442	15	0.00	0.00	0.04	0	3.11	0	—	—	—	—	—	—
12	100	21	926	546	14	0.00	0.00	0.06	0	58.08	11	—	—	—	—	—	—
13	100	26	1146	676	13	0.00	0.00	0.07	0	600.99	78	—	—	—	—	—	—
14	100	31	1366	806	14	0.00	0.00	0.09	0	—	—	—	—	—	—	—	—
15	100	36	1586	936	19	0.00	0.00	0.11	0	—	—	—	—	—	—	—	—
16	100	41	1806	1066	26	0.00	0.00	0.13	0	—	—	—	—	—	—	—	—
17	100	46	2026	1196	24	0.00	0.00	0.18	0	—	—	—	—	—	—	—	—
18	100	51	2246	1326	32	0.00	0.00	0.20	0	—	—	—	—	—	—	—	—
19	100	101	4446	2626	49	0.00	0.00	0.49	0	—	—	—	—	—	—	—	—
20	100	151	6646	3926	68	0.00	0.00	0.77	0	—	—	—	—	—	—	—	—
21	100	201	8846	5226	71	0.00	0.00	0.93	0	—	—	—	—	—	—	—	—

In the first group of experiments we consider the reduced version of the CGMs (i.e. without nice-to-have requirements) without random binary preference relations. We fix $k = 2, 4, 5, 8$. In each setting, we run experiments on three functionalities:

a. plain realizability check (without objectives),

b. single-objective optimization on `cost`, `workTime`, and `Weight` respectively,

c. lexicographic optimization respectively on ⟨`cost`,`workTime`,`Weight`⟩ and on ⟨`Weight`,`workTime`,`cost`⟩

Figure 9.1 shows the overall median CPU time over the solved instances of the first group of experiments, which are plotted against the total number of nodes of the CGM under test. [2] Figure 9.3-Figure 9.6 show the CPU time over the solved instances for each experiment case.

[2] The choice of using the total number of nodes for the X axis in all our plots aims at providing an eye-catching indication of the actual size of the CGMs under test.

Table 9.4: First group of experiments, $k = 4$: median time over solved instances.

								Optimum cost (2N terms)		Optimum time (5N terms)		Optimum weight (16N terms)		Lexic. Order cost time weight		Lexic. Order weight time cost	
Experiment	Number of Instances	Number of Replicas (N)	Total Number of Nodes	Number of Rational Variables	% Unrealizable	Solving Time	Time for Proving Unrealizable	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout
1	100	2	90	52	2	0.00	0.00	0.00	0	0.01	0	0.02	0	0.03	0	0.04	0
2	100	3	134	78	1	0.00	0.00	0.01	0	0.01	0	0.03	0	0.04	0	0.05	0
3	100	4	178	104	2	0.00	0.00	0.01	0	0.02	0	0.08	0	0.06	0	0.13	0
4	100	5	222	130	4	0.00	0.00	0.01	0	0.01	0	0.41	0	0.08	0	0.79	0
5	100	6	266	156	7	0.00	0.00	0.01	0	0.02	0	1.09	0	0.11	0	2.82	0
6	100	7	310	182	7	0.00	0.00	0.02	0	0.05	0	9.53	4	0.13	0	14.47	4
7	100	9	398	234	7	0.00	0.00	0.02	0	0.06	0	13.54	56	0.64	0	447.13	67
8	100	12	486	286	10	0.00	0.00	0.02	0	0.14	0	—	—	1.53	5	—	—
9	100	13	574	338	9	0.00	0.00	0.03	0	0.43	0	—	—	36.78	23	—	—
10	100	15	662	390	11	0.00	0.00	0.04	0	2.42	0	—	—	368.94	55	—	—
11	100	17	750	442	9	0.00	0.00	0.04	0	28.45	0	—	—	—	—	—	—
12	100	21	926	546	15	0.00	0.00	0.05	0	97.86	2	—	—	—	—	—	—
13	100	26	1146	676	22	0.00	0.00	0.09	0	537.73	62	—	—	—	—	—	—
14	100	31	1366	806	25	0.00	0.00	0.12	0	—	—	—	—	—	—	—	—
15	100	36	1586	936	27	0.00	0.00	0.16	0	—	—	—	—	—	—	—	—
16	100	41	1806	1066	32	0.00	0.00	0.14	0	—	—	—	—	—	—	—	—
17	100	46	2026	1196	36	0.00	0.00	0.17	0	—	—	—	—	—	—	—	—
18	100	51	2246	1326	40	0.00	0.00	0.20	0	—	—	—	—	—	—	—	—
19	100	101	4446	2626	55	0.00	0.00	0.80	0	—	—	—	—	—	—	—	—
20	100	151	6646	3926	77	0.00	0.00	0.72	0	—	—	—	—	—	—	—	—
21	100	201	8846	5226	85	0.00	0.00	1.18	0	—	—	—	—	—	—	—	—

First, we notice that checking the realizability of the CGM, that is, finding one realization or verifying there is none, requires negligible time, even with huge CGMs (>8,000 nodes, >5,000 rational variables) and even when the CGM is not realizable. Second, the time taken to find optimal solutions on single objectives seem to depend more on the number of variables in the objective than on the actual size of the CGM: for `cost` ($2 \cdot N$ variables) the solver can find optimum solutions very quickly even with huge CGMs (>8.000 nodes, >5,000 rational variables) whilst with `Weight` ($16 \cdot N$ variables) it can handle problems of up to ≈ 400 nodes and ≈ 200 rational variables. Third, lexicographic optimization takes more time than single-objective optimization, but the time mostly depends on the first objective in the list.

In the second group of experiments we consider the full version of the CGMs (with nice-to-have requirements) and introduce the random binary preference relations. We fix $k = 2$ and we run different experiments for $p = 6$, $p = 8$ and $p = 12$. In each setting, we

Table 9.5: First group of experiments, $k = 5$: median time over solved instances.

								Optimum cost (2N terms)		Optimum time (5N terms)		Optimum weight (16N terms)		Lexic. Order cost time weight		Lexic. Order weight time cost	
Experiment	Number of Instances	Number of Replicas (N)	Total Number of Nodes	Number of Rational Variables	% Unrealizable	Solving Time	Time for Proving Unrealizable	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout
1	100	2	90	52	4	0.00	0.00	0.01	0	0.01	0	0.02	0	0.03	0	0.03	0
2	100	3	134	78	3	0.00	0.00	0.00	0	0.01	0	0.06	0	0.04	0	0.08	0
3	100	4	178	104	6	0.00	0.00	0.01	0	0.01	0	0.07	0	0.04	0	0.11	0
4	100	5	222	130	6	0.00	0.00	0.01	0	0.02	0	0.56	0	0.07	0	0.67	0
5	100	6	266	156	7	0.00	0.00	0.03	0	0.03	0	1.51	0	0.14	0	1.69	0
6	100	7	310	182	5	0.00	0.00	0.01	0	0.03	0	0.45	0	0.11	0	0.69	0
7	100	9	398	234	7	0.00	0.00	0.02	0	0.27	0	284.79	31	0.71	0	557.00	36
8	100	11	486	286	9	0.00	0.00	0.02	0	0.13	0	852.66	80	0.92	0	705.92	85
9	100	13	574	338	17	0.00	0.00	0.03	0	0.17	0	—	—	47.55	9	—	—
10	100	15	662	390	14	0.00	0.00	0.04	0	1.23	0	—	—	111.58	28	—	—
11	100	17	750	442	13	0.00	0.00	0.05	0	11.87	0	—	—	35.31	56	—	—
12	100	21	926	546	24	0.00	0.00	0.07	0	104.67	0	—	—	—	—	—	—
13	100	26	1146	676	27	0.00	0.00	0.12	0	455.20	51	—	—	—	—	—	—
14	100	31	1366	806	32	0.00	0.00	0.12	0	—	—	—	—	—	—	—	—
15	100	36	1586	936	33	0.00	0.00	0.12	0	—	—	—	—	—	—	—	—
16	100	41	1806	1066	33	0.00	0.00	0.16	0	—	—	—	—	—	—	—	—
17	100	46	2026	1196	53	0.00	0.00	0.16	0	—	—	—	—	—	—	—	—
18	100	51	2246	1326	48	0.00	0.00	0.23	0	—	—	—	—	—	—	—	—
19	100	101	4446	2626	73	0.00	0.00	0.51	0	—	—	—	—	—	—	—	—
20	100	151	6646	3926	76	0.00	0.00	3.33	0	—	—	—	—	—	—	—	—
21	100	201	8846	5226	93	0.00	0.00	1.49	0	—	—	—	—	—	—	—	—

run experiments on three functionalities:

a. plain realizability check (without objectives),

b. lexicographic optimization on ⟨`numUnsatPrefs`,`numUnsatRequirements`,`numSatTasks`⟩ (PRT),

c. lexicographic optimization on ⟨`numUnsatRequirements`,`numUnsatPrefs`,`numSatTasks`⟩ (RPT).

Figure 9.2 shows the overall median CPU time over the solved instances of the second group of experiments.

Figure 9.7-Figure 9.9 show the median CPU time over the solved instances for each experiment case.

Table 9.6: First group of experiments, $k = 8$: median time over solved instances.

								Optimum cost (2N terms)		Optimum time (5N terms)		Optimum weight (16N terms)		Lexic. Order cost time weight		Lexic. Order weight time cost	
Experiment	Number of Instances	Number of Replicas (N)	Total Number of Nodes	Number of Rational Variables	% Unrealizable	Solving Time	Time for Proving Unrealizable	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout	Optimization Time	% Timeout
1	100	2	90	52	10	0.00	0.00	0.01	0	0.01	0	0.03	0	0.02	0	0.04	0
2	100	3	134	78	15	0.00	0.00	0.01	0	0.01	0	0.03	0	0.04	0	0.06	0
3	100	4	178	104	9	0.00	0.00	0.01	0	0.01	0	0.14	0	0.04	0	0.19	0
4	100	5	222	130	11	0.00	0.00	0.01	0	0.02	0	0.07	0	0.06	0	0.09	0
5	100	6	266	156	21	0.00	0.00	0.01	0	0.03	0	1.85	0	0.07	0	2.25	0
6	100	7	310	182	24	0.00	0.00	0.01	0	0.02	0	14.71	0	0.10	0	14.11	0
7	100	9	398	234	33	0.00	0.00	0.01	0	0.11	0	17.37	1	0.15	0	25.14	1
8	100	11	486	286	23	0.00	0.00	0.03	0	0.31	0	79.55	19	0.51	0	253.57	28
9	100	13	574	338	28	0.00	0.00	0.03	0	0.22	0	131.37	55	0.64	0	240.96	59
10	100	15	662	390	36	0.00	0.00	0.04	0	0.41	0	—	—	6.89	0	—	—
11	100	17	750	442	20	0.00	0.00	0.05	0	0.86	0	—	—	0.56	1	—	—
12	100	21	926	546	48	0.00	0.00	0.05	0	149.86	7	—	—	104.81	17	—	—
13	100	26	1146	676	43	0.00	0.00	0.06	0	406.31	23	—	—	—	—	—	—
14	100	31	1366	806	61	0.00	0.00	0.10	0	—	—	—	—	—	—	—	—
15	100	36	1586	936	67	0.00	0.00	0.23	0	—	—	—	—	—	—	—	—
16	100	41	1806	1066	71	0.00	0.00	0.39	0	—	—	—	—	—	—	—	—
17	100	46	2026	1196	77	0.00	0.00	0.17	0	—	—	—	—	—	—	—	—
18	100	51	2246	1326	75	0.00	0.00	0.17	0	—	—	—	—	—	—	—	—
19	100	101	4446	2626	98	0.00	0.00	1.47	0	—	—	—	—	—	—	—	—
20	100	151	6646	3926	97	0.00	0.00	40.11	0	—	—	—	—	—	—	—	—
21	100	201	8846	5226	100	0.00	0.00	—	—	—	—	—	—	—	—	—	—

First, checking realizability is accomplished in negligible time even with huge CGMs (>10,000 nodes, >6,000 rational variables), as before. Second, we notice that optimal solutions, even with a three-level lexicographic combination of objectives, can be found with large CGMs (>1,000 nodes, >600 rational variables).

On the negative side, for some problems, in particular large ones with objectives involving large amounts of elements, we notice that the search for the optimal realization could not be accomplished within the timeout.

To this extent, a few remarks are in order.

First, when interrupted by a timeout, OptiMathSAT can be instructed to return the current best solution. Since OptiMathSAT typically takes most of its time in fine-tuning the optimum and in checking there is no better one (see [ST15a]), we envisage that good sub-optimal solutions can be found even when optimal ones are out of reach.

Second, our CGMs are very large in breadth and small in depth, with a dominating

Table 9.7: Second group of experiments, $k = 2$, $p = 6$: median time over solved instances.

								Lexic. Order PRT		Lexic. Order RPT	
Experiment	Number of Instances.	Number of Replicas (N)	Total Number of Nodes	Number of Rational Variables	% Unrealizable	Solving Time	Time for Proving Unrealizable	Optimization Time	% Timeout	Optimization Time	% Timeout
1	100	2	110	60	1	0.00	0.00	0.04	0	0.08	0
2	100	3	164	90	2	0.00	0.00	0.07	0	0.08	0
3	100	4	218	120	1	0.00	0.00	0.11	0	0.09	0
4	100	5	272	150	3	0.00	0.00	0.12	0	0.15	0
5	100	6	326	180	2	0.00	0.00	0.13	0	0.20	0
6	100	7	380	210	3	0.00	0.00	0.21	0	0.26	0
7	100	9	488	270	2	0.00	0.00	0.52	0	0.45	0
8	100	11	596	330	7	0.00	0.00	0.90	0	0.50	0
9	100	13	704	390	8	0.00	0.00	2.42	0	2.33	0
10	100	15	812	450	4	0.00	0.00	39.45	0	1.55	0
11	100	17	920	510	6	0.00	0.00	1.64	0	1.57	0
12	100	21	1136	630	7	0.00	0.00	694.50	52	468.88	20
13	100	26	1406	780	6	0.00	0.00	—	—	—	—
14	100	31	1676	930	14	0.00	0.00	—	—	—	—
15	100	36	1946	1080	15	0.00	0.00	—	—	—	—
16	100	41	2216	1230	19	0.00	0.00	—	—	—	—
17	100	46	2486	1380	16	0.00	0.00	—	—	—	—
18	100	51	2756	1530	27	0.00	0.00	—	—	—	—
19	100	101	5456	3030	33	0.00	0.00	—	—	—	—
20	100	151	8156	4530	46	0.00	0.00	—	—	—	—
21	100	201	10856	6030	56	0.00	0.00	—	—	—	—

Table 9.8: Second group of experiments, $k = 2$, $p = 8$: median time over solved instances.

								Lexic. Order PRT		Lexic. Order RPT	
Experiment	Number of Instances.	Number of Replicas (N)	Total Number of Nodes	Number of Rational Variables	% Unrealizable	Solving Time	Time for Proving Unrealizable	Optimization Time	% Timeout	Optimization Time	% Timeout
1	100	2	110	60	0	0.00	0.00	0.06	0	0.07	0
2	100	3	164	90	1	0.00	0.00	0.08	0	0.08	0
3	100	4	218	120	0	0.00	0.00	0.18	0	0.09	0
4	100	5	272	150	2	0.00	0.00	0.18	0	0.14	0
5	100	6	326	180	1	0.00	0.00	0.36	0	0.18	0
6	100	7	380	210	2	0.00	0.00	0.21	0	0.20	0
7	100	9	488	270	6	0.00	0.00	0.28	0	0.30	0
8	100	11	596	330	4	0.00	0.00	0.61	0	0.47	0
9	100	13	704	390	6	0.00	0.00	0.73	0	0.53	0
10	100	15	812	450	12	0.00	0.00	1.38	0	0.69	0
11	100	17	920	510	6	0.00	0.00	1.81	0	0.99	0
12	100	21	1136	630	10	0.00	0.00	7.00	0	3.92	0
13	100	26	1406	780	11	0.00	0.00	330.39	10	9.38	1
14	100	31	1676	930	11	0.00	0.00	327.86	72	8.40	10
15	100	36	1946	1080	14	0.00	0.00	—	—	—	—
16	100	41	2216	1230	13	0.00	0.00	—	—	—	—
17	100	46	2486	1380	14	0.00	0.00	—	—	—	—
18	100	51	2756	1530	20	0.00	0.00	—	—	—	—
19	100	101	5456	3030	33	0.00	0.00	—	—	—	—
20	100	151	8156	4530	40	0.00	0.00	—	—	—	—
21	100	201	10856	6030	59	0.00	0.00	—	—	—	—

Table 9.9: Second group of experiments, $k = 2$, $p = 12$: median time over solved instances.

								Lexic. Order PRT		Lexic. Order RPT	
Experiment	Number of Instances.	Number of Replicas (N)	Total Number of Nodes	Number of Rational Variables	% Unrealizable	Solving Time	Time for Proving Unrealizable	Optimization Time	% Timeout	Optimization Time	% Timeout
1	100	2	110	60	1	0.00	0.00	0.06	0	0.06	0
2	100	3	164	90	0	0.00	0.00	0.09	0	0.07	0
3	100	4	218	120	0	0.00	0.00	0.13	0	0.12	0
4	100	5	272	150	0	0.00	0.00	0.15	0	0.17	0
5	100	6	326	180	0	0.00	0.00	0.25	0	0.20	0
6	100	7	380	210	0	0.00	0.00	0.39	0	0.30	0
7	100	9	488	270	0	0.00	0.00	0.43	0	0.49	0
8	100	11	596	330	0	0.00	0.00	0.81	0	0.56	0
9	100	13	704	390	1	0.00	0.00	1.15	0	0.89	0
10	100	15	812	450	1	0.00	0.00	1.32	0	0.37	0
11	100	17	920	510	2	0.00	0.00	14.66	0	1.97	0
12	100	21	1136	630	0	0.00	0.00	602.22	23	2.13	0
13	100	26	1406	780	2	0.00	0.00	911.26	87	905.11	9
14	100	31	1676	930	4	0.00	0.00	—	—	14.79	24
15	100	36	1946	1080	0	0.00	0.00	—	—	—	—
16	100	41	2216	1230	1	0.00	0.00	—	—	—	—
17	100	46	2486	1380	2	0.00	0.00	—	—	—	—
18	100	51	2756	1530	1	0.00	0.00	—	—	—	—
19	100	101	5456	3030	5	0.00	0.00	—	—	—	—
20	100	151	8156	4530	5	0.00	0.00	—	—	—	—
21	100	201	10856	6030	10	0.00	0.00	—	—	—	—

percentage of tasks over the total number of goals. We envisage that this may have made the number of variables in the sums defining `Weight` and `numSatTasks` unrealistically large wrt. the total size of the CGMs. This underscores the need for further experimentation to confirm the scalability of our proposal.

Third, in our experiments we did not consider user assertions which, if considered, would force deterministic assignments and hence reduce drastically the size of the OMT search space.

Fourth, OMT is a recent technology [ST12] which is progressing at a very high pace, so that it is reasonable to expect further performance improvements for the future versions of OMT tools. In particular, a recent enhancement for handling Pseudo-Boolean cost functions as in (5.2) has provided interesting preliminary results [ST17].

Overall, our evaluation showed that CGM-Tool always checks the realizability of huge CGMs in negligible time and finds optimal realizations on problems whose size ranges from few hundreds to thousands of nodes, mostly depending on the number of variables involved in the objective functions.

Figure 9.10-Figure 9.14 show the median CPU time comparison, while Figure 9.15 shows the unsatisfiable percentage of the experiments.

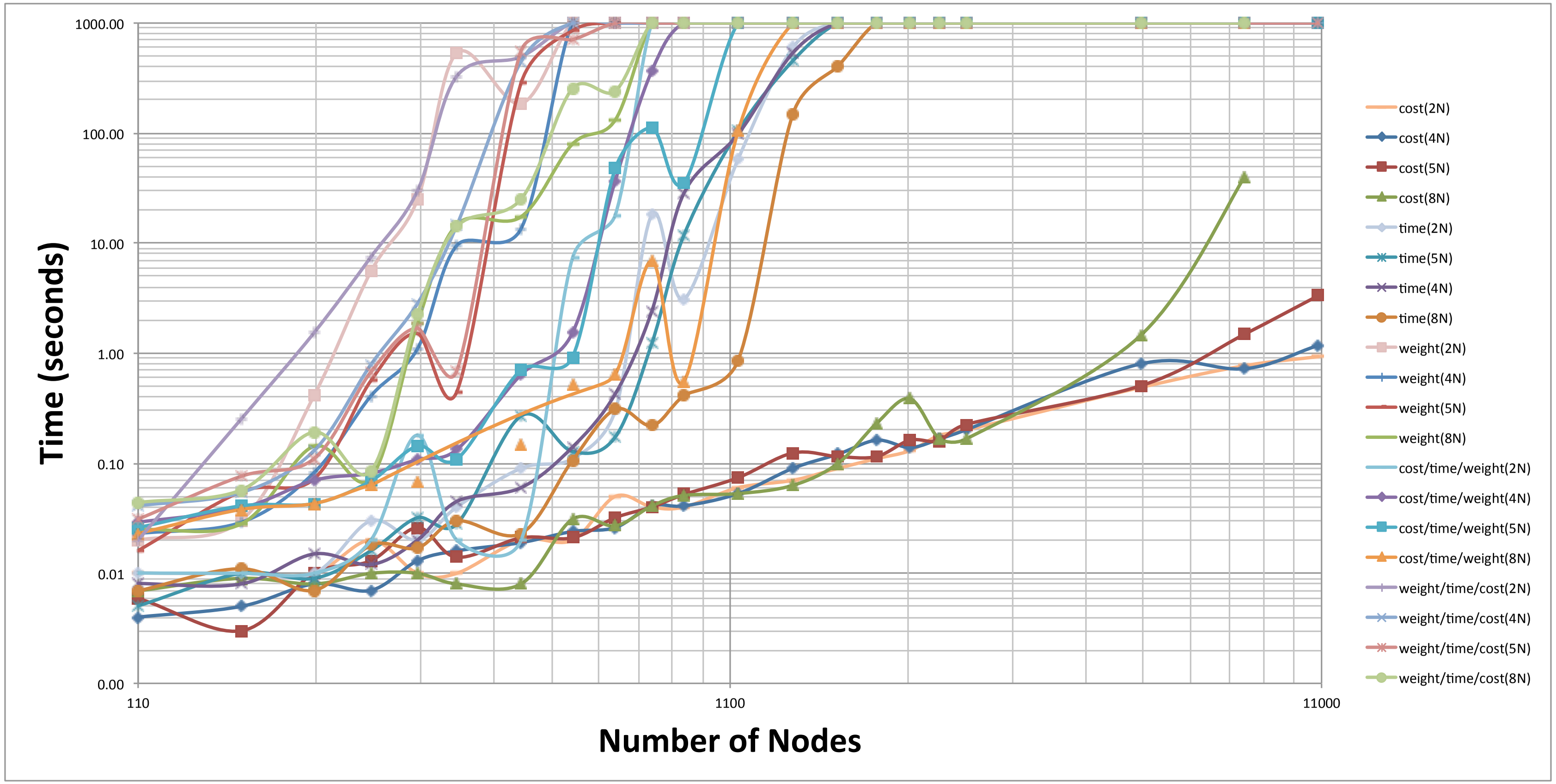


Figure 9.1: The name of each plot denotes the cost function used and the value of N: e.g., `cost/time/weight(2N)` detotes the lexicographic optimization of $\{\texttt{cost}, \texttt{WorkTime}, \texttt{Penalty} - \texttt{Reward}\}$ on problems built on $N = 2$ replicas.

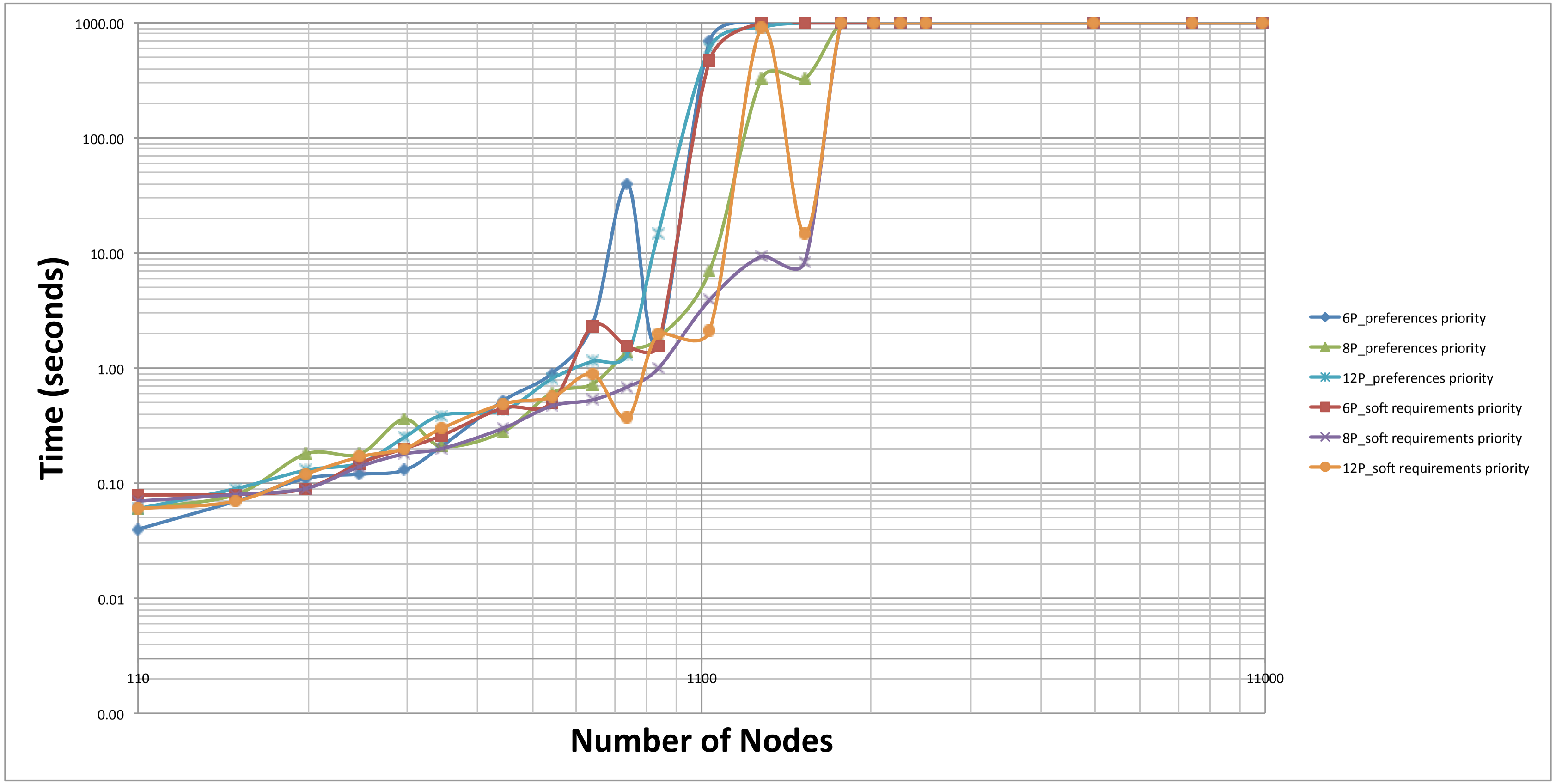


Figure 9.2: Second group of experiments: overall median CPU times over solved instances.

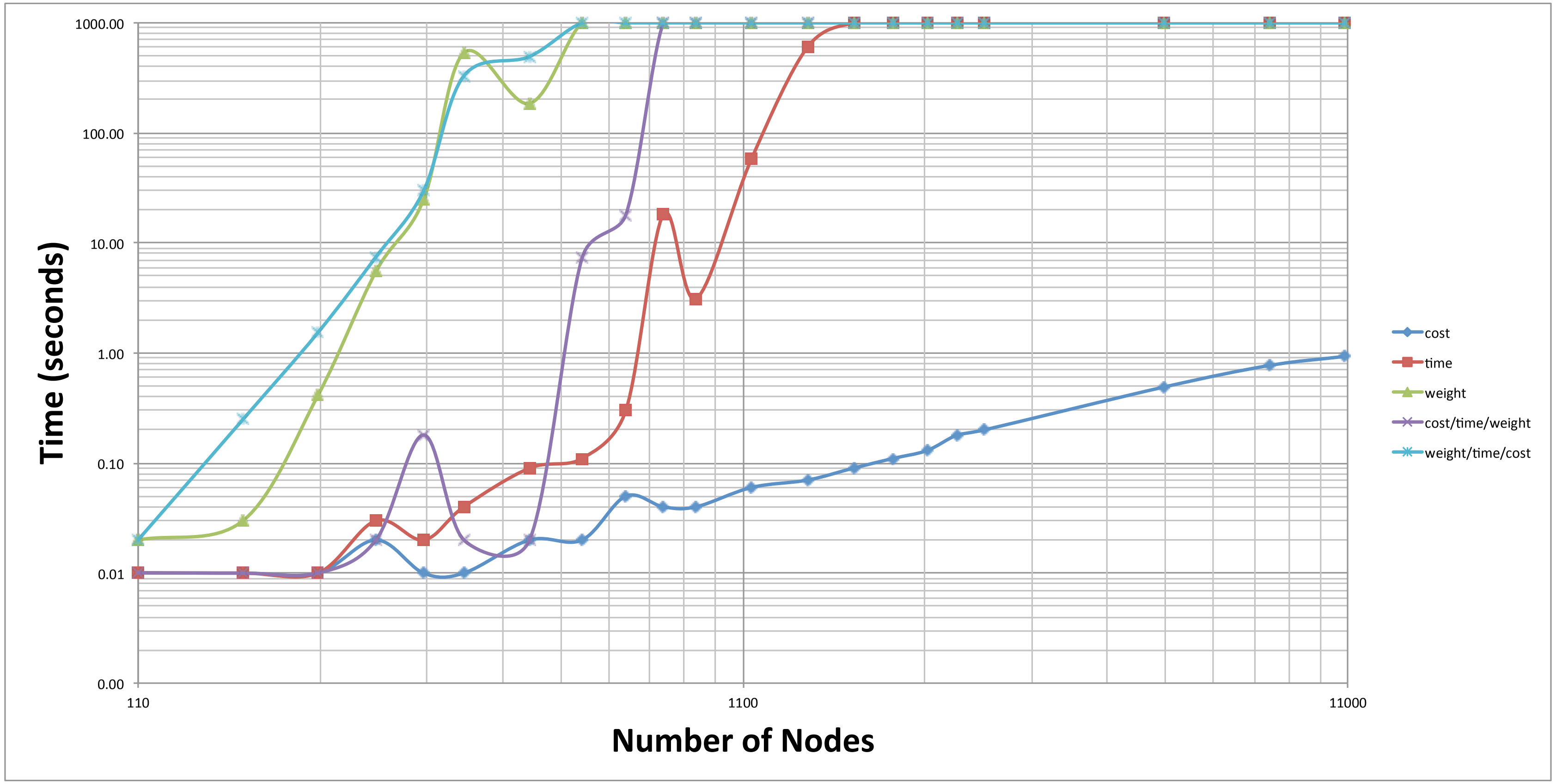


Figure 9.3: First group of experiments, $k = 2$, median run times over solved instances.

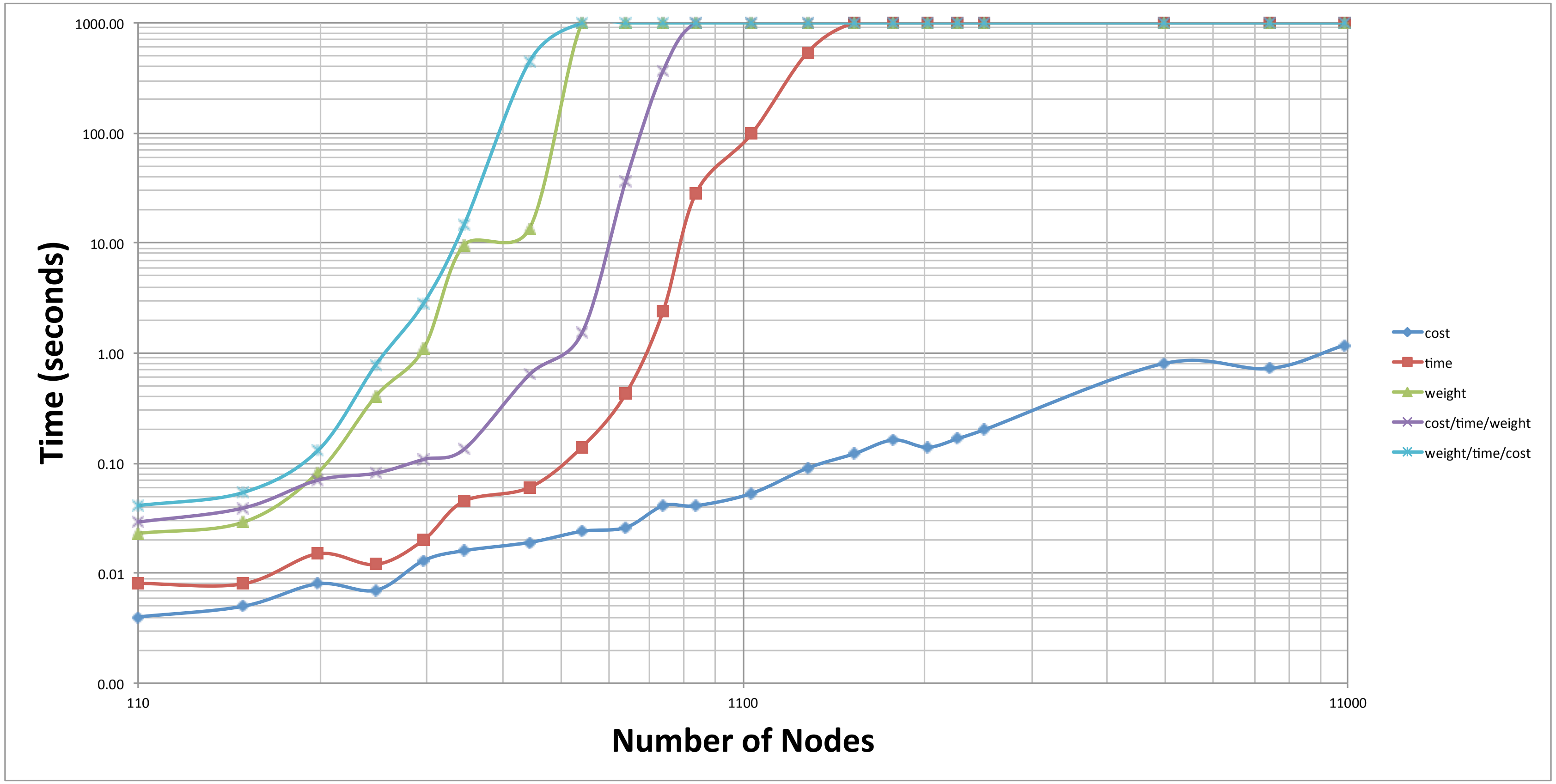


Figure 9.4: First group of experiments, $k = 4$, median run times over solved instances.

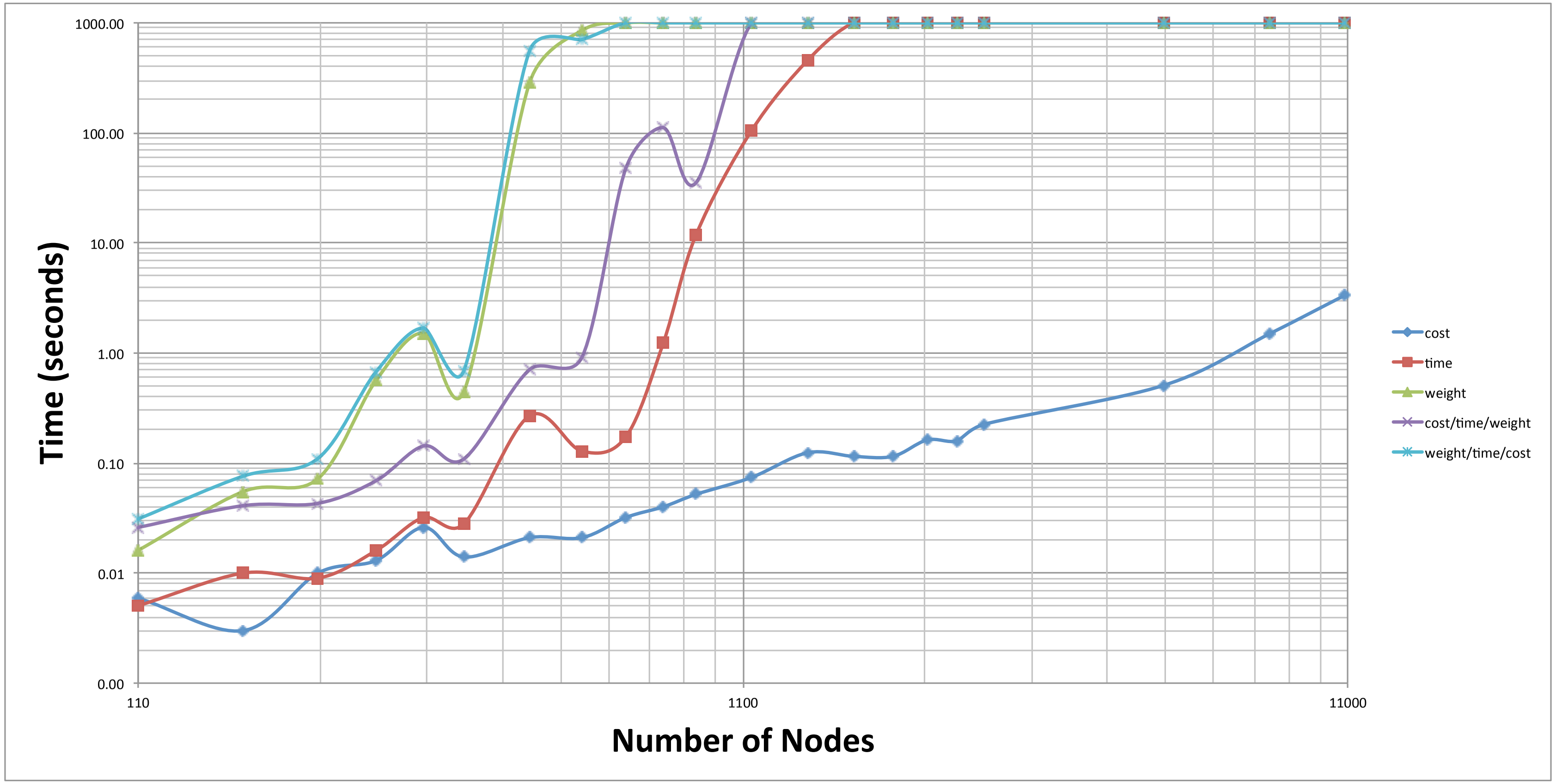


Figure 9.5: First group of experiments, $k = 5$, median run times over solved instances.

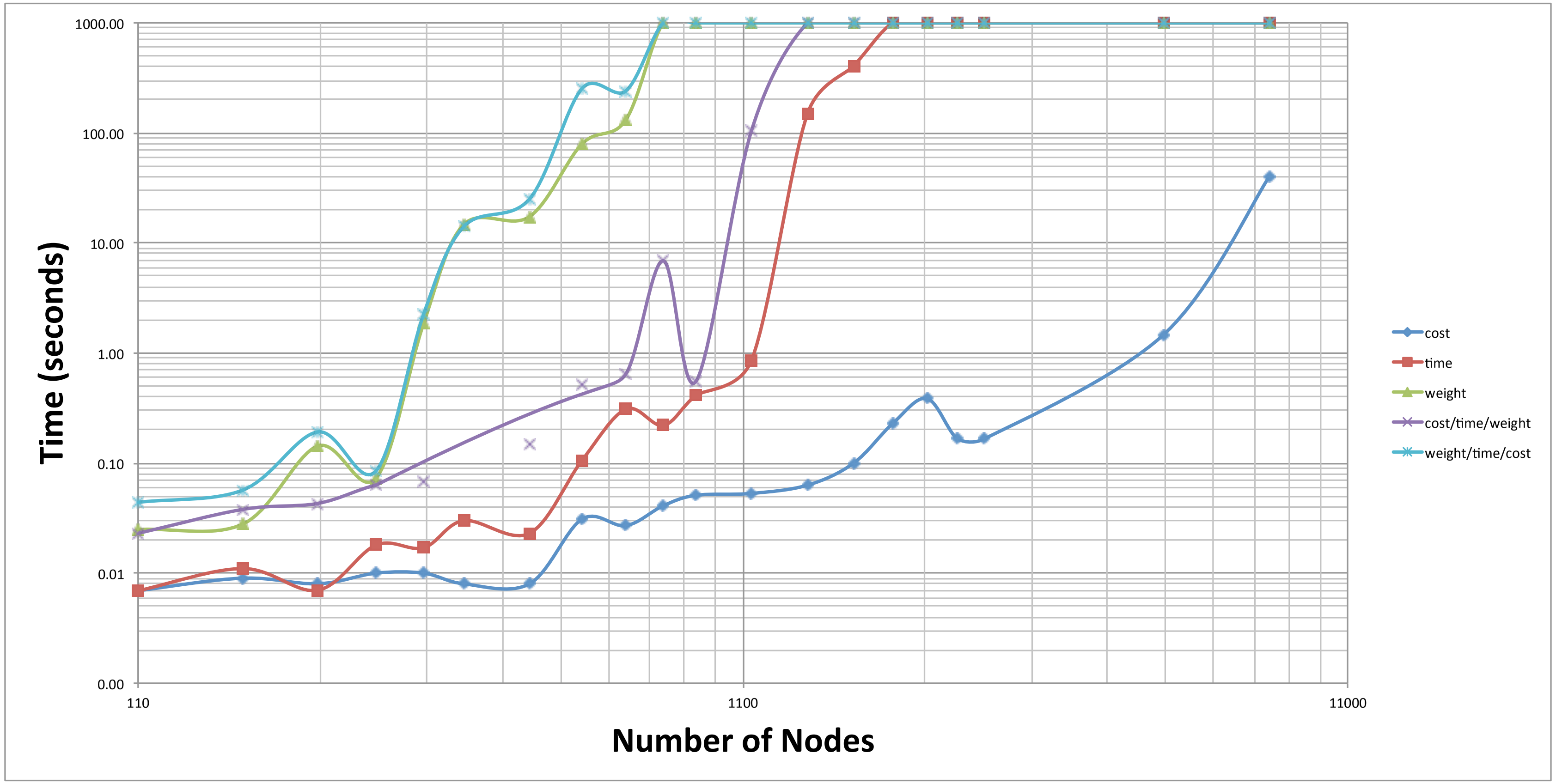


Figure 9.6: First group of experiments, $k = 8$, median run times over solved instances.

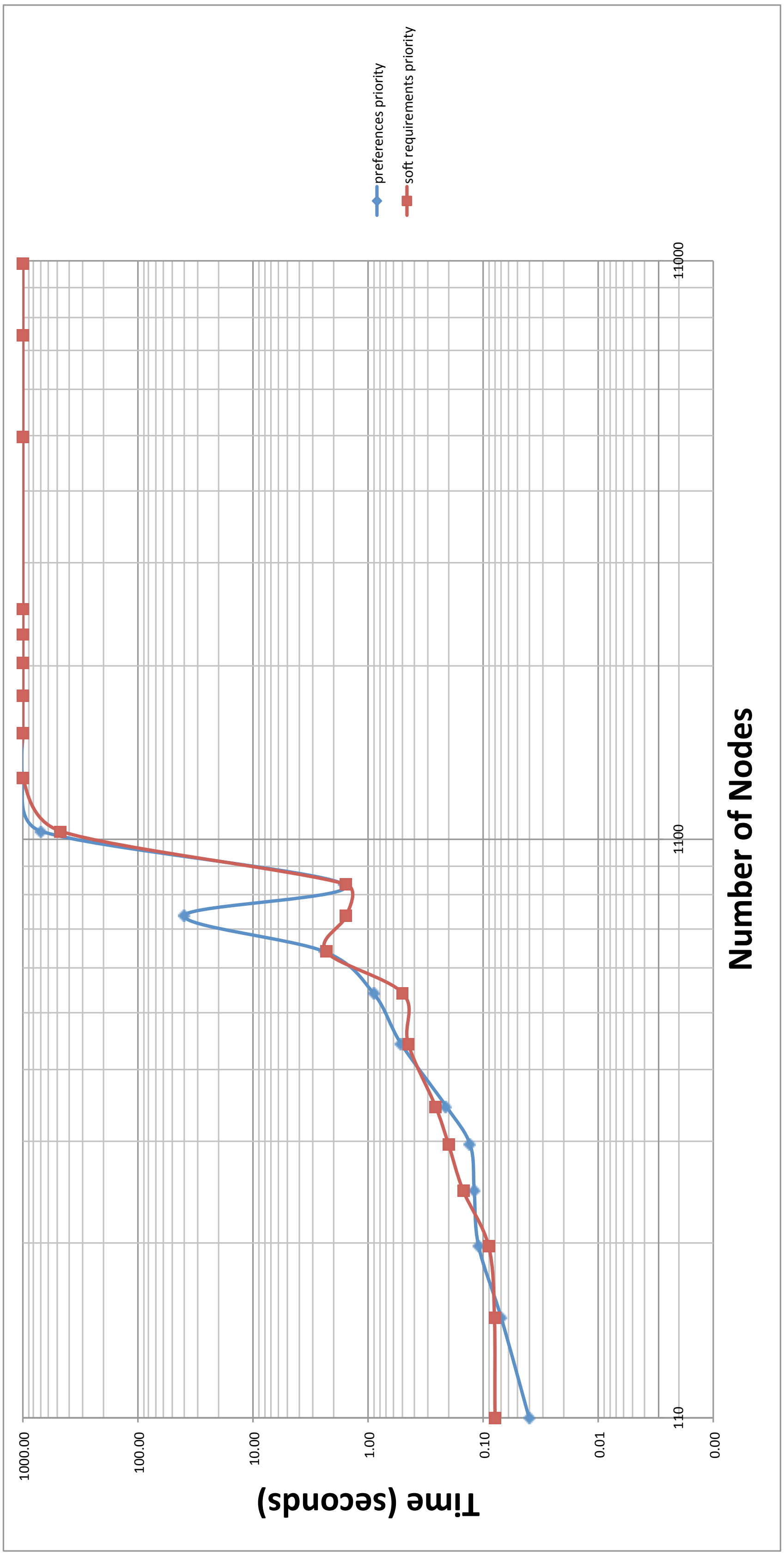


Figure 9.7: Second group of experiments, $k = 2$, $p = 6$, median run times over solved instances.

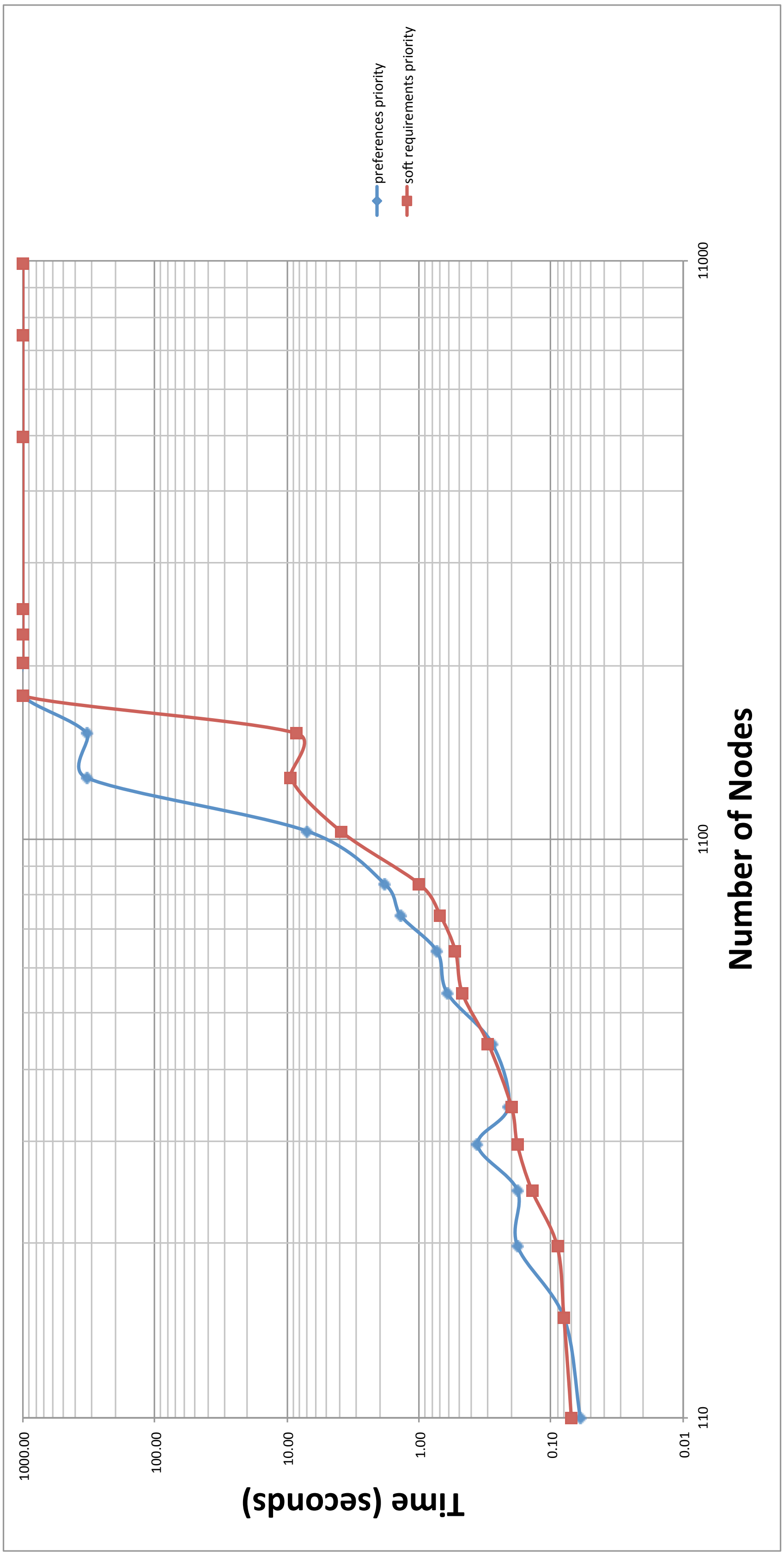


Figure 9.8: Second group of experiments, $k = 2$, $p = 8$, median run times over solved instances.

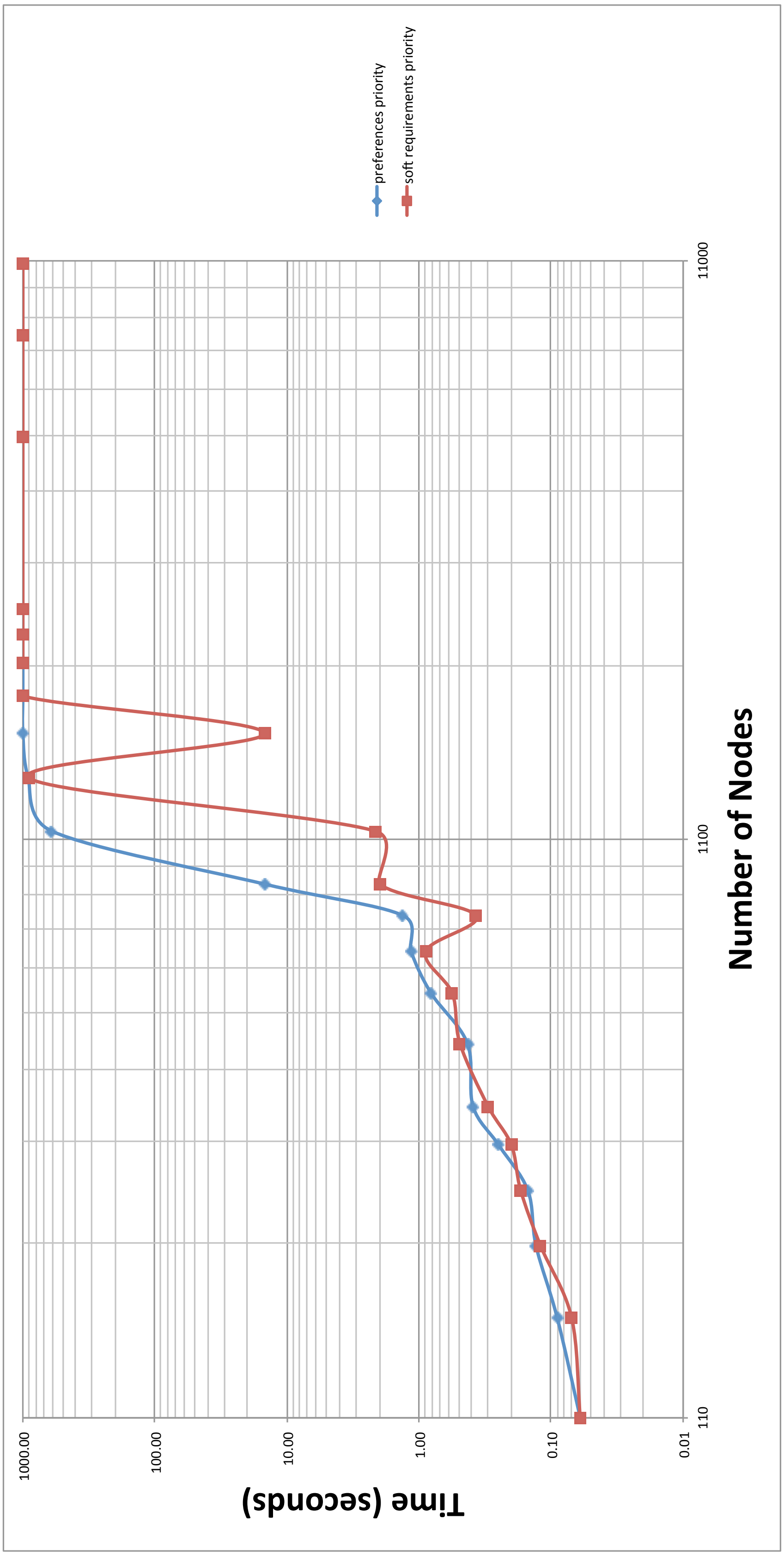


Figure 9.9: Second group of experiments, $k = 2$, $p = 12$, median run times over solved instances.

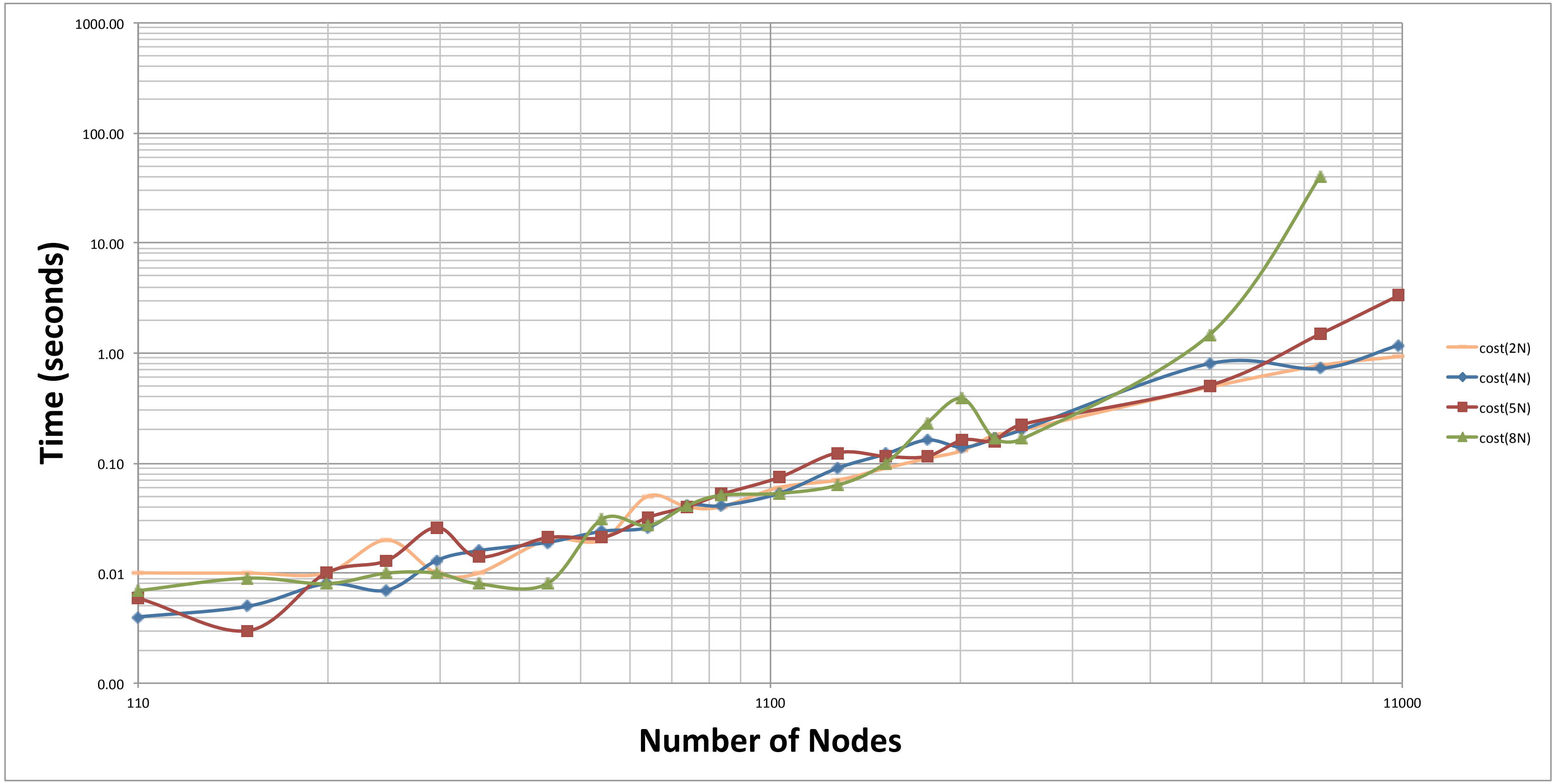


Figure 9.10: Experimental Median Runtime Comparison: Cost Optimization

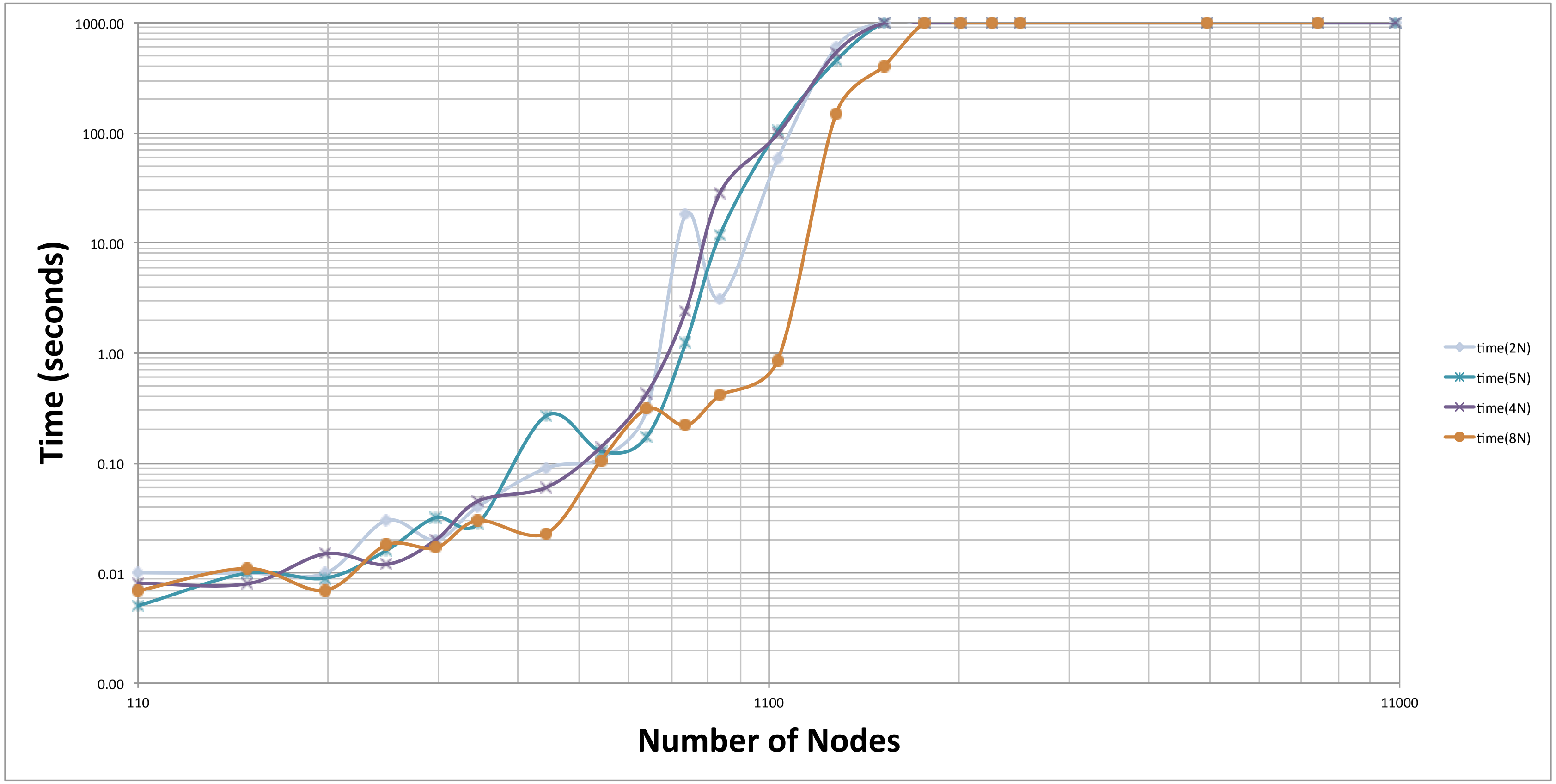


Figure 9.11: Experimental Median Runtime Comparison: Time Optimization

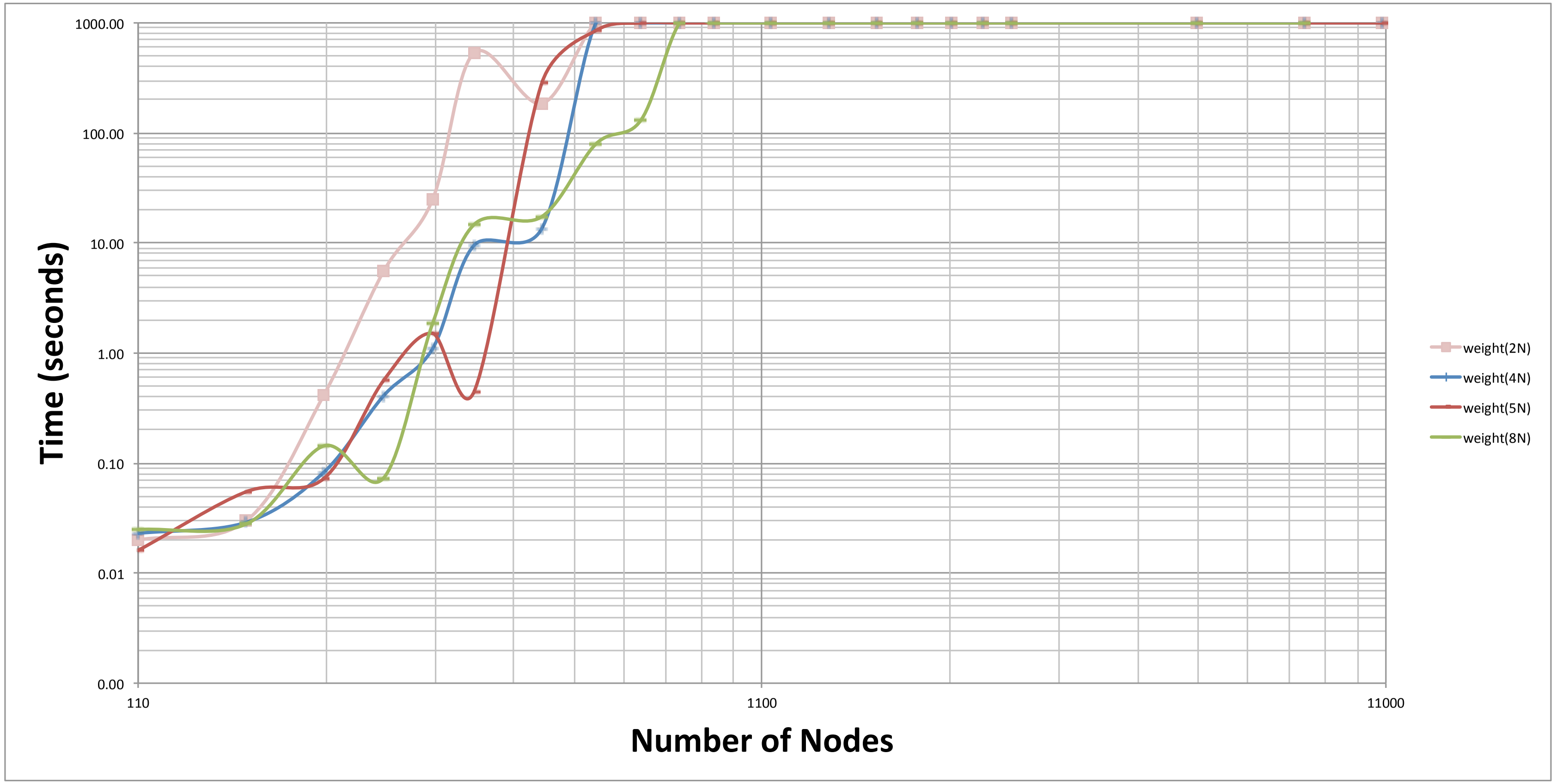


Figure 9.12: Experimental Median Runtime Comparison: Weight Optimization

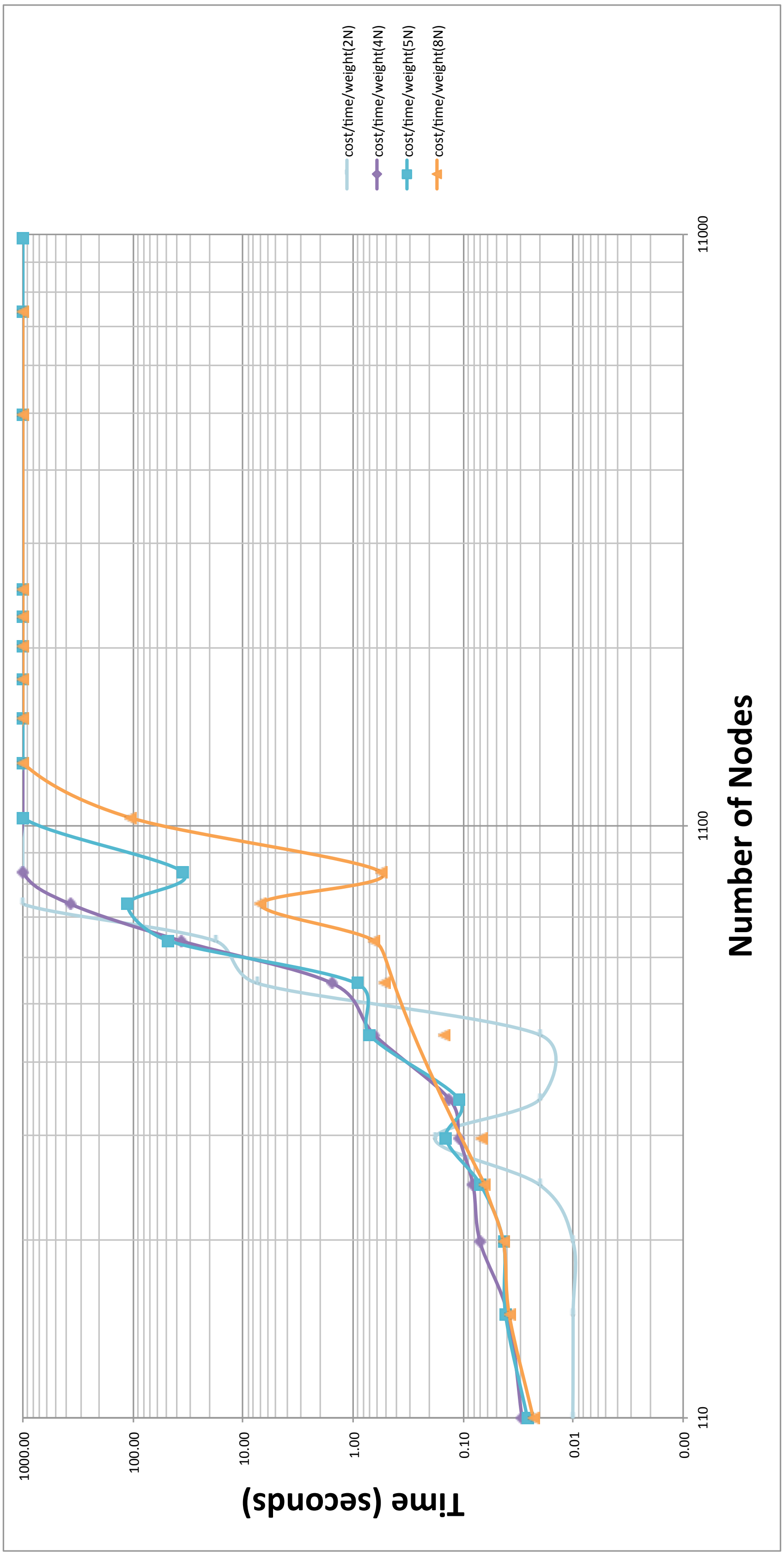


Figure 9.13: Experimental Median Runtime Comparison: Cost/Time/Weight Lex-Order Optimization

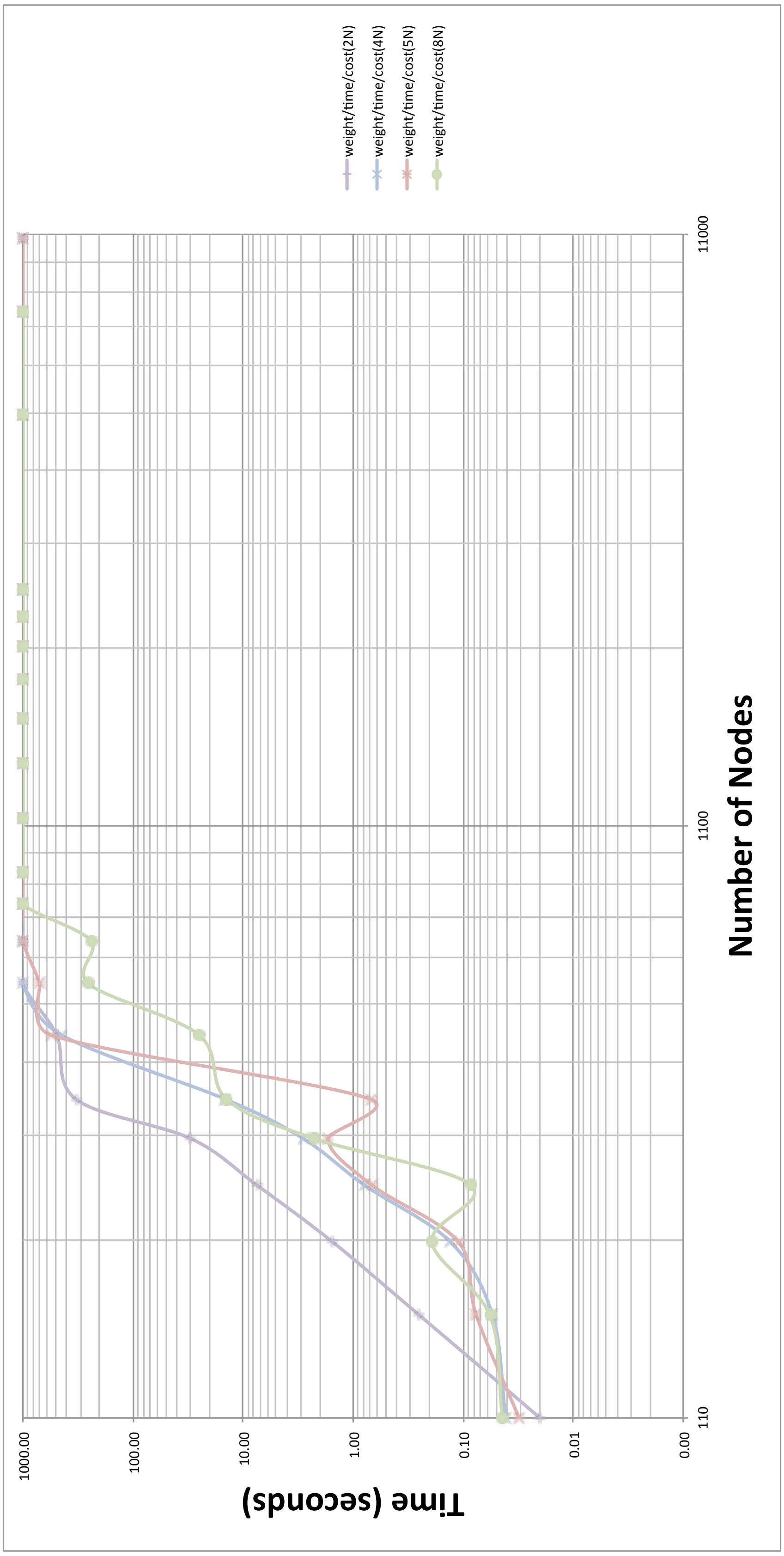


Figure 9.14: Experimental Median Runtime Comparison: Weight/Time/Cost Lex-Order Optimization

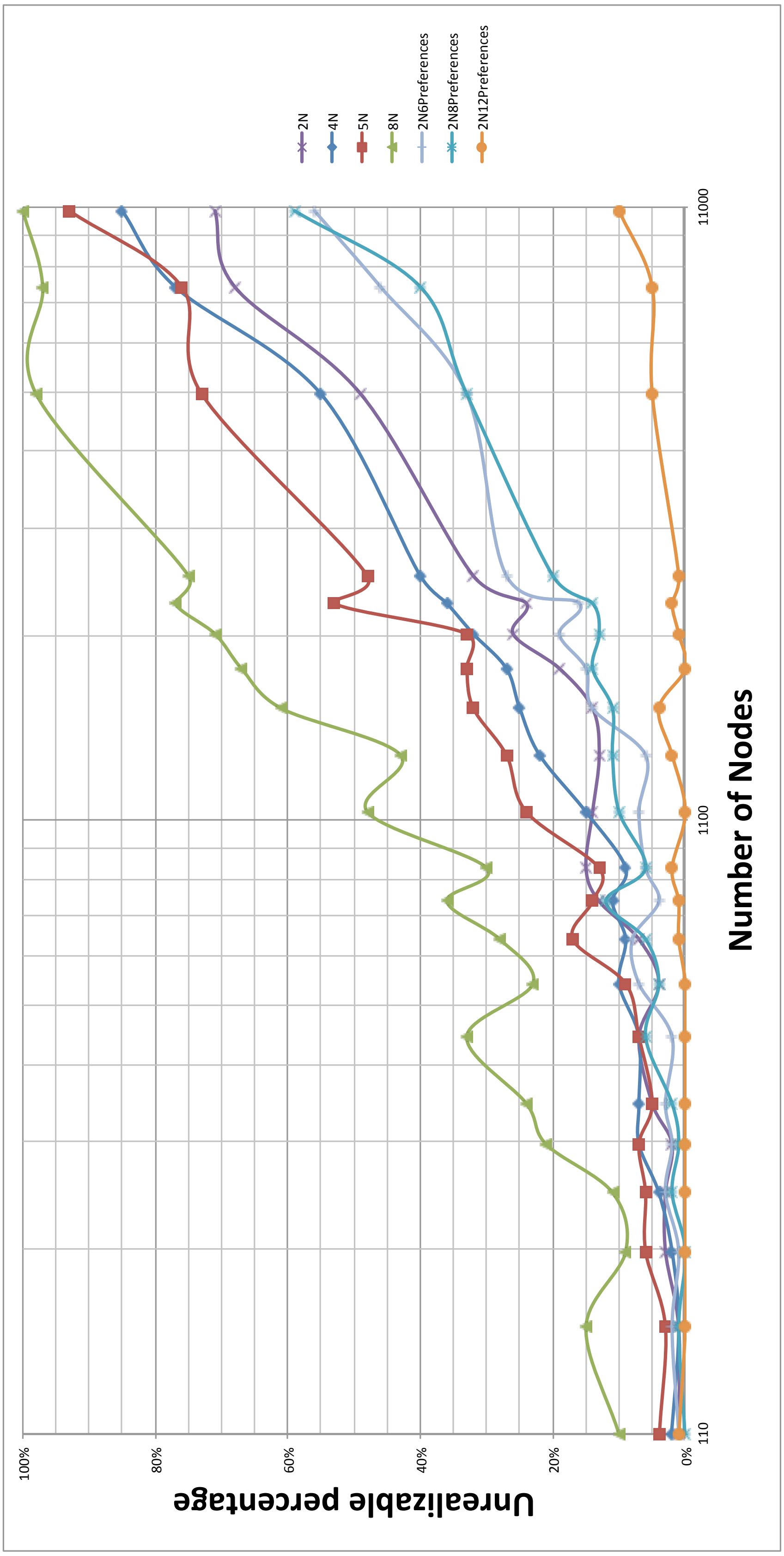


Figure 9.15: Percentage of unrealizable instances, both groups of experiments.

CHAPTER 10

USER-ORIENTED EVALUATION

This chapter reports the evaluation process and results of our approach with the end-users (modellers).

* * ★ * *

As stated in chapter 1 our project aim is to advancing the state-of-the-art in goal models and reasoning by proposing a more expressive modelling language that encompasses many of the modelling constructs proposed in the literature, and at the same time offers sound, complete, and tractable reasoning facilities. Hence, we have conducted a user-oriented experiment and evaluation with the end-users (modellers). The evaluation aimed to assess the *usability* of the CGM modelling language and CGM-tool for the modellers.

10.1 Evaluation Objectives

As mentioned above, the goal of the evaluation is to assess the *usability* of the CGM framework. However, the definitions and measurement models of usability is quite varied. The *International Organization for Standardization (ISO)* measures usability based on the understanding, learnability, attractiveness of the software product for the users when used under specification conditions [ISO06]. [ISO98] measures usability based on the effectiveness in achieving specified goal for the users. The *glsieee* describes usability as the ease a user can learn how to operate, prepare inputs for, understand and interpret the outputs of a system or component [IEE90]. In [DR93], usability is defined by the quickness and simplicity of a user task accomplishment. This definition based on four assumptions: usability means focusing on users, usability includes productivity, usability means ease of use, and) usability means efficient task accomplishment. [Sha91] defines usability based on five attributes: speed, time to learn, retention, errors, and user specific attitude. [PRS$^+$94] measures usability based on effectiveness and efficiency to throughput. While [CL99, Nie93] defines usability by learnability, memorability, effectiveness, efficiency, and user satisfaction. Besides the mentioned above, there are many more definition and measurement models of usability. The variety of definitions and measurement of usability lead to inconsistency across the literature due to the use of different terms for the same or similar characteristic. Therefore, this complicates the extraction of attributes to measure the usability of modelling framework.

In our evaluation, we adopted the usability definition for modelling framework presented in [SCR11]. The usability, therefore, is specified by "learnability, memorability, effectiveness, efficiency, user satisfaction and perceptibility". Their framework, however, has a different focus than our evaluation objectives. [SCR11] goes for the usability of modelling languages for model interpretation, while we are more interested in model development scenarios and modellers. As stated in [SCR11], for a model development, "a modeller needs

(i) to learn the modelling language,

(i) to remember the language's elements and syntax to ensure correctness of the model,

(i) to reach a fast and correct task accomplishment, and

(i) to be satisfied with the modelling language".

In this evaluation, we focus on three main criteria:

E-C$_1$ What is the *learnability* of the approach? (point (i) and (ii))

E-C$_2$ What is the *effectiveness* and *efficiency* of the approach? (point (iii))

E-C$_3$ What is the *user satisfaction* of the approach in modelling requirements and requirements evolution? (point (iv))

We measure the *learnability* by the ease of learning the CGM for first time user and the memorability, i.e., the proficiency of a user after a period of non-use. The *effectiveness* is measured by the completeness, correctness of the produced CGM model and the *efficiency* is measured by the time taken to model a scenario with CGM. As there is no standardized method for measuring *user satisfaction*, we evaluated the satisfaction through general impression and willingness to adopt the CGM framework. Therefore, to evaluate the proposed criteria, we need to answer the following research questions:

E-RQ$_1$ How long does it take to learn using the CGM modelling language and CGM-Tool? (learnability)

E-RQ$_2$ How effective is the CGM modelling language in capturing requirements problem? (effectiveness)

E-RQ$_3$ How easy to use the CGM modelling language and CGM-Tool? (efficiency)

E-RQ$_4$ What is the willingness of the end-users in adopting CGM? (user satisfaction)

10.2 Experiment Design

There are many usability evaluation methods such as cognitive modelling methods, inspection methods, inquiry methods, prototyping methods, testing methods, etc.. Some use data from users, others rely on usability experts. Usability evaluation methods also vary when using in different stages of design and development. Each method has their own advantages and disadvantages. For example, *think aloud protocol* proposed in [LR94] is frequently used in designing, coding, testing and releasing phases of a software/application. It is a low-cost evaluation method and the results are accurate to user experience, however, the experiment environment is not natural to the participants. *remote usability testing* introduced in [Nie93] is usually used in the same phases as *think aloud protocol*. This method can cover three usability attributes: efficiency, effectiveness, and satisfaction. However, additional software is necessary to observe the participants from distance, which can be costly. Both *Cognitive walkthrough* [WRLP94] and *pluralistic walkthrough* [Bia94] are using inspection evaluation methods. The methods may not require a fully functional prototype, and usability issues are resolved faster. However, both methods do not address the issue of efficiency aspect of the usability, and they are more appropriate for the designing phase.

Given the consideration of cost, time constraints, and availability of resources, we had chosen an inquiry evaluation method: evaluating through questionnaires/surveys. Although the data collected is subjective, it provides valuable information on the user expectation. The evaluation process consists three main phases:

Training:
- Participants attend two 90-minute lectures introducing the CGM modelling language and given hand-on experience with the CGM-tool.
- Participants are given a training material consisting of slides used for the introduction of the CGM modelling language and the CGM-tool manual.

Application:
- Participants work alone or in groups and applying the CGM-modelling approach to the Scheduling Meeting scenario.
- At the end of the application phase, participants have to deliver a report documenting the application of the method.

Evaluation:
- Participants are requested to evaluate the modelling approach through answering questionnaires.

The training phase is needed to access the learnability of the framework (**E-C**$_1$), while the application phase is used to measure the efficiency and effectiveness of the framework (**E-C**$_2$ and **E-C**$_3$). Evaluation phase helps to collect data for measuring all the evaluation criteria (learnability, efficiency/effectiveness, and user satisfaction).

10.3 Experiment Procedure

In accordance with the objectives of the evaluation, we recruited two group of participants:

1. The M.Sc. students attend the M.Sc. courses in *Requirements Engineering and Organisational Information Systems* at the University of Trento. Most of the participants had already some experience in requirements modelling and goal-orientation, however, they were all method ignorant, i.e., none of them had previous knowledge of CGM modelling language or CGM-Tool.

2. The Ph.D. students, who research subjects involve requirements engineering and modelling, in the *Department of Information Engineering and Computer Science* at the University of Trento. They can be considered as expert modellers, who are

very familiar with goal-oriented requirements modelling and have knowledge of our approach baseline: the qualitative goal model introduced in [SGM04].

The main reason for choosing this two groups of participants is because of the **E-C$_1$**: learnability. We want to access the learnability of the CGM framework on complete novices, the M. Sc. students, who have some background in requirements engineering and modelling and may possess some knowledge of goal-orientation modelling, as well as the "expert" modellers, the Ph.D. students, who have strong background in requirements engineering and modelling and possess solid knowledge of goal-oriented requirements modelling. Moreover, we choose university students as participants over industrial workers/researchers not only because of the limitation of participants selection but also because of the open-mindedness and motivation of the students when being introduced to a new approach. Industrial workers/researchers, sometimes, set a limitation for themselves and refuse to try a new approach if it is different from what they have been using. Furthermore, we believe that users' opinions collecting from university students would be less bias than industrial workers/researchers.

Notice that: to avoid learning effects, and domain knowledge drawback, all participants were asked to use the Scheduling Meeting scenario as presented in chapter 4 which is a very common and familiar scenario of requirements engineer.

10.3.1 Study with Master Students

Participants. Eight students enrolled in the Master's course *Requirements Engineering and Organisational Information Systems* at the University of Trento have participated in the evaluation. They had a background in Security Engineering and Information Systems.

Setting. The participants were introduced to the CGM modelling language and they were trained to use the CGM-Tool. At the end of the training, the participants were asked to fill in a questionnaire which assessed their gained knowledge, their understanding of the approach, and their estimation on the quality of the training. During the application phase, the participants were introduced to the Scheduling Meeting scenario and were given supporting material: slides and tutorials on CGM modelling language and CGM-Tool, and a cheat sheet summarising all concepts, relations, and formula form of the CGM-Tool. The participants worked alone or in a group of maximum three students. After examining the scenario, they first drafted the goal model using i* goal model, then built the model using CGM modelling language, and used CGM-tool to build and optimize the model. The method designer (lecture) were present and observe the whole application phase. The lecturer took notes of the questions raised by the participants, their behaviour (discussions, notes, use of training material, etc.), and the behaviour of the method designers (reaction to questions, answering individually or towards all present participants, and other general impressions). Lecturer also answered to general questions on the assigned task and on the features of the tool, and the responses were given to all the students (not only to the student who made the question). No solution to the raised problems or doubts was provided. After the modelling session students were asked to fill in a questionnaire focused on their overall impression of CGM modelling language and CGM-Tool respectively.

10.3.2 Study with Doctoral Students

Participants. Five Doctoral Students in the *Department of Information Engineering and Computer Science* at the University of Trento participated in the evaluation. They all have background in requirements engineering and have knowledge of the CGM approach baseline: the qualitative goal model introduced in [SGM04].,

Setting. The participants were introduced to the CGM modelling language and given hands-on experienced the CGM-Tool. To make a comparison with the other group, the participants were also asked to fill in the same questionnaire (which assessed their knowledge/understanding of the CGM) as the M.Sc. student. During the application phase, the participants were introduced to the Scheduling Meeting scenario, and were given supporting material: slides and tutorials on CGM modelling language and CGM-Tool, and a cheat sheet summarising all concepts, relations, and formula form of the CGM-Tool. The participants worked alone. After examining the scenario, they were asked to build the goal model using both i* goal model and CGM modelling language, then they were asked to use the CGM-tool to build and optimize the model. The participants were not observed during the application phase. Thus, to collect data about the application phase, students were asked to deliver a report describing in details the application of the approach and the generated models. Moreover, they were also asked to were asked to fill in the same questionnaire provided for the M.Sc. students at the end of their application phase.

10.4 Evaluation Result

As mentioned above, the experiment is evaluated using the collected questionnaires filled by the students. There are two questionnaires, one for modelling language, and another for the tool. In addition to the questionnaire, the experiment is also evaluated through the observation of Master Students while doing the application phase as well as the reports of the Ph.D. Students on the application phase.

10.4.1 Training phase

In the questionnaire of the training phase, an example scenario was presented and the student was also presented with the goal graph of the scenarios. There are 21 questions in total (multiple-choice) focusing on the main concept of CGM modelling language to measure the understanding of the student about CGM modelling language. The students was also asked to indicate their level of confident when choosing an answer. The results of the questionnaires are presented in Table 10.1. As showed in the graph plots the correct answer versus the confidence of the student in Figure 10.1, the Ph.D. Students get a better result than the M.Sc. student, so do their level of confidence. In most case, the confidence rate is on par with the number of correct answers.

M.Sc. students. It is noticeable from the result of the questionnaire that the concept of contribution and the distinction of goal vs. task and requirement were not so well explained/presented. Five out of eight of the M.Sc. students either fail to understand or misunderstood the concept of contribution. Three of them considered contribution relation between two goals as refinement, while the other two completely fail to grasp the meaning of contribution relation. Half of the M.Sc. students got confused about the concept of goal vs. task and requirement. They sometimes cannot distinguish the difference between a normal goal and a task/requirement. Two M.Sc. students with low scores actually fail to understand what is a goal model refinement in general. Surprisingly, most of the M.Sc. students got the numerical attribute/variable/SMT constrained and optimization concept quite well. Six out of eight have correct answers in the question related to the calculation of optimized realization.

Ph.D. students. The Ph.D. students, on the other hand, have done quite well on understanding the CGM modelling language. Three of them fail to score 100% due to wrong calculation of numerical attribution optimization. This, when asked, was explained

Table 10.1: CGM Modelling Language Questionnaire Result

Student	Percentages of Correct Answer	Percentages of Confidence
Ms. 1	76%	88%
Ms. 2	81%	100%
Ms. 3	48%	50%
Ms. 4	81%	82%
Ms. 5	81%	88%
Ms. 6	71%	80%
Ms. 7	86%	89%
Ms. 8	52%	36%
PhD. 1	95%	83%
PhD. 2	90%	88%
PhD. 3	100%	100%
PhD. 4	100%	94%
PhD. 5	95%	94%

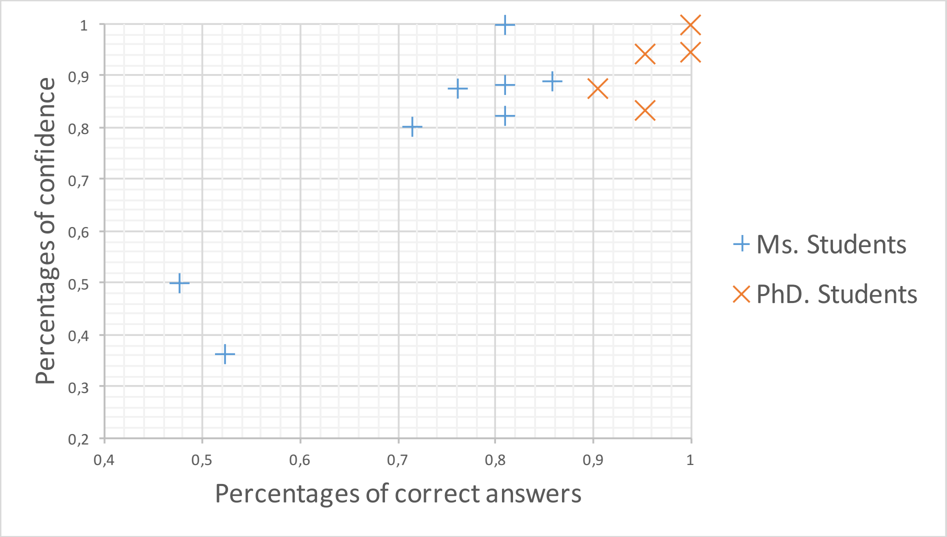


Figure 10.1: CGM Modelling Language Questionnaire Result Plot.

as their carelessness while checking the variables in the goal model graph. Two of the Ph.D. students with lower than 90% of confidence stated that the low confidence was due to their confusion about the concept of contribution relation. It was hard for them to distinguish the difference between contribution relation and a goal refinement that has only one source.

10.4.2 Application phase

In the application phase, the students were given a Scheduling Meeting scenarios (as presented in chapter 4) and were asked to build the goal model/draw the goal graph of the scenarios using both CGM modelling language and i* goal model. Later, they were asked to used the CGM-Tool to generate the optimal realization of the model based on their choices of optimization objective.

At the end of the application phase, both groups of participants were asked to filled in a questionnaire focused on their overall impression of CGM modelling language (effectiveness/expressiveness) and CGM-Tool (usability) respectively. In addition to the questionnaire, the experiment is also evaluated with the observation of the M.Sc. students during the application phase/modelling section, and the report of the Ph.D. students in the application phase.

M.Sc. students. Through observation, all of the students can use the CGM-Tool with ease when given a goal graph or using their own goal graph/model. However, five out of eight students complained about the installation steps of the tool. Most of them reported that installing the CGM-Tool and its back-end OptiMathSat separately was a real hassle. During the modelling phase, two of the students were struggling with the goal model hierarchy, they could not distinguish the goal refinement direction (confusing about parent/child in a refinement). It was also noticeable that the students had harder time building goal model with i* than CGM. Half of them can build the model (both CGM and i*) without any help. While the others can build the CGM model with some hints (mostly concerning the goal refinement hierarchy). One of the students cannot build the i* model. While the other two can build i* model after they were given some hints on the structure.

In the questionnaire, all of the students agreed that CGM language is simple to use, easy to learn with intuitive concepts, and expressive enough to describe a complex model. Five of them agreed that the CGM-Tool graphical notions are clear and easy to understand. There were mixed feedbacks in the rest of the participants, however, they are mostly concerning the size, colour of the GUI only. All of them reported that CGM is effective in capturing the given scenarios while i* can capture most part of the goal model but was not able to express variables or numerical attributes as well as optimization aspect. Seven out of eight considered the CGM modelling language is expressive and efficient in modelling requirements problem. Comparing to i*, all of the students agreed that CGM was less complicated and easier to grasp. While six of them considered CGM more expressive and efficient than i*, one of them preferred i* over CGM and the other chose “undecidable” option. All of the students agreed that CGM provided many features supported by different goal modelling language, while given the option of optimizing the candidate solution based on user-defined preferences and numerical attributes. Seven of them agreed that CGM-Tool is very intuitive and easy to use and support automatic solution generation/optimization. One of them complained about the missing concepts of actors and links. Half of the students agreed that they would want to adopt the CGM framework. One of them “refused” to use CGM framework, however, he confirmed that it was not because of the framework but rather because he did not think he would need to use goal model in future. The rest of the student choose “difficult to say” (concerning adopting CGM framework for requirements modelling).

Ph.D. students. As the results of the Ph.D. students’ reports, all of them manage to build the CGM model with ease, while only one of them was struggling with i* model. All of the Ph.D. students reported that the CGM-Tool was easy to use and they installed and run the tool without any problem.

In the questionnaire, all of the doctoral students agreed that CGM language is intuitive, expressive, and effective in modelling requirements problem. Two of them agree that the CGM-Tool graphical notions are clear and easy to understand. The rest complained about the plain GUI and the “not-so-clear” distinction between conflict edge and contribution edge graphical notions in the tool (due to the small font of the + and − sign). All of them agreed that CGM captured the given scenarios more intuitive and effective than i*. All of them agreed that CGM modelling language is expressive and efficient in modelling requirements problem. When asked to compared to i*, all of them state that it was hard to say which “less complicated” and/or “easier to grasp” as well as “more expressive and efficient” due to the difference in nature between the two language. All five of the students, however, agreed that CGM provided many features supported by different goal modelling language, while given the option of optimizing the candidate solution based on user-defined preferences and numerical attributes. They all accepted that CGM-Tool is very intuitive and easy to use and support automatic solution generation/optimization. All of the doctoral students agreed that they would want to adopt the CGM framework for requirements modelling.

10.5 Result Analysis

Based on the experiment results, we can evaluate our research questions (presented in section 10.1):

E-RQ$_1$ We cannot give the qualitative answer to this question. However, participants were able to understand all the basic concepts of the CGM modelling language and got a grasp of the CGM-Tool after a 90 minutes lectures on each. Moreover, most of the participants perceived the language as simple to use, easy to learn with intuitive concepts, and expressive enough to describe a complex model. The results of the questionnaires and observation/reports support this outcome. However, a few participants encountered difficulties in clearly distinguish the concepts of goals, tasks, and requirements in CGM as well as the concept of contribution.

E-RQ$_2$ All participants reported that CGM is effective in capturing the given scenarios while i* can capture most of the goal model except for the numerical attributes and optimization part. Participants agreed that CGM provided many features supported by different SoTA goal modelling languages, while given the option of optimizing the candidate solution based on user-defined preferences and numerical attributes. Furthermore, CGM-Tool provided automatic generation/optimization of the candidate solution.

E-RQ$_3$ Most of the participants agreed that the tool is easy to use, while a few of them gave mixed feedback about the tool. However, there was complain that it was a hassle to install the CGM-Tool and its backend (OptimathSAT) separately.

E-RQ$_4$ Most of the participants agreed that they would want to adopt the CGM framework, three of them chose ”difficult to say“, and the one refused to adopt CGM in the future. However, when asked, the ”refused“ participant confirmed that it was not because of the framework, but rather because he did not think he would need to do goal modelling in the future.

From the research question evaluation, we can assess our framework usability through the three usability criteria:

E-C$_1$ As for the learnability, it can be agreed that the CGM modelling language is simple to learn and there are not much gap in the learnability between a novice modeller (like a M.Sc. student) and an “expert” modeller (a Ph.D. student). Though the concept of contribution relation between goals of the CGM can be confusing, first time user can understand and get a grasp of it when given detailed explanation. Differentiating a goal from a task and/or a requirement in the CGM can be hard for novice modellers, however, it does not have a great impact on their ability to understand the modelling language and their capability to use it for modelling. Nevertheless, more detailed explanation, and a better, clearer description of goal/task/requirement and contribution concept should be needed to prevent any possible confusion in the future. Though there was a gap between the training phase and application phase (2 days for the group of M. Sc. students, and 4 days for the group of Ph.D. students), all the participants can still do well in the modelling. Their main obstacle while modelling is not the CGM modelling language, but the hierarchy/structure

E-C$_2$ The efficiency of the framework is proven as the participant can use CGM to model the scenarios with ease (or with some minor hints) compare to their struggling when building the model with i*. Moreover, the produced models of all participant are correct and complete, unlike when they model with i*. This also proves the effectiveness of the framework. Though not reported in previous sections, The second group participants have stated in their application phase reports that they found CGM very effective to capture even scenarios of their research works (which vary from software requirements model, business model, to security model).

E-C$_3$ The satisfaction of the users is mainly measured by their willingness to adopt the framework. Since only one of the participants refused to adopt the framework (and the reason was not related to the framework itself), we come to a conclusion that the user satisfaction of the framework is quite good. Moreover, through observation during the application phase of the first group of participants and the reports of the second group of participants, we saw that users are very interested in the CGM framework, as they asked questions and showed interest in getting to know more about the language as well as how should they apply the framework to their own research/project.

As the results of the evaluation and analysis, we can conclude that:

(i) CGM is effective in modelling requirements and requirements evolution,

(i) CGM is easy to learn and simple to use in modelling,

(i) CGM is expressive enough to capture a variety of requirements model (software/business/security/etc.), and

(i) most modellers are willing to adopt CGM framework.

10.6 Threats to Validity

In this section, we discuss the four main types of threats to validity, introduced in [WRH$^+$12].

Conclusion validity. Conclusion validity concerns all issues that would affect the ability to produce correct conclusion about the outcome of the experiment. The main threat to conclusion validity in this experiment is the low statistical power due to the low number of participants. The size of the sample is too small to come to correct conclusions.

Internal validity. Internal validity concerns all issues that may indicate a causal relationship between the experiment procedure and the outcome of the experiment. The main threat to internal validity in this experiment is that the selection of the participants is too narrow. Moreover, some of the participants were in our professional network, thus, they gained knowledge of the framework over time, and their opinion and experience might be affected.

Construct validity. Construct validity concerns the generalization of the result of the experiment to the concept and theory behind the experiment. Our evaluation suffers from the so-called mono-method bias, i.e., the subjects were treated only with our method. Participants were only asked to try out the scenarios with i* goal model, but it was not a fair comparison since i* goal model (as well as most of the current goal modelling language) does not provide many of the CGM features.

External validity. External validation concerns the ability to generalize experiment results outside of the experiment setting. The main threat to external validity in this experiment is that only small to medium sized examples was used in the experiments. Furthermore, there was a limitation in the domain aspect as only one scenario was used in the experiment.

As discussed above, the experiment meets various threats to validity. However, given the short time constraint and the limited resources available for an academic doctoral student research project, the outcome is acceptable for a dissertation in academic level. Nevertheless, re-run the experiment with more subjects, wider selection of participants, and larger/more complex sample would be necessary to obtain a more precise conclusion.

CHAPTER 11

CONCLUSIONS

We have proposed, an expressive goal-based modelling language for requirements that supports the representation of nice-to-have requirements, preferences, optimization requirements, constraints and more. Moreover, we have exploited automated reasoning solvers in order to develop a tool that supports sound and complete reasoning with respect to such goal models, and scales well to goal models with thousands of elements. Our proposal advances the state-of-the-art on goal modelling and reasoning with respect to both expressiveness and scalability of reasoning.

The contributions of this work are being exploited in several directions. [AMGM16] has proposed an expressive modelling framework for the next release problem that is founded on the same OMT/SMT solver technology as this work. [AAGM16] has offered a formalization of the next adaptation problem that chooses a next adaptation for an adaptive software system that minimizes the degree of failure over existing requirements.

We have also proposed to model changing requirements in terms of changes to CGMs. Moreover, we have introduced a new class of requirements (evolution requirements) that impose constraints on allowable evolutions, such as minimizing (implementation) effort or maximizing (user) familiarity. We have demonstrated how to model such requirements in terms of CGMs and how to reason with them in order to find optimal evolutions.

The future plan for this work includes further evaluation with larger case studies, as well as further exploration for new kinds of evolution requirements that can guide software evolution. We have also planned to do an empirical validation of the CGM-Tool with modellers and domain experts. We are currently working in this direction within our research group with PhD students and post-docs who are expert in the modelling field. Next, we will extend the validation to industrial experts of different domains. We have also planned to do different case studies with real-life-complex-large-scale goal models of a specific domain, such as Air-Traffic Control Management, healthcare, and smart cities and smart environments.

Our proposal does not address another notorious scalability problem of goal models, namely scalability-of-use. Goal models have been shown empirically to become more difficult to conceptualize and comprehend as they grow in size [ERPM06], and therefore become unwieldy for use. As with other kinds of artefacts (e.g., programs, ontologies) where scalability-of-use is an issue, the solution lies in introducing modularization facilities that limit interactions between model elements and make the resulting models easier to understand and evolve. This is an important problem on our agenda for future research on goal models.

BIBLIOGRAPHY

[AAGM16] K. Angelopoulos, F. Aydemir, P. Giorgini, and J. Mylopoulos.
Solving the next adaptation problem with prometheus.
RCIS, 2016.

[AMGM16] F. Aydemir, D. Mekuria, P. Giorgini, and J. Mylopoulos.
Scalable solutions to the next release problem: A goal-oriented perspective, 2016.
Under submission.

[Ant96] Annie I. Anton.
Goal-based requirements analysis.
In *Proceedings of the 2nd International Conference on Requirements Engineering*, ICRE '96, pages 136–. IEEE Computer Society, 1996.

[AP98] Annie I. Anton and Colin Potts.
The use of goals to surface requirements for evolving systems.
In *Proceedings of the 20th international conference on Software engineering*, ICSE '98, pages 157–166. IEEE Computer Society, 1998.

[BDHM13] Alexander Borgida, Fabiano Dalpiaz, Jennifer Horkoff, and John Mylopoulos.
Requirements models for design- and runtime: A position paper.
In *Proceedings of the 5th International Workshop on Modeling in Software Engineering*, MiSE '13, pages 62–68. IEEE Press, 2013.

[Bia94] Randolph G. Bias.
Usability inspection methods.
chapter The Pluralistic Usability Walkthrough: Coordinated Empathies, pages 63–76. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[BSST09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli.

Satisfiability Modulo Theories.
In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.

[CBH11] Andreas Classen, Quentin Boucher, and Patrick Heymans.
A text-based approach to feature modelling: Syntax and semantics of TVL.
Sci. Comput. Program., 76(12):1130–1143, 2011.

[CGS11] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani.
Computing Small Unsatisfiable Cores in SAT Modulo Theories.
Journal of Artificial Intelligence Research, JAIR, 40:701–728, April 2011.

[CGSS13] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani.
The MathSAT 5 SMT Solver.
In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'13.*, volume 7795 of *LNCS*, pages 95–109. Springer, 2013.

[CKM02] Jaelson Castro, Manuel Kolp, and John Mylopoulos.
Towards requirements-driven information systems engineering: The tropos project.
Inf. Syst., 27(6):365–389, September 2002.

[CL99] Larry L. Constantine and Lucy A. D. Lockwood.
Software for Use: A Practical Guide to the Models and Methods of Usage-centered Design.
ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[DR93] Joseph F. Dumas and Janice C. Redish.
A Practical Guide to Usability Testing.
Greenwood Publishing Group Inc., Westport, CT, USA, 1st edition, 1993.

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas.
Goal-directed requirements acquisition.
Sci. Comput. Program., 20(1-2):3–50, April 1993.

[EBJ11] Neil A. Ernst, Alexander Borgida, and Ivan Jureta.
Finding incremental solutions for evolving requirements.
In *RE*, pages 15–24. IEEE, 2011.

[EBJM14] Neil A. Ernst, Alexander Borgida, Ivan J. Jureta, and John Mylopoulos. Agile requirements engineering via paraconsistent reasoning. *Information Systems*, 43:100 – 116, 2014.

[EBMJ12] Neil A. Ernst, Alexander Borgida, John Mylopoulos, and Ivan Jureta. Agile Requirements Evolution via Paraconsistent Reasoning. In Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 382–397. Springer, 2012.

[EMBJ10] Neil A. Ernst, John Mylopoulos, Alex Borgida, and Ivan J. Jureta. Reasoning with optional and preferred requirements. In Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo, and Yair Wand, editors, *Conceptual Modeling – ER 2010: 29th International Conference on Conceptual Modeling, Vancouver, BC, Canada, November 1-4, 2010. Proceedings*, pages 118–131, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[ERPM06] Hugo Estrada, Alicia Martínez Rebollar, Oscar Pastor, and John Mylopoulos. An empirical evaluation of the *i** framework in a model-based software generation environment. In Eric Dubois and Klaus Pohl, editors, *Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg, June 5-9, 2006, Proceedings*, volume 4001 of *Lecture Notes in Computer Science*, pages 513–527. Springer, 2006.

[FLM+04] Ariel Fuxman, Lin Liu, John Mylopoulos, Marco Pistore, Marco Roveri, and Paolo Traverso. Specifying and analyzing early requirements in tropos. *Requir. Eng.*, 9(2):132–150, May 2004.

[GMNS04] Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Formal reasoning techniques for goal models. *JOURNAL OF DATA SEMANTICS*, 1:1–20, 2004.

[Hor12] Jennifer Marie Horkoff. *Iterative, Interactive Analysis of Agent-goal Models for Early Requirements Engineering*.

PhD thesis, University of Toronto, 2012.

[HPSS00] I. Horrocks, P. F. Patel-Schneider, and R. Sebastiani.
An Analysis of Empirical Testing for Modal Decision Procedures.
Logic Journal of the IGPL, 8(3):293–323, May 2000.

[IEE90] Ieee standard glossary of software engineering terminology.
IEEE Std 610.12-1990, pages 1–84, Dec 1990.

[ISO98] ISO.
Ergonomic requirements for office work with visual display terminals (vdts) — Part 11: Guidance on usability.
Iso, International Organization for Standardization, 1998.

[ISO06] ISO.
Ergonomics of human system interaction — Part 110: Dialogue principles.
Iso, International Organization for Standardization, 2006.

[JBEM10] Ivan Jureta, Alexander Borgida, Neil A. Ernst, and John Mylopoulos.
Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling.
In *RE*, pages 115–124. IEEE Computer Society, 2010.

[JMF08] Ivan Jureta, John Mylopoulos, and Stephane Faulkner.
Revisiting the core ontology and problem in requirements engineering.
In *Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*, RE '08, pages 71–80. IEEE Computer Society, 2008.

[Lap05] Alexei Lapouchnian.
Goal-Oriented Requirements Engineering: An Overview of the Current Research.
Technical report, Department of Computer Science, University of Toronto, Juni 2005.

[Leh80] Meir M. Lehman.
Programs, Life Cycles, and Laws of Software Evolution.
In *Proceedings of the IEEE*, pages 1060–1076, September 1980.

[Lia12] Sotirios Liaskos.
On eliciting contribution measures in goal models.

In *Proceedings of the 2012 IEEE 20th International Requirements Engineering Conference (RE)*, RE '12, pages 221–230. IEEE Computer Society, 2012.

[LL04] Axel Lamsweerde and Emmanuel Letier.
From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering, pages 325–340.
Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[LMSM10] Sotirios Liaskos, Sheila A. McIlraith, Shirin Sohrabi, and John Mylopoulos.
Integrating preferences into goal models for requirements engineering.
In *RE*, pages 135–144. IEEE Computer Society, 2010.

[LR94] Clayton Lewis and John Rieman.
Task-Centered User Interface Design:A Practical Introduction.
1994.

[MCN92] J. Mylopoulos, L. Chung, and B. Nixon.
Representing and using nonfunctional requirements: A process-oriented approach.
IEEE Trans. Softw. Eng., 18(6):483–497, June 1992.

[Mek] Dagmawi Neway Mekuria.
Constrained goal modeling and reasoning tool's user manual.

[Nie93] Jakob Nielsen.
Usability Engineering.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[NO06] Robert Nieuwenhuis and Albert Oliveras.
On SAT Modulo Theories and Optimization Problems.
In *Proc SAT'06*, volume 4121 of *LNCS*. Springer, 2006.

[NSGM16a] Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos.
Multi-objective reasoning with constrained goal models.
Requirements Engineering Journal, pages 1–37, 2016.

[NSGM16b] Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos.
Requirements Evolution and Evolution Requirements with Constrained Goal Models.

In *Proceedings of the 37nd International Conference on Conceptual Modeling*, LNCS. Springer, 2016.

[NSGM17] Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos.
Modeling and reasoning on requirements evolution with constrained goal models.
Software Engineering and Formal Method, 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, 2017.
submitted.

[PDP$^+$12] Elda Paja, Fabiano Dalpiaz, Mauro Poggianella, Pierluigi Roberti, and Paolo Giorgini.
STS-Tool: socio-technical security requirements through social commitments.
In *Proceedings of the 20th IEEE International Conference on Requirements Engineering*, pages 331–332, 2012.

[PRS$^+$94] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey.
Human-Computer Interaction.
Addison-Wesley Longman Ltd., Essex, UK, UK, 1994.

[PSS03] P. F. Patel-Schneider and R. Sebastiani.
A New General Method to Generate Random Modal Formulae for Testing Decision Procedures.
Journal of Artificial Intelligence Research, (JAIR), 18:351–389, May 2003.

[SCR11] Christian Schalles, John Creagh, and Michael Rebstock.
Usability of modelling languages for model interpretation: An empirical research report, 2011.

[Seb07] Roberto Sebastiani.
Lazy Satisfiability Modulo Theories.
Journal on Satisfiability, Boolean Modeling and Computation, JSAT, 3(3-4):141–224, 2007.

[SGM04] R. Sebastiani, P. Giorgini, and J. Mylopoulos.
Simple and Minimum-Cost Satisfiability for Goal Models.

In *Proc. 16th International Conference on Advanced Information Systems Engineering - CAISE'04*, LNCS, Riga, Latvia, May 2004. Springer.

[Sha91] Brian Shackel.
Human factors for informatics usability.
chapter Usability&Mdash;Context, Framework, Definition, Design and Evaluation, pages 21–37. Cambridge University Press, New York, NY, USA, 1991.

[Sou12] Vitor E. S. Souza.
Requirements-based Software System Adaptation.
Phd thesis, University of Trento, 2012.

[ST12] Roberto Sebastiani and Silvia Tomasi.
Optimization in SMT with LA(Q) Cost Functions.
In *IJCAR*, volume 7364 of *LNAI*, pages 484–498. Springer, July 2012.

[ST15a] Roberto Sebastiani and Silvia Tomasi.
Optimization Modulo Theories with Linear Rational Costs.
ACM Transactions on Computational Logics, 16(2), March 2015.

[ST15b] Roberto Sebastiani and Patrick Trentin.
OptiMathSAT: A Tool for Optimization Modulo Theories.
In *Proc. International Conference on Computer-Aided Verification, CAV 2015*, volume 9206 of *LNCS*. Springer, 2015.

[ST15c] Roberto Sebastiani and Patrick Trentin.
Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions.
In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, volume 9035 of *LNCS*. Springer, 2015.

[ST17] Roberto Sebastiani and Patrick Trentin.
On optimization modulo theories, maxsmt and sorting networks.
In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'17.*, LNCS, 2017.

[VL01] Axel Van Lamsweerde.
Goal-oriented requirements engineering: A guided tour.

In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 249–. IEEE Computer Society, 2001.

[vLL00] Axel van Lamsweerde and Emmanuel Letier.
Handling obstacles in goal-oriented requirements engineering.
IEEE Trans. Softw. Eng., 26(10):978–1005, October 2000.

[vLLD98] Axel van Lamsweerde, Emmanual Letier, and Robert Darimont.
Managing conflicts in goal-driven requirements engineering.
IEEE Trans. Softw. Eng., 24(11):908–926, November 1998.

[WRH$^+$12] Claes Wohlin, Per Runeson, Martin Host, Magnus Ohlsson, and Bjorn Regnell.
Experimentation in Software Engineering.
Springer, 2012.

[WRLP94] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson.
Usability inspection methods.
chapter The Cognitive Walkthrough Method: A Practitioner's Guide, pages 105–140. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[Yu97] Eric S. K. Yu.
Towards modeling and reasoning support for early-phase requirements engineering.
In *RE '97: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, page 226. IEEE Computer Society, 1997.

[ZHM07] Yuanyuan Zhang, Mark Harman, and S. Afshin Mansouri.
The multi-objective next release problem.
In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 1129–1137, New York, NY, USA, 2007. ACM.