A Refinement Calculus for Requirements Engineering (CaRE)

John Mylopoulos
University of Ottawa

Centro de Informática, FUPE
Recife, November 13, 2019

Abstract

The requirements problem consists of transforming stakeholder requirements - however informal, ambiguous, conflicting, unattainable, imprecise and incomplete – into a consistent, complete and realizable specification through a systematic process. We propose a refinement calculus for requirements engineering (CaRE) for solving this problem, which takes into account the typically dialectical nature of requirements activities. The calculus casts the requirement problem as an iterative argument between stakeholders and requirements engineers, where posited requirements are attacked for being ambiguous, incomplete, etc. and refined into new requirements that address the defect pointed out by the attack. Refinements are carried out by operators provided by CaRE that refine (e.g., strengthen, weaken, decompose) existing requirements, to build a refinement graph. The semantics of the operators is provided by using the notion of acceptable arguments in Dung's argumentation theory.

This is joint work with Yehia ElRakaiby, Alessio Ferrari and Alex Borgida.

The requirements problem

In its original formulation [Jackson95], a requirements problem consists of finding a ***specification*** S for a given set of ***requirements*** R and ***indicative environment properties*** E such that

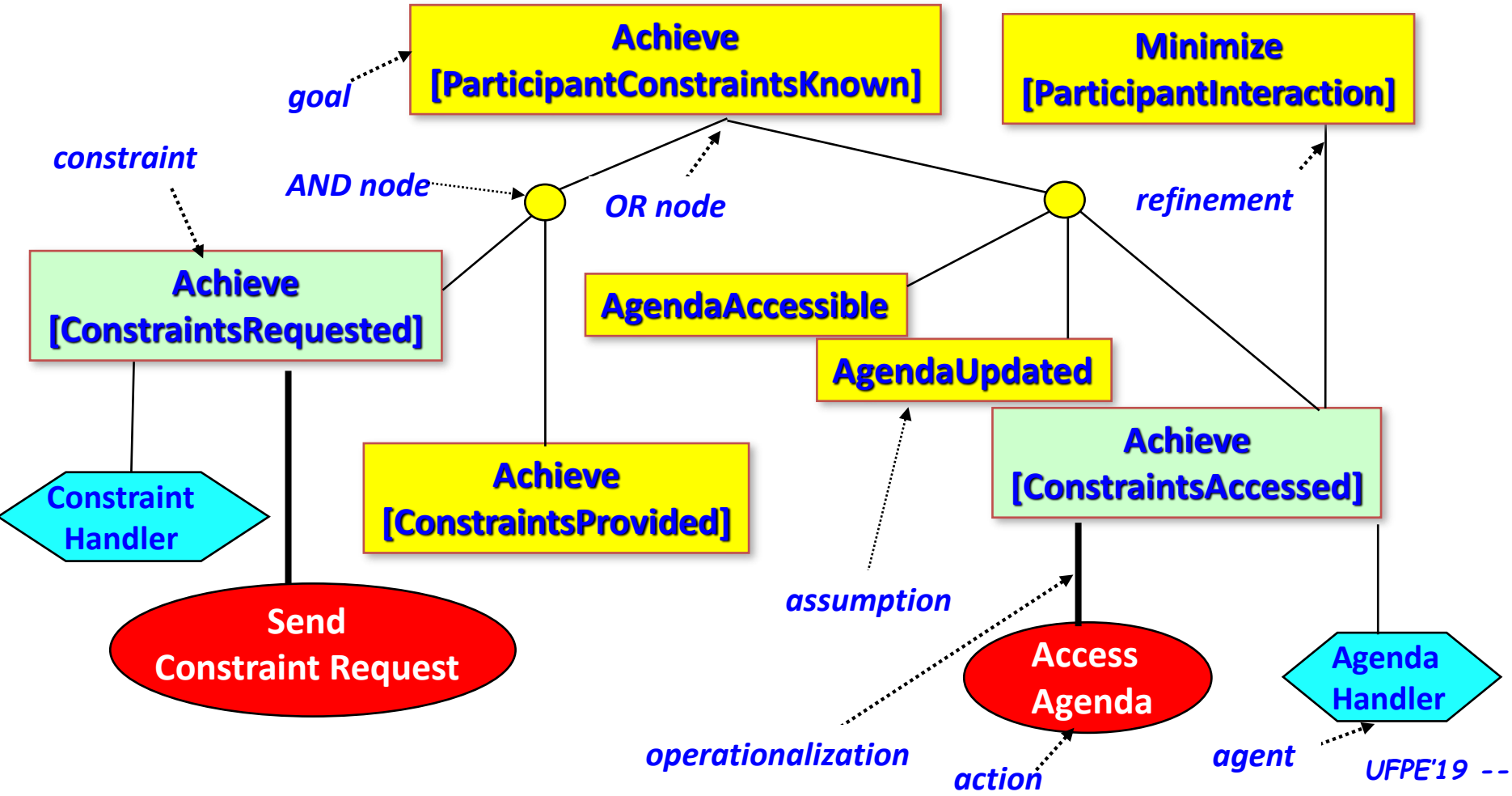
E, S |- R

meaning: "… satisfaction of the requirements can be deduced from satisfaction of the specification, together with the environment properties…" [Jackson95]

Solution through refinement (as in program refinement): Start with requirements and keep refining them to eliminate mention of non-executable elements.

Requirements as goals

Requirements are now goals and (requirements) problem solving amounts to incremental AND/OR goal refinement (Axel van Lamsweerde, c.1993).



Interesting ideas in RE ...

Requirements derived via refinement from models of the ***domain*** (Ross, c.1977).

Stakeholder requirements and ***specifications*** are different things, though logically related (Jackson&Zave, c.1995).

Requirements are stakeholder ***goals*** (vanLamsweerde, c.1993).

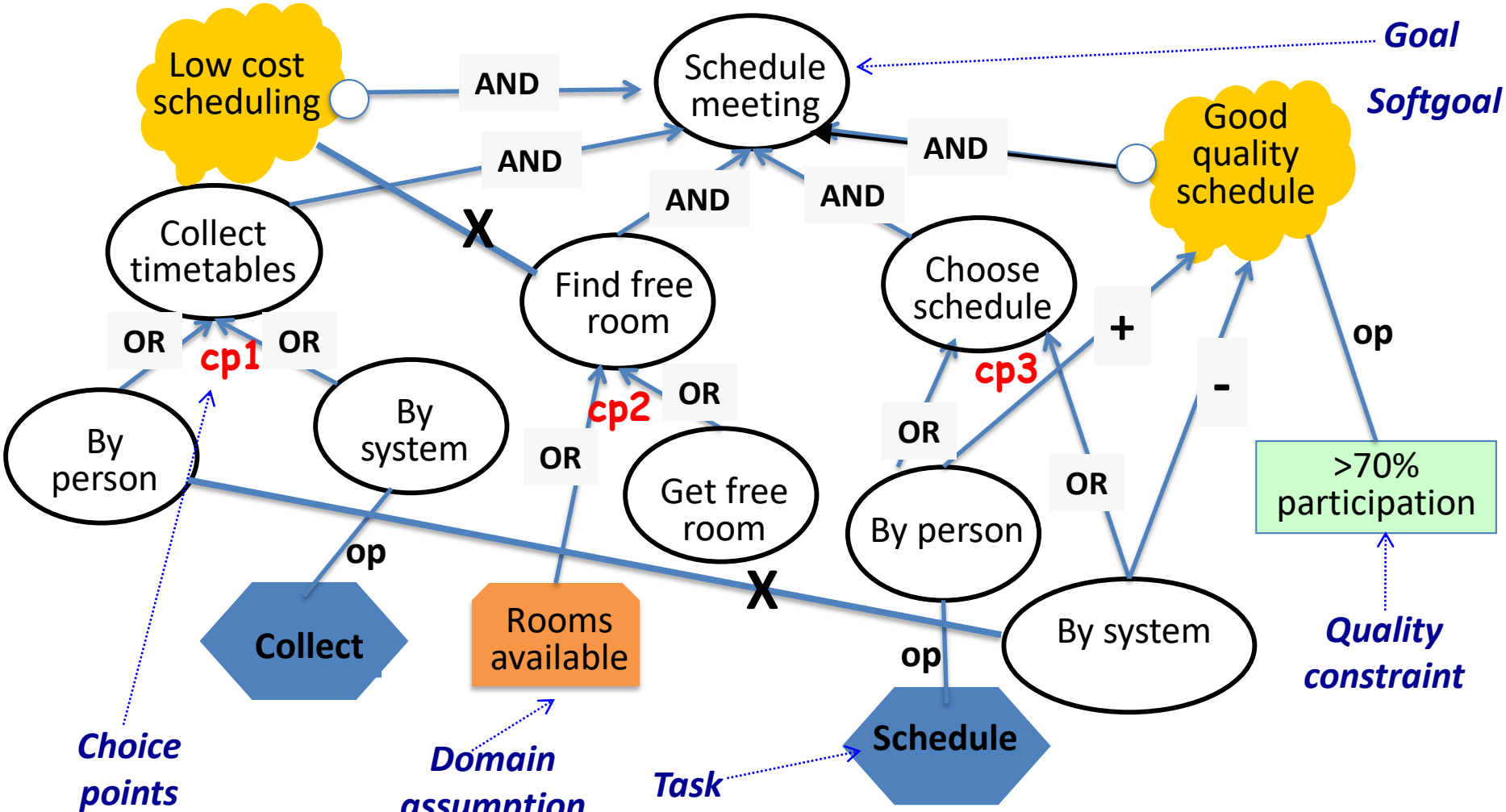
The requirements problem is a ***social*** problem, calls for social solutions (Yu, c.1993).

The requirements problem is solved through problem ***refinement*** (all), and this refinement has many forms: activity decomposition (Ross), abductive inference (Jackson), goal refinement (vanLamsweerde), social delegation (Yu).

With goal models and refinement, you are not exploring a design, but rather a ***design space*** (GORE).

Goal models circa 2018

- Goals can be mandatory/nice-to-have, can have priorities [Liaskos10], probabilities [Letier04], utilities, …



What do these models tell us?

- They allow us to derive alternative specifications (***solutions***) – each consisting of functions/tasks/actions, quality constraints and assumptions for fulfilling requirements.
- These models are founded on two important concepts of Science and Engineering: ***refinement*** and ***operationalization***.

Refinement

- Literally means "the process of removing impurities/defects/unwanted elements", as with oil or sugar refineries.
- Refinement has an illustrious history in Computer Science, specifically in programming methodology (Abrial, Hoare et al), used to remove non-executability defects.
- In GORE, refinement has been used to iteratively reduce a goal G to specification S (collections of tasks, quality constraints) and assumptions such that A, S |- G
- Goal refinements come in two flavours, AND/alternative:

  $G \rightarrow_{AND} G_1, G_2, \ldots, G_n$

  $G \rightarrow G_1, G \rightarrow G_2, \ldots, G \rightarrow G_n$
- Refinements ***decompose*** or ***reduce*** a goal into subgoals., the defect being non-atomicity.

Operationalization

In spoken English, operationalization means “to make something operational/working” [Spoken]

Operationalizing a goal in terms of a task/action uses this [Spoken] sense.

In the Sciences (Natural, Life and Social), operationalization means “defining a concept that is not directly observable through the operations by which we measure it” [Sciences] [Bridgeman27].

e.g., ‘mass’ can be operationalized inertially or gravitationally.

Operationalizing a non-functional requirement in terms of quality constraints/metrics uses that [Sciences] sense.

Operationalization marks the boundary between problem and solution space.

But, many things can't be told in GORE …

- Stakeholder requirements are often:
  - ✓ Redundant/not needed so they can be dropped

    "Highly secure system" → X (forget it, not needed!)
  - ✓ Unattainable, so they need to be weakened,

    "7/24 availability" → "office hour availability"
  - ✓ Conflicting, so they need to be weakened

    "Low cost" & "High security" → "modest cost" & "secure from DoS attacks"
- Refinements provided by GORE can't address problems of unattainability, conflict, ambiguity, incompleteness, etc.
- … we need a new calculus!

A calculus for RE (CaRE)

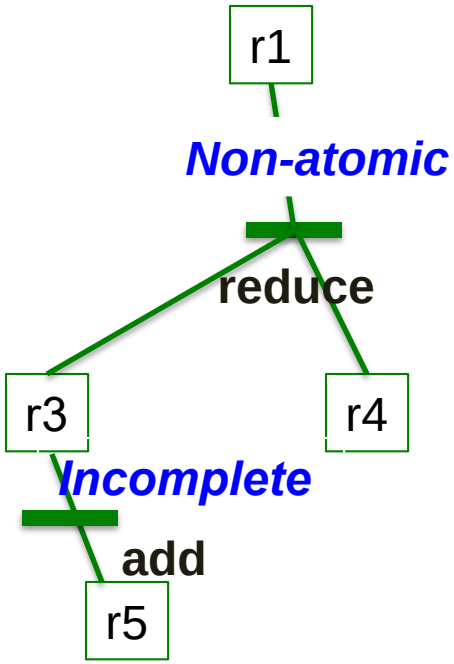
- Consists of refinement operators that operate on requirements and derive other requirements.
- Through this calculus we propose to solve the requirements problem incrementally by applying refinement operators until we can derive specifications/solutions from the refinement graph that address all their attacks.
- Incremental refinement is cast in the form of a Hegelian dialectical argument between stakeholders (including requirements engineers) where posited requirements are ***attacked*** for being ambiguous/incomplete/conflicting/unattainable/non-atomic/… etc., and ***refinements*** are proposed that eliminate requirements under attack.

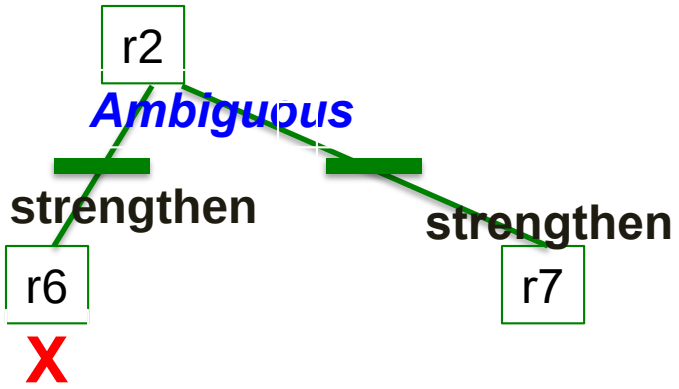
Examples

- r1:= “System shall schedule meetings upon request”
- r2:= “Meeting schedules shall be of good quality”
- Non-atomic(r1) “No single function for r1” (reduce)➔
  r3:= “Collect timetables”,r4:= “Generate a schedule”
- Incomplete(r3) “No privacy requirement” (add) ➔
  r5:= “timetable data shall be confidential”
- Ambiguous(r2) (strengthen)➔ r6:=“≥70% participation rate”
- Rejected(r6)
- Ambiguous(r2) (strengthen)➔ r7:=“≥80% participation rate”

…

Refinement graph





- r1:= “System shall schedule meetings upon request”
- r2:= “Meeting schedules shall be of good quality”
- r3:= “Collect timetables”, r4:= “Generate a schedule”
- r5:= “timetable data are confidential”
- r6:=“≥70% participation”
- r7:=“≥80% participation”

Hegelian dialectics

For Hegel, dialectics is a form of argument where a thesis is attacked with an antithesis, leading to a synthesis.

This is a different form of dialectic than the original version presented by Plato (and practiced by Socrates).

For our purposes, a thesis is a requirement (or set thereof), an antithesis is a attack that points to a defect of the requirement(s), and a synthesis is the result of a refinement that refines attacked requirement(s) into new ones that don't have the defect.

Hegel calls refinement a sublation (from German verb "aufheben", to sublate) meaning that a refinement at the same time cancels (or negates) and preserves what it refines [SEP16].

Refinement operators

- CaRE operators are as follows:
  - ✓ Strengthen(r): refines r into r' such that r' ⇒ r.
  - ✓ Weaken(r): refines r into r' such that r ⇒ r'.
  - ✓ Reduce(r): refines r into r1, …, rn such that r1 ∧ … ∧ rn ⇒ r
  - ✓ Add(r): refines r into r' such that requirement r has not been dealt with until r' has.
  - ✓ Resolve({r1,…,rn}): refines r1,…,rn into r1',…,rm' such that each ri' i=1,…m there is some rj, j=1…n such that rj ⇒ ri'.
- The semantics of 'dealt with' is analogous to that for acceptability in argumentation logic: a requirement is ***dealt with*** if all attacks against it have been dealt with.
- This means that leaf nodes of an argumentation graph not under attack have been dealt with; moreover, they are atomic, unambiguous, attainable, non-conflicting etc.

Attack types

- Attack types are inspired by the IEEE 1998 standard on requirements specifications [IEEE98]:
  - ✓ Non-atomic(r): there is no function that fulfills r.
  - ✓ Ambiguous(r): r has multiple interpretations
  - ✓ Unattainable(r): r can't be fulfilled.
  - ✓ Unjustified(r): unclear why is r needed.
  - ✓ TooStrong/TooWeak(r)
  - ✓ Rejected(r)
  - ✓ Conflicting({r1,…rn})
  - ✓ Incomplete(r)
- For each of these attack types, there is at least one operator that can be applied to eliminate the defect pointed out by the attack, with the exception of 'Rejected'.

Refinement graphs

- Refinement graphs are labelled hyper-graphs, with nodes representing requirements, hyper-edges representing refinements each taking one or more inputs and having one or more nodes as outputs. Edge labels represent attack types and operators.
- Note that such graphs may have cycles:

Unjustified(r:="Collect timetables") (add)➔r':="Schedule mtg"

Non-atomic(r') (reduce)➔r, r2:=…

The requirements problem, revisited

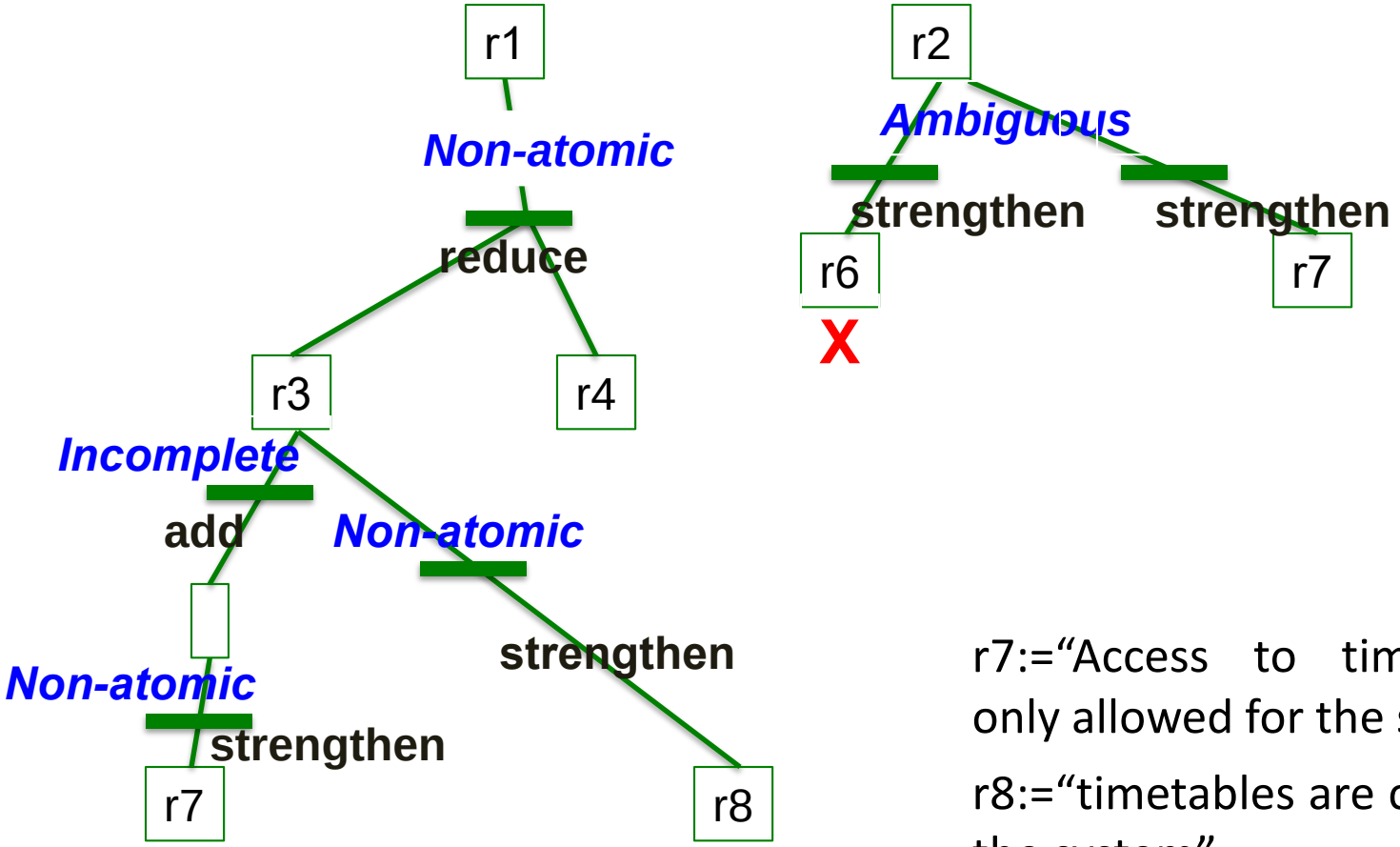
Given a refinement hyper-graph, the requirements problem can be recast in two ways:

- ✓ (RP-x) Is there a specification that deals with all root-level requirements?
- ✓ (RP-all) Find all specifications that deal with all root-level requirements.

For RP-x, we adopt a label-propagation algorithm.

For RP-all there is a combinatorial algorithm, we are looking for better …

Label propagation on refinement graphs



r7:="Access to timetables is only allowed for the scheduler"

r8:="timetables are collected by the system"

Solving RP-x

- ✓ Label leaf-level nodes not under attack ***S*** (solved).
- ✓ When all outgoing nodes of a refinement edge are ***S***, this refinement has been dealt with, label the edge ***S***.
- ✓ When for every attack on a node there is at least one refinement edge that is ***S***, label the node ***S***.

Note: RP-x determines if there is a solution to a RP, i.e., an ***S*** labelling of unattacked leaf level nodes that leads to labelling ***S*** all root-level nodes.

Solving RP-all (naïve)

Note that specifications must be ***minimal*** solutions, otherwise we will always have O(2**n) solutions (where n is number of unattacked leaf nodes) for any refinement graph that has a solution.

Naïve algorithm:

- ✓ S := fully labelled refinement graph that has a solution,
- ✓ Try all combinations of removing one ***S*** leaf label and assign all solutions to S.
- ✓ Repeat this process until you find solutions that don't have any smaller solution; each of these constitutes a specification.

Meta-comment: I'm sure we can do better …

History

- We proposed a calculus for RE in the PhD thesis of Feng-Lin Li (University of Trento, 2016) [FengLinLi16].
- The calculus was more elaborate in the refinement operators it offered than CaRE. For example, a quality goal could be weakened in a probabilistic, fuzzy or user-oriented sense.
- However, that proposal didn't come with a semantics of what does it mean for a specification to "solve" or "deal with" stakeholder requirements, when in fact these requirements may have been rejected, weakened or supplemented during analysis in search of specifications.

Summary

- We have proposed a calculus for RE, that supports the incremental derivation of a specification from stakeholder requirements.
- Our proposal extends GORE techniques by introducing operators that can add new requirements, or weaken and even reject stakeholder requirements.
- The meaning of the claim “Specification S deals with requirements R, assuming assumptions A” has been cast in argumentation logic semantics, instead of GORE semantics.

References

- [Bridgeman27] Bridgeman P., *The Logic of Modern Physics*, 1927.
- [Dardenne93] Dardenne, A., van Lamsweerde, A. and Fickas, S.,"Goal-Directed Requirements Acquisition", in The Science of Computer Programming 20, 1993.
- [ElRakaiby18] ElRakaiby Y., Ferrari A., Mylopoulos J., "A Refinement Calculus for Requirements Engineering Based on Argumentation Semantics", submitted for publication.
- [Feng-LinLi16] Feng-Lin Li, *A Refinement Calculus for Requirements Engineering*, PhD thesis, Department of Information Engineering and Computer Science (DISI), University of Trento, January 2016.
- [IEEE98] IEEE, *Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1998, 1–40, October 1998.
- [Jackson95] Jackson M., Zave P., "Deriving Specifications from Requirements: An Example", 17th International Conference on Software Engineering (ICSE'95).
- [Jureta10] Jureta, I., Borgida, A., Ernst, N., Mylopoulos, J., "Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences and Inconsistency Handling", 19th Int. IEEE Conference on Requirements Engineering (RE'10), Sydney, Sept. 2010.

References (cont'd)

[Letier04] Letier E., van Lamsweerde A., "Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering", 12th Int. Symposium on Foundation of Software Engineering, 53–62, Newport Beach CA, Nov. 2004.

[Liaskos10] Liaskos, S., McIlraith, S., Sohrabi, S., Mylopoulos, J., "Integrating Preferences into Goal Models for Requirements Engineering", 19th International IEEE Conference on Requirements Engineering (RE'10), Sydney Australia, September 2010.

[Ross77] Ross, D., Schoman T., "Structured Analysis: A Language for Communicating Ideas," *IEEE Transactions on Software Engineering 3(1)*, Special Issue on Requirements Analysis, January 1977, 16-34.

[SEP16] Stanford Encyclopedia of Philosophy, "Hegel's Dialectics", June 2016 .

[Yu93] Yu Eric, "Modelling Organizations for Information Systems Requirements Engineering", First IEEE International Symposium on Requirements Engineering (ISRE'93), San Jose, January 1993.