Static and Dynamic Definitions of Refinement

Ivan J. Jureta
FNRS & Louvain School of Management
University of Namur
ivan.jureta@fundp.ac.be

Stéphane Faulkner
Louvain School of Management
University of Namur
stephane.faulkner@fundp.ac.be

***Abstract*—Stakeholders express in natural language their requirements about a system-to-be. It is a longstanding conceptual and methodological challenge to systematically increase the detail and precision of natural language requirements to the point at which their rewriting in a predicate formalism becomes relevant and formal methods applicable. This paper (i) argues that the current de facto standard refinement approach in requirements engineering (i.e., Darimont and van Lamsweerde's goal refinement [5]) cannot be relevantly applied to refine requirements in natural language, and (ii) proposes a new form of refinement, called nl-refinement, designed specifically for the refinement of natural language requirements. NL-refinement is defined and its use illustrated to increase the detail and precision of natural language requirements to the point where standard forms of refinement apply.**

***Keywords*-refinement of requirements**

I. Introduction

The research and practice of requirements engineering (RE) clearly shows that it requires considerable skill, effort, and guidance to arrive at a point where most of the important requirements for a system-to-be are known, and only a few alternative ways of satisfying them are identified. A prominent source of difficulty is the fact that the staleholders in RE communicate *in natural language* (NL) their beliefs, desires, intentions, and preferences about the system-to-be. Because the specification of the system-to-be is by definition a detailed and precise artifact, and the NL information obtained from the stakeholders is imprecise and lacks detail, a gap stands between the two representations. Bridging this gap in a systematic, controlled, but also feasible process is a longstanding conceptual and methodological challenge.

Systematic increase in detail and precision is an old problem in computer science, and has been with RE since the early days. The seminal works of Wirth [22], Dijkstra [7], and Hoare [10] established *program refinement* as a systematic method for replacing pieces of abstract programs with pieces of concrete programs. Fundamental principles of program refinement, such as proof obligations on the abstract and concrete artifacts have their analogues today in Darimont and van Lamsweerde's *goal refinement* [5], the current de facto standard form of refinement in RE. It should be clear at the outset that we have nothing to contribute in this paper towards the improvement or revision of these two basic forms of refinement. What we do argue is that an additional form of refinement is needed; namely, one that is designed precisely for the refinement of NL descriptions of the system-to-be, and thereby applicable *before* goal refinement becomes relevant, and certainly much earlier than classical program refinement is.

To understand the problem discussed in this paper, we recall the two forms of goal refinement, defined within the context of the KAOS framework [4] for RE. We can see the AND-goal refinement as a triple (C, AND, g), where C is called a *refinement set*, and is a set $C = \{g_1, g_2, \dots, g_n\}$ with $n > 2$ goals. g is a goal, and AND is the relationship indicating that g is satisfied if and only if all goals in C are satisfied. The members of C are typically called *concrete* (i.e., more precise and/or detailed) goals, in contrast to the *abstract* (i.e., less precise and/or detailed) goal g, which they AND-refine. An OR-goal refinement is the triple (C, OR, g), where the relationship OR indicates that g will be satisfied if any one of the goals in R is satisfied. Both the AND and OR relationships obey specific conditions, which play the same role as proof obligations in program refinement. The interest in goal refinement stems from precise proof obligations. For AND-goal refinement, these are: (C1) that the satisfaction of the abstract goal is provable from the satisfaction of the concrete goals (i.e., $g_1 \wedge g_2 \wedge \dots \wedge g_n \vdash g$), (C2) that the set of concrete goals contains only the absolutely necessary concrete goals (i.e., $\forall i \in \{1, \dots, n\}$, $\bigwedge_{j \neq i} g_j \nvdash g$), (C3) that the concrete goals are not inconsistent (i.e., $g_1 \wedge g_2 \wedge \dots \wedge g_n \nvdash \perp$), and (C4) that the concrete goals are not a trivial rewriting of the abstract goal (i.e., $n > 1$). By definition [5], it is only if all of these conditions (C1)–(C4) hold that it can be said that $g_1, \dots, g_n$ AND-refine g. We will recall proof obligations for OR-goal refinement further down.

A convenient way to represent goal refinements is via directed acyclic graphs, where AND- and OR-labeled links relate nodes that capture concrete and abstract goals. It is common to see these graphs being drawn "informally", that is, without checking the relevant proof obligations. This typically happens very early on in RE, when requirements expressed in NL are being manipulated. Doing so subsumes the assumption that Darimont and van Lamsweerde's goal refinement applies not only to those requirements, which are precise and detailed enough to be specified in a linear temporal Boolean first-order logic (as in KAOS) and to which formal refinement patterns apply [5], but also to the early requirements expressed in NL. While goal refinement certainly *can* be applied in such a simplistic way to the early requirements expressed in NL, the important question is whether it is relevant to do so.

This paper argues that *goal refinement cannot be relevantly applied to the early requirements expressed in* NL. It cannot be *relevantly* applied, because doing so admits two paradoxes, called *the soundness paradox* and *the maturity paradox*. Roughly speaking, the former says that the form of reasoning that can be applied when refining, namely ampliative inference, violates the soudness property, all the while the proof obligations that must be obeyed when refining are defined in a sound formalism. The second says that if proof obligations do verify for a given (C, AND, g) (or (C, OR, g)), then this refinement is certain and indubitable, which contradicts the fact that the early requirements in NL are subject to vagueness and ambiguity, among others, which makes the refinement carrying this information necessarily only tentative, preliminary, and open to revision, and hence *neither* certain *nor* indubitable. We consequently propose a new form of refinement, called NL-refinement, short of *natural language refinement*, designed specifically for the refinement of requirements elicited early on in RE and expressed in NL. NL-refinement suffers neither from the soundness paradox nor from the maturity paradox.

The rest of the paper is organized as follows. In discussing related work, we briefly recall classical program refinement and Banach's retrenchment, then outline the salient features (not mentioned above) of goal refinement (§II). The soundness and maturity paradoxes are then presented (§III), and NL-refinement is defined and illustrated (§IV). We close with a summary of conclusions and directions for future work (§V).

II. Related Work

Program refinement consists of replacing a piece of abstract program with a piece of more concrete program [22], [7], [10]. Detail can thereby be added incrementally, with the benefit of delaying lower level detail to later steps of system development. The various kinds of refinement (for an overview, cf., e.g., [6]) share the common strategy for establishing the correctness of an implementation. Namely, for every run of the concrete system, there must be a run of the abstract system, which maintains the correct correspondence between these runs. The idea is that there is a correct correspondence between concrete and abstract models of the systems in question, which leads to proof obligations. For illustration, we borrow from Banach and colleagues [2] the following formulation of the proof obligation for generalized forward refinement correctness:

$$R(u,v) \land RIn_{Op}(i,j) \land pre(Op_a)(u,i) \land Op_b(v,j,v',p) \Rightarrow \\ (\exists u', o \bullet Op_a(u,i,u',o) \land R(u',v') \land ROut_{Op}(o,p))$$

Above, u, v are abstract/concrete states that are primed for after-states; i, j are abstract/concrete inputs; o, p are abstract/concrete outputs; Op_a and Op_b are abstract/concrete versions of the operation Op. R is the retrieve relation, whereas $RIn_{Op}, ROut_{Op}$ are input/output relations respectively. With generalized forward refinement, one relates a sequence of (at least one) steps of an abstract model of a system *en bloc*, to a sequence of steps of a concrete model *en bloc*. The aim is to show that the retrieve relation is preserved on the states between the ends of the two sequences, although it need not necessarily be preserved at points internal to the two sequences.

Proof obligations make refinement a powerful tool for progressing from abstract to concrete models of a system-to-be. However, as Banach and colleagues observe ([2]: p.102), "[s]ince the human notion of abstraction is inevitably imprecise, and the mathematical notion of abstraction pertaining to any specific refinement formalism is de facto extremely precise, some dislocation between the two is bound to occur sometimes." In order to accommodate these "dislocations", that is, cases when refinement proves too restrictive, Banach and colleagues introduced [3] *retrenchment*. Retrenchment manipulates proof obligations; e.g., the retrenchment reformulation of the proof obligation for generalized forward refinement correctness is:

$$R(u,v) \land W_{Op}(i,j,u,v) \land Op_b(v,j,v',p) \Rightarrow \\ (\exists u', o \bullet Op_a(u,i,u',o) \land R(u',v') \land O_{Op}(o,p,u',v',i,j,u,v) \\ \lor C_{Op}(u',v',o,p,i,j,u,v))$$

Above, $RIn_{Op} \land pre(Op_a)$ has been generalized to the relation $W_{Op}(i,j,u,v)$, which is an arbitrary relation in the before-values. $ROut_{Op}$ has been generalized to the retrenchment output relation $O_{Op}(o,p,u',v',i,j,u,v)$, which allows all the variables to occur. The concedes relation $C_{Op}(u',v',o,p,i,j,u,v)$ describes what happens if the retrieve relation cannot be reestablished by a given pair of abstract/concrete steps. Retrenchment can be used to handle cases when refinement proves too restrictive, but also to capture general evolutions of system definitions (e.g., [3], [2]).

In parallel with the development of retrenchment, RE has seen the introduction and widespread use of AND-goal refinement and OR-goal refinement. The same benefits are pursued as for refinement and retrenchment, which is to delay details until later. AND- and OR-goal refinement apply to instances of the *goal* concept in RE. Goals describe environment conditions, whereby participants in RE desire that these conditions become true by the use of the system-to-be. Darimont and van Lamsweerde's [5] definition of a complete AND-refinement of a goal remains a standard one in RE. We recalled its definition earlier (cf., conditions

(C1)–(C4) in §I). Darimont and van Lamsweerde provide formal patterns for the AND-refinement of goals expressed in a first-order linear temporal Boolean logic, thereby facilitating the refinement process. This also keeps the AND-goal refinement relationship in line with the ideas of program refinement. To do goal refinement justice, we should mention first that the goal refinement we recalled above (§I) is what is usually called AND-goal refinement, and is available alongside OR-goal refinement: $g_1, \ldots, g_n$ OR-refine g if and only if: $\forall i \in \{1, \ldots, n\}$, (i) $g_i \vdash g$, and (ii) $g_i \nvdash \bot$. Second, we should mention that Letier [13] adds a domain theory, denoted *Dom*, and replaces $\vdash$ with the semantic entailment relation $\models$. Letier's AND-goal refinement indicates that $g_1, g_2, \ldots, g_n$ AND-refine g iff: (i) $g_1 \wedge g_2 \wedge \ldots \wedge g_n, \mathit{Dom} \models g$, (ii) $\forall i \in \{1, \ldots, n\}$, $\bigwedge_{j \neq i} g_j, \mathit{Dom} \not\models g$, (iii) $g_1 \wedge g_2 \wedge \ldots \wedge g_n, \mathit{Dom} \not\models \bot$, and (iv) $n > 1$; $g_1, g_2, \ldots, g_n$ OR-REFINE g iff $\forall i \in \{1, \ldots, n\}$, (i) $g_i, \mathit{Dom} \models g$, and (ii) $g_i, \mathit{Dom} \not\models \bot$. Letier does not clarify if $\models$ is standard or not; we assume by default that it is standard semantic entailment. As McCarthy [14] observes, standard semantic entailment $\models$ is monotonic. Recall that the monotonicity property tells us that if a sentence p follows from the collection of sentences X and $X \subset X'$, then p follows from X'. In the notation of proof theory: if $X \vdash p$ and $X \subset X'$, then $X' \vdash p$. We say that X entails p, and write $X \models p$ iff p is true in all models of X. But if $X \models p$ and $X \subset X'$, then every model of X' is also a model of X, which shows that $X' \models p$. It follows that the discussions in this paper remain equally relevant regardless of the choice of Darimont and van Lamsweerde's goal refinement defined via proof theory, or Letier's goal refinement defined via semantic entailment.

III. Two Paradoxes

This section presents the soundness paradox (§III-A) and the maturity paradox (§III-B), which arise when *goal refinement is used to refine early requirements, expressed in* NL. These paradoxes do *not* offer a critique of goal refinement *per se*; instead, they highlight that to use goal refinement when refining early requirements expressed in NL is in fact to *misuse* goal refinement. Avoiding the two paradoxes are the two critical desiderata for NL-refinement.

A. Soundness Paradox

The soundness paradox is rooted in the distinction between deductive and ampliative inference. Inference is ampliative when the inferred conclusion contains information absent from the premises (or data, or reasons) from which it was inferred; some examples of such reasoning are induction by enumeration, reasoning with analogies, and causal reasoning [21]. Recall that the benefit of goal refinement is to delay details to later steps of RE, then consider the following example, borrowed as-is from Darimont and van Lamsweerde [5]. Say that we have the following goal, which specifies a point-to-point communication between two agents (the operators $\diamond$ and $\Box$ read, respectively "eventually" and "always in the future", and have the semantics that they usually obtain in a linear temporal Boolean first-order logic):

$$\forall p_1, p_2 : \mathit{Point},\ m : \mathit{Message}, \\ \mathit{CallRequest}(p_1, p_2) \wedge (p_1.\mathit{msgOut} = m) \Rightarrow \diamond(p_2.\mathit{msgIn} = m)$$

This goal can be satisfied by requiring that both (i) a channel be allocated for the communication, and (ii) the communication uses the allocated channel. The conjunction of these two goals is an AND-goal refinement of the above goal. The first (i) of the two can be written as follows:

$$\forall p_1, p_2 : \mathit{Point},\ \mathit{Connection}(p_1, p_2) \\ \Rightarrow \Box(\exists c : \mathit{Channel})(\mathit{Allocation}(p_1, p_2, c))$$

Above, the notion of a channel is absent from the abstract (former) goal, but present in the less abstract (latter) goal. The question then is if this additional information was available in our "background" knowledge, and we simply chose not to make it explicit in the abstract goal (but did make it explicit in the less abstract goal), or is this additional information new – that is, we *added* it to our background knowledge when we were seeking less abstract goals for the refinement. In the former case, we reason by deduction when refining; the latter is instead a case of ampliative inference. Now, if refining a goal did in fact require deductive inference only, this would subsume some heroical assumptions, including (i) that we have all the necessary background knowledge when refining, and (ii) that it is feasible to make explicit all that background knowledge. Such assumptions conflict with the reality of RE, where:

> "clients may change their minds once they see the possibilities more clearly, and discoveries made during later phases may also force retrofitting requirements. The requirements of real systems are rarely static. There are very good reasons why clients often do not, or cannot, know exactly what they need; they may want to see models, explore alternatives, and envision new possibilities." [9]

It is then very often the case that inference applied during the early steps of RE, when requirements are expressed in NL, *is ampliative*. With this in mind, observe that the proof obligations (C1)–(C4) are defined in a *sound* logic, and we know that *if a logic is sound, it cannot lead by valid inference to conclusions that go beyond the contents of our background knowledge* [12]. In contrast, ampliative inference violates soundness, for it involves going beyond available knowledge in a deductive way. Hence the soundness paradox: *on one hand, when we perform ampliative inference to refine an* *abstract requirement, we violate soudness; on the other hand, the proof obligations we need to obey when we are refining are defined in a sound formalism*. This begs the question of why the constraints on the refinement relationship are not defined in a formalism that fits the fact that ampliative inference can be, and often quite obviously is applied when refining requirements in NL.

One could attempt to undermine the discussion above by saying that proof obligations are ideal rather than actual constraints that need to be satisfied, especially when requirements are early and expressed in NL. In such a perspective, proof obligations say what we aim for, not what we can/must achieve; they end up being not obligations, but a set of guidelines. It follows that we will not ask that proof obligations be satisfied, but some less demanding proxies of these constraints. This perspective does not alleviate the soundness paradox; it merely says that we can live with it; moreover, it suffers from two problems. The first is that goal refinement does not tell us how to check the less demanding proxies of the ideal constraints; in fact, it does not say what these proxies are in the first place! The second problem is that, while we can live with the soundness paradox, it is also true that rigor commands that this position becomes acceptable only after no solution that avoids the soundness paradox is found. This paper offers one such solution, thereby making the said position untenable. Another position can be to ignore the existence of the soundness paradox — to do so, one could introduce the following methodological twist: after the concrete requirements are found, consider first the concrete and abstract requirements, then check the proof obligations, and finally, modify any one or more of the concrete and/or abstract requirements to make sure that the proof obligations are not violated. The twist simply turns a blind eye to the violation of soundness when the concrete requirements are sought, but neither resolves, nor falsifies the soundness paradox.

B. Maturity Paradox

The maturity paradox stems from the opposition between the certain and indubitable character that a refinement obtains as soon as the proof obligations hold, and the tentative, preliminary, debatable character that a refinement is usually observed to actually have. The conclusion to draw from the maturity paradox is that we cannot reasonably follow proof obligations defined in a monotonic formalism when refining early requirements expressed in NL. Instead, we need obligations defined in a non-monotonic formalism.

Proof obligations (C1)–(C4) are defined in a formalism that enjoys the property of monotonicity. As noted above (cf., §II), this is the case in their definition via the notation of proof theory (as in [5]) and their definition via semantic entailment (as in [13]). By the definition of AND-goal refinement, the claim "*goals* $g_1, \ldots, g_n$ AND-*refine the goal* g" can be given if and only if all four proof obligations (C1)–(C4) verify. If (C1)–(C4) do verify, then the AND-refinement relationship in the tuple $(\{g_1, \ldots, g_n\}, \text{AND}, g)$ enjoys the property of monotonicity. This is because the proof obligations (C1)–(C3) require proofs, whereby the proofs are defined in a monotonic formalism. As Antonelli observes [1], proofs (as those in proof obligations) involve a kind of mathematical reasoning, which:

> "is characterized by a particular type of cogency: the conclusions are not merely probable or plausible on the basis of whatever evidential support the basic principles might provide, but certain and indubitable. In particular, mathematical reasoning enjoys a property referred to as monotonicity by modern logicians: if a conclusion follows from given premises A, B, C,... then it also follows from any larger set of premises, as long as the original premises A, B, C,... are included."

If the monotonic proof obligations hold, then the refinement relationship is *certain* and *indubitable*. By the virtue of being proved, and thereby certain and indubitable, the refinement relationship is *not* tentative, *not* based on partial, imprecise, ambiguous, or vague premises, and *cannot* be retracted when new facts are learned. This may well be true when we have goals expressed in a mathematical logic, because we tend to formalize at later steps of RE, when the purpose of the system-to-be is clear(er), (more) precise, and detailed (enough) to warrant the resources that formal methods require. But can this be true for early requirements expressed in NL? Strong arguments favor the negative answer: (i) it is clear that NL is notoriously prone to ambiguity, vagueness, and imprecision, among others (cf., e.g., [20]); (ii) at the outset of RE, the purpose of the system-to-be tends to be ill-defined and unclear (this is one of very basic reasons for doing RE in the first place); and (iii) requirements tend to change as participants in RE familiarize themselves with the problem at hand, and its potential solutions (cf., e.g., [9]). Requirements expressed in NL are *tentative*, *preliminary*, and *may be subject to revision*, and this by the very fact of being expressed in NL and early on in RE. They are *neither* certain, *nor* indubitable.

Consider an example. Suppose that the engineer needs to satisfy the goal "g: inform customers of price changes" for an e-commerce system. She can reason as follows:

> Usually, e-commerce systems inform a customer of price changes when the customer logs on to their profile, and sometimes by email.

This will lead the engineer to define two goals, "g_1: Show price changes in the user's profile" and "g_2: Send price changes to the user's email address". Since it is likely that she will deem reasonable enough to say that satisfying g_1 and g_2 satisfies g, she will define an AND-goal refinement $(\{g_1, g_2\}, \text{AND}, g)$. For this to be a goal refinement, proof obligations must verify. Hence, by claiming that it is a goal refinement, she subsumes that the proof obligations do verify. The next day, a stakeholder says that it is convenient to be informed of price changes via the mobile phone. This makes inadequate the refinement defined on the previous day. In response, the engineer will go back to the refinement defined before, and add a third goal, say "g_3: Send price changes via the short message service to the user's mobile phone, if the user chooses to receive price updates in this way". Just as she did on the previous day, she will assume that this is in fact a goal refinement. This pattern of doing refinement in RE is familiar and hardly to blame, for it matches commonsense reasoning, in which adding new information can lead to the change in prior conclusions.

Does the reasoning illustrated above fit the definition of goal refinement via proof obligations? It does not, because of the following contradiction. Because of monotonicity, any information that is added does not invalidate established proof. Above, observe that we initially have $g_1 \wedge g_2 \vdash g$; once we know g_3, we conclude that $g_1 \wedge g_2 \nvdash g$, but that $g_1 \wedge g_2 \wedge g_3 \vdash g$. This highlights a contradiction with regards to the second proof obligation (C2). The second proof obligation (cf., §I) indicates that the set of concrete goals contains only goals that are absolutely necessary: i.e., the refinement set is minimal. If we said that g_1 and g_2 AND-refine g, then the refinement set $\{g_1, g_2\}$ is minimal by definition of refinement. When we add g_3, we cannot conclude without abstracting from monotonicity that it is instead g_1, g_2, and g_3 that are the minimal AND-goal refinement of g.

It is apparent at this point that asking for monotonic proof obligations when refining early requirements expressed in NL negates what seems quite obvious: namely, that because requirements in NL are not certain and indubitable, the refinement of such requirements is at best tentative, preliminary, and open to revision. The obligations that the refinement of such requirements should instead obey must enjoy the property of non-monotonicity. Hence the maturity paradox: *on the one hand, we are asked to ensure that monotonic proof obligations hold; on the other hand, we know that the refinement relationship between early requirements expressed in* NL *is at best preliminary, tentative, and open to revision.*

Observe that, ultimately, the cause of the problem here is the maturity of the requirements, and the inherent characteristics (e.g., vagueness and ambiguity, among others) of the medium in which the requirements are expressed. It is clear that the potential for ambiguity, vagueness, and imprecision of early requirements in NL is strong, so that, when refining, we can at best "arrive to conclusions only tentatively, based on partial or incomplete information, reserving the right to retract those conclusions should they learn new facts" [1]. Refinement of early requirements is thus a case of reasoning, which produces conclusions that enjoy the property of nonmonotonicity.

IV. NL-Refinement

The soundness and maturity paradoxes (cf., §III) illustrate that the degree of rigor asked by proof obligations cannot be achieved when refining early requirements expressed in NL. Requirements engineers can presumably still be helpful and remain rigorous without pretending to a degree of stability, detail, precision, and completeness, which they apparently cannot deliver.

While it will be clear from the details below that NL-refinement does avoid the two paradoxes, it is relevant to briefly mention here how it does so. We define NL-refinement within a nonmonotonic formalism for the refinement of early requirements expressed in NL, in which there is no provability relation (i.e., logical consequence) $\vdash$, but instead a nonmonotonic consequence relation $\mid\!\sim$ (sometimes called nonmonotonic derivation, or simply derivation). Obligations on the refinement relationship are consequently defined via via nonmonotonic consequence, which makes refinement tentative, preliminary, and open to subsequent revision. This is closely related to allowing defaults in the formalism, which ensures that we avoid the soundness paradox. A default is an inference rule that relates prima facie reasons to their defeasible conclusion. Note the distinction between conclusive reasons and prima facie reasons: a set of sentences X are *conclusive reasons* for a sentence p only if we can deductively conclude p from X; members of X are *prima facie reasons* for p if we cannot deductively conclude p from X – instead, "prima facie reasons create a presumption in favor of their conclusion, but it is defeasible" [18]. Our formalism can thereby capture the application of ampliative inference by relating prima facie reasons to the conclusion defeasibly drawn from these reasons. Nonomonotonic derivation uses defaults to provide us with the implicit but tentative conclusions from a collection of sentences.

To define NL-refinement, we propose a formalism that will capture the early requirements expressed in NL, between which we are to determine refinement relationships. The formalism acts as a knowledge representation framework, in which a language is given to capture the information of interest, and reasoning procedures serve to verify obligations of NL-refinement. Inspiration came from what Dung [8] calls argument-based nonmonotonic reasoning in artificial intelligence, which originates in Pollock's contributions (cf., [17] and later). Simari and Loui's treatment of argument-based nomonotonic reasoning [19] has been a strong influence in the definition of the nonmonotonic consequence below; parallels will be obvious. No contributions are made in relation to the literature and lines of research on argumentation-based nonmonotonic reasoning in artificial intelligence; we are using established ideas and organizing them to respond to the present problem.

We go step by step below towards the definition of NL-refinement, within a nonmonotonic formal system for the representation of refinements and reasoning thereon. In terms of reasoning, we are interested in whether the obligations of a refinement are violated. We start with the details of the formal system (§IV-A), then define NL-refinement via the obligations, themselves written in the language (§IV-B). The section closes with the discussion of some basic reasoning about refinements (§IV-C).

A. Language

Overview. We build the formal system $\mathbb{A}$ in order to define NL-refinement using the facilities of $\mathbb{A}$. The language of $\mathbb{A}$ consists of a *propositional language* $\mathcal{L}$ and a binary relationship called *conclude* between well formed formulas of $\mathcal{L}$. Expressions in $\mathcal{L}$ are formed over propositions, whereby a proposition is (as usual) a shareable content of beliefs, desires, or intentions and the primary bearer of truth and falsity [15]. Propositions are assumed written in NL, and capture the information communicated or otherwise made available to the stakeholders and the engineer over the course of the early steps of RE, before it is relevant to rewrite some of these (potentially transformed) propositions in a predicate formalism. Sorts of propositions are distinguished to separate the propositions conveying relationships between other propositions, from the propositions they convey these relationships between. Of particular interest are the conclude relationship, and its variant, the *defeat* relationships. E.g., the proposition "the conjunction of propositions $p_1, \ldots, p_n$ is a prima facie reason to conclude p" is an instance of the conclude relationship, written $(\bigwedge_{i=1}^{n} p) \rightsquigarrow p$. The defeat relationship is introduced to mark that the satisfaction of some proposition(s) is prima facie reason for the failure of another proposition. E.g., the proposition "the conjunction of propositions $p_1, \ldots, p_n$ is a prima facie reason to conclude the failure of p" is an instance of the defeat relationship, and is written $(\bigwedge_{i=1}^{n} p) \dashrightarrow p$. We refer to $\rightsquigarrow$ and $\dashrightarrow$ as, respectively, the conclude and defeat connectives, and call a *defeasible rule* any expression having either the form $A \rightsquigarrow B$ or $X \dashrightarrow Y$. The language of $\mathbb{A}$ thus involves two kinds of expressions, defeasible rules and expressions written in $\mathcal{L}$. Given a set of expressions in $\mathcal{L}$ and a set of defeasible rules, we are interested in deriving expressions from those in the set and in determining which of them are satisfied. In a purely deductive formalism, the reasoner would establish as satisfied all theorems. In the case of defeasible reasoning, the conclusions of some of the premises may defeat other premises and/or conclusions, so that only some conclusions should be evaluated as satisfied. After discussing the syntax, we provide an account of derivation in $\mathbb{A}$, of the interpretation of syntactic elements in $\mathbb{A}$, and finally, of how to evaluate satisfaction.

Syntax. Let p, q, r, s, indexed or primed when needed, be symbols for propositions. (P, S) is the signature of the propositional language $\mathcal{L}$, where P and S are sets of, respectively, symbols for propositions and sorts. Each proposition $p \in P$ is sorted, so that P partitions onto sets of propositional symbols for each sort; if p is of sort σ, then $p \in P_\sigma$. We start with three sorts, the symbols of which are $S = \{\boldsymbol{i}, \boldsymbol{C}, \boldsymbol{D}\}$, and discuss the specialization of sorts later on. Expressions over the members of P are formed via standard propositional connectives: negation $\neg$, conjunction $\wedge$, disjunction $\vee$, and implication $\rightarrow$. $|S|$ unary operators are also used, one per sort in S, in order to abbreviate $p \in P_{\boldsymbol{i}}$ by $\boldsymbol{i}p$, and so on, for each sort. Defeasible rules are expressions formed by connecting two expressions of $\mathcal{L}$ via the conclude $\rightsquigarrow$ and defeat $\dashrightarrow$ connectives.

Definition IV.1. ***Symbolic Syntax.*** *For a given signature* (P, S)*, the set of well-formed formulas (wffs) is the smallest set of expressions built by applying the following rules, and only those, finitely many times:*

1) *Every* σp *with* $p \in P_\sigma$ *and* $\sigma \in S$ *is an* σ*-atom.*
2) *If* $\boldsymbol{i}p$ *is an* $\boldsymbol{i}$*-atom, then* $\neg \boldsymbol{i}p$ *is an* $\boldsymbol{i}$*-wff.*
3) *If* ψ *and* ϕ *are* $\boldsymbol{i}$*-wffs, then so are* $\psi \wedge \phi$, $\psi \vee \phi$, *and* $\psi \rightarrow \phi$.
4) *If* ψ *and* ϕ *are* $\boldsymbol{i}$*-wffs, then* $\psi \rightsquigarrow \phi$ *and* $\psi \dashrightarrow \phi$ *are defeasible rules.*
5) *Every* σ*-atom,* $\boldsymbol{i}$*-wff, and defeasible rule is a wff.*

Remark IV.2. $\neg \boldsymbol{i}p$ is equivalent to $\boldsymbol{i}(\neg p)$. $\{\neg, \wedge, \rightsquigarrow\}$ are basic connectives, the others are defined: $\psi \vee \phi$ abbreviates $\neg(\neg\psi \wedge \neg\phi)$, $\psi \rightarrow \phi$ abbrev. $\neg\psi \vee \phi$, and $\psi \dashrightarrow \phi$ abbrev. $\psi \rightsquigarrow \neg\phi$.

Remark IV.3. According to Definition IV.1, expressions with connectives in $\mathcal{L}$ are formed only over $\boldsymbol{i}$-atoms. Hence, $\boldsymbol{i}p \wedge \boldsymbol{i}q$ is a wff, but $\boldsymbol{D}r \wedge \boldsymbol{D}s$ is not. The reason why this is intended, is that the only purpose here for each $\boldsymbol{C}$- and $\boldsymbol{D}$-atom is to state the relationship between $\boldsymbol{i}$-propositions. We discuss this further below, in the presentation of the interpretation of wffs in $\mathbb{A}$.

Derivation. The question at present is what other formulas can be derived from a given set of formulas in $\mathbb{A}$.

Remark IV.4. Any axiomatization of $\mathcal{L}$ is acceptable in the rest of the discussion, as long modus ponens is attached to the axiomatization as a rule of inference.

Definition IV.5. ***Consequence.*** *Let* ϕ *be an* $\boldsymbol{i}$*-wff and* $\Gamma = \{\psi_i \mid 1 \leq i \leq n\}$ *a set, where each* ψ_i *is either an* $\boldsymbol{i}$*-wff or a defeasible rule in* $\mathbb{A}$. ϕ *is a consequence of* Γ*, written* $\Gamma \mid\!\sim \phi$ *if and only if:*

1) *there is a sequence* $\langle B_1, \ldots, B_m \rangle$*, called a derivation from* Γ*, such that* $\phi = B_m$ *and, for each* j*, either* B_j *is in* Γ*, or* B_j *is an axiom of* $\mathcal{L}$*, or* B_j *is a direct consequence of the preceding members of the sequence using modus ponens. Defeasible rules are regarded as material implications for the application of modus ponens; and*
2) *there is no derivation from* Γ *to* $\neg\phi$.

Example IV.6. Let $\Gamma = \{\boldsymbol{i}p_1, \boldsymbol{i}p_2, \boldsymbol{i}p_3, \boldsymbol{i}p_4, \boldsymbol{i}p_1 \wedge \boldsymbol{i}p_2 \rightsquigarrow \boldsymbol{i}p_5, \boldsymbol{i}p_3 \wedge \boldsymbol{i}p_4 \rightsquigarrow \boldsymbol{i}p_6, \boldsymbol{i}p_5 \wedge \boldsymbol{i}p_6 \rightsquigarrow \boldsymbol{i}p_7\}$. It is clear that $\Gamma \mid\!\sim \boldsymbol{i}p_7$, by the derivation $\langle \boldsymbol{i}p_1, \boldsymbol{i}p_2, \boldsymbol{i}p_5, \boldsymbol{i}p_3, \boldsymbol{i}p_4, \boldsymbol{i}p_6, \boldsymbol{i}p_7 \rangle$.

Proposition IV.7. *Consequence relation $\mid\!\sim$ is nonmonotonic.*

Proof: If we have $\{\boldsymbol{i}p\}$, then $\{\boldsymbol{i}p\} \mid\!\sim \boldsymbol{i}p$. If we add $\boldsymbol{i}q$ and $\boldsymbol{i}q \rightsquigarrow \neg \boldsymbol{i}p$ to $\{\boldsymbol{i}p\}$, then $\boldsymbol{i}p$ is not a consequence of $\{\boldsymbol{i}p\} \cup \{\boldsymbol{i}q, \boldsymbol{i}q \rightsquigarrow \neg \boldsymbol{i}p\}$. ■

Interpretation. Two particular considerations must be accounted for in defining the interpretation of a signature (P, S). The first concerns the idea that an $\boldsymbol{C}$- or $\boldsymbol{D}$-atom *conveys* conclude or defeat relationships between $\boldsymbol{i}$-atoms. The second focuses on the potential for partitioning the set of all $\boldsymbol{i}$-atoms, or equivalently, specializing the sort $\boldsymbol{i}$.

Since any one $\boldsymbol{C}$- or $\boldsymbol{D}$-atom conveys either the conclude or the defeat relationship between $\boldsymbol{i}$-atoms, $\mathbb{A}$ must incorporate facilities that relate the former propositions to the latter sets of propositions: we must make explicit that any wff involving the conclude and defeat connectives requires the presence of one/more $\boldsymbol{C}$- or $\boldsymbol{D}$-atoms. E.g., anytime we have a wff $\left(\bigwedge_{i=1}^{n-1} \psi_i\right) \rightsquigarrow \psi_n$, there must be an $\boldsymbol{C}p$ that conveys the said relationship between $\psi_1, \ldots, \psi_n$. We make this explicit below within the interpretation of a signature (cf., Definition IV.9).

We have argued throughout the preceding sections that early requirements expressed in NL cannot be refined via goal refinement, and have introduced sorts to distinguish, so to speak, the "relationship propositions" (i.e., $\boldsymbol{C}$- and $\boldsymbol{D}$-atoms) from $\boldsymbol{i}$-atoms. The question then is what propositions are admissible as $\boldsymbol{i}$-atoms? This depends entirely on the choice of a taxonomy of requirements, which will delimit the kinds of information we are interested in. Suppose that we choose to use with $\mathbb{A}$ some taxonomy $\mathfrak{T}$ of requirements, where $\mathfrak{T}$ is represented as a tree, in which the nodes symbolize the concepts of the taxonomy and the links are instances of the standard *is-a* relationship. If $N(\mathfrak{T})$ is the set of all nodes of $\mathfrak{T}$, then our interpretation of a signature (P, S) must partition $P_{\boldsymbol{i}}$ onto sets of symbols for propositions that instantiate each of the concepts in $N(\mathfrak{T})$ – i.e., $P_{\boldsymbol{i}}$ will have $|N(\mathfrak{T})|$ partitions. E.g., if a taxonomy says that any requirement is either a goal or a task, then $P_{\boldsymbol{i}} = P_{goal} \cup P_{task}$, and an interpretation will map each goal $p \in P_{goal}$ or task proposition symbol $p \in P_{task}$ to, respectively, a goal or task proposition. For the sake of generality, we do not choose at this point a specific taxonomy of requirements, but simply acknowledge that some taxonomy $\mathfrak{T}$ is used together with $\mathbb{A}$.

Remark IV.8. In Definition IV.9, we assume the following: (a) $\mathfrak{T}$ a taxonomy of requirements; (b) $N(\mathfrak{T})$ the set of symbols, one for each node in $\mathfrak{T}$; and (c) $\mathcal{C}(\boldsymbol{c} \in N(\mathfrak{T}))$ a function that returns the concept in $\mathfrak{T}$ that is symbolized by a given member of $N(\mathfrak{T})$. Since there is a taxonomy $\mathfrak{T}$ of requirements, we replace $\boldsymbol{i}$ in S with the members of $N(\mathfrak{T})$ (i.e., the symbols for the various kinds of requirements), and have $S = \{\boldsymbol{C}, \boldsymbol{D}\} \cup N(\mathfrak{T})$ in a signature (P, S).

Definition IV.9. ***Interpretation.*** *An interpretation $\mathcal{I}$ of a signature (P, S) in $\mathbb{A}$ is a function satisfying only all following properties:*

1) *For each sort $\sigma \in S$, $\mathcal{I}(P_\sigma)$ is a set of propositions.*
2) *For each $\boldsymbol{c} \in N(\mathfrak{T})$ and $p \in P_{\boldsymbol{c}}$, $\mathcal{I}(\boldsymbol{c}p)$ is an instance of the concept $\mathcal{C}(\boldsymbol{c})$.*
3) *For each $p \in P_{\boldsymbol{C}}$, $\mathcal{I}(\boldsymbol{C}p)$ is an instance of the* Conclude *relationship, whereby an instance of the conclude relationship conveys that an $\mathcal{I}(\psi)$ is prima facie reason for $\mathcal{I}(\phi)$, where ψ and ϕ are $\boldsymbol{i}$-wffs.*
4) *For each $p \in P_{\boldsymbol{D}}$, $\mathcal{I}(\boldsymbol{D}p)$ is an instance of the* Defeat *relationship, whereby an instance of the defeat relationship conveys that an $\mathcal{I}(\psi)$ is prima facie reason for $\mathcal{I}(\neg\phi)$, where ψ and $\neg\phi$ are $\boldsymbol{i}$-wffs.*

Remark IV.10. According to Definition IV.9, the codomain of $\mathcal{I}$ consists of instances of all concepts in the requirements taxonomy $\mathfrak{T}$, and the instances of the conclude and defeat relationship.

Evaluation. The participation of potentially many stakeholders and engineers to the early phase of RE can usually be expected. Each will offer information, which will be symbolized by σ-atoms and wffs in $\mathbb{A}$. Assume consequently m participants, or sources of information. Each participant k volunteers the information I_k, where I_k is a (potentially inconsistent) set of σ-atoms and $\boldsymbol{i}$-wffs. We assume for simplicity that there is a single set Δ of defeasible rules, and that all m participants agree on it.[1] Given (I_k, Δ), the k-th participant can voice arguments. Given a set of arguments from different participants, the aim of evaluation is to tell us if a particular piece of information (an $\boldsymbol{i}$-wff) is satisfied or denied. In intuitive terms, evaluation tells us what the participants agree on. We assume below that $I = \bigcup_{k=1}^{m} I_k$.

Definition IV.11. ***Argument.*** *Given a set of $\boldsymbol{i}$-wffs I and a set Δ of defeasible rules, (A, D), $A \subseteq I$, $D \subseteq \Delta$ is an argument for an $\boldsymbol{i}$-wff ϕ, denoted $\langle A, D, \phi \rangle$, if and only if:*

1) *(Consequence:) $A \cup D \mid\!\sim \phi$,*
2) *(Consistency:) $A \cup D \not\mid\!\sim \bot$, and*
3) *(Minimality:) $\nexists A' \subset A$, $A' \cup D \mid\!\sim \phi$ and $\nexists D' \subset D$, $A \cup D' \mid\!\sim \phi$.*

Definition IV.12. ***Attack.*** *An argument $\langle A_1, D_1, \phi_1 \rangle$ attacks another argument $\langle A_2, D_2, \phi_2 \rangle$ if and only if $A_1 \cup A_2, D_1 \cup D_2 \mid\!\sim \bot$.*

Definition IV.13. ***Support.*** *An argument $\langle A_1, D_1, \phi_1 \rangle$ supports another argument $\langle A_2, D_2, \phi_2 \rangle$ if and only if $A_1, D_1 \mid\!\sim \psi$, such that $\psi \in A_2$.*

Example IV.14. Consider $\langle \{\boldsymbol{i}p_1 \wedge \boldsymbol{i}p_2, \neg\boldsymbol{i}p_3\}, \{\boldsymbol{i}p_1 \wedge \boldsymbol{i}p_2 \dashrightarrow \boldsymbol{i}p_3, \neg\boldsymbol{i}p_3 \rightsquigarrow \boldsymbol{i}p\}, \boldsymbol{i}p \rangle$ and $\langle \{\boldsymbol{i}p_3\}, \{\boldsymbol{i}p_3 \rightsquigarrow \boldsymbol{i}p_4\}, \boldsymbol{i}p_4 \rangle$. The latter is an argument; the former is not, for it is not

[1]Releasing this assumption is an important topic for future work, allowing thereby updates to the set Δ, which certainly can happen over the course of early RE.

minimal. The former can be rewritten as two arguments: $\langle\{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2\}, \{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2 \dashrightarrow \boldsymbol{ip}_3\}, \neg\boldsymbol{ip}_3\rangle$ and $\langle\{\neg\boldsymbol{ip}_3\}, \{\neg\boldsymbol{ip}_3 \rightsquigarrow \boldsymbol{ip}\}, \boldsymbol{ip}\rangle$. $\langle\{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2\}, \{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2 \dashrightarrow \boldsymbol{ip}_3\}, \neg\boldsymbol{ip}_3\rangle$ attacks $\langle\{\boldsymbol{ip}_3\}, \{\boldsymbol{ip}_3 \rightsquigarrow \boldsymbol{ip}_4\}, \boldsymbol{ip}_4\rangle$ via $\boldsymbol{ip}_3$. The argument $\langle\{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2\}, \{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2 \dashrightarrow \boldsymbol{ip}_3\}, \neg\boldsymbol{ip}_3\rangle$ supports the argument $\langle\{\neg\boldsymbol{ip}_3\}, \{\neg\boldsymbol{ip}_3 \rightsquigarrow \boldsymbol{ip}\}, \boldsymbol{ip}\rangle$.

Remark IV.15. If $\langle A_1, D_1, \phi_1\rangle$ supports $\langle A_2, D_2, \phi_2\rangle$, we call the former the *supporter*; if $\langle A_1, D_1, \phi_1\rangle$ attacks $\langle A_2, D_2, \phi_2\rangle$, then we call the former *attacker*.

Given a set of arguments, we can chain them via attack and support relationships, and then evaluate the $\boldsymbol{i}$-wffs in all chained arguments. To evaluate whether an argument is satisfied (equivalently, acceptable or justified), we consider if all of its supporting arguments are themselves satisfied, and if all of its attacking arguments are all themselves attacked. To provide the sufficient conditions for the satisfaction of an argument in a given set $\mathbf{A}$ of arguments, we need an argument network for $\mathbf{A}$.

Definition IV.16. ***Argument Network.*** *Let $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$ be a directed labeled and connected graph, and $\boldsymbol{A}$ a set of arguments. $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$ is called an argument network for $\boldsymbol{A}$ if and only if it has the following properties:*

1) *$\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$ is acyclic;*
2) *no two nodes in $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$ are connected by more than one line;*
3) *$N(\boldsymbol{G}) \equiv \boldsymbol{A}$, i.e., arguments are nodes;*
4) *for any pair of distinct arguments $(\boldsymbol{a}_i, \boldsymbol{a}_j)$, $\boldsymbol{a}_i \neq \boldsymbol{a}_j$ in $\boldsymbol{A}$:*
    a) *$\exists \boldsymbol{a}_i\boldsymbol{a}_j \in L(\boldsymbol{G})$ labeled A only if $\boldsymbol{a}_i$ attacks $\boldsymbol{a}_j$;*
    b) *$\exists \boldsymbol{a}_i\boldsymbol{a}_j \in L(\boldsymbol{G})$ labeled S only if $\boldsymbol{a}_i$ supports $\boldsymbol{a}_j$.*

Remark IV.17. In Definition IV.16, $N(\mathbf{G})$ and $L(\mathbf{G})$ denote, respectively, the set of nodes and lines of the graph $\mathbf{G}(N(\mathbf{G}), L(\mathbf{G}))$. A line directed from a node $\mathbf{a}_i$ to a node $\mathbf{a}_j$ is written $\mathbf{a}_i\mathbf{a}_j$.

Remark IV.18. The first condition in Definition IV.16 makes the evaluation procedure below not applicable to cyclical argument networks. This limitation either demands the preprocessing of a cyclical argument network, to eliminate cycles prior to evaluation, or evaluation conditions robust w.r.t. cycles. Pollock recently discussed the latter [16].

Definition IV.19. ***Labeling Procedure.*** *Nodes of a connected argument network $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$ are recursively labeled as follows:*

1) *Leaves of $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$ are labeled* U.
2) *If $\boldsymbol{a}$ is an inner node of $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$, then $\boldsymbol{a}$ is labeled* U *only if each of its attackers is labeled* D *and each of its supporters is labeled* U.

Definition IV.20. ***Justified Argument.*** *Given a labeled argument network $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$, an argument $\boldsymbol{a} \in N(\boldsymbol{G})$ is justified if and only if it is labeled* U *in that labeled argument network.*

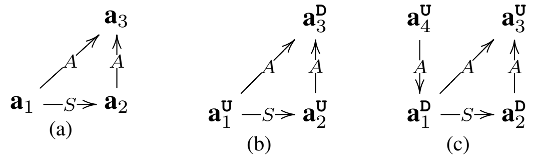

Figure 1. Example IV.21.

Example IV.21. Following Example IV.14, let $\mathbf{a}_1 = \langle\{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2\}, \{\boldsymbol{ip}_1 \wedge \boldsymbol{ip}_2 \dashrightarrow \boldsymbol{ip}_3\}, \neg\boldsymbol{ip}_3\rangle$, $\mathbf{a}_2 = \langle\{\neg\boldsymbol{ip}_3\}, \{\neg\boldsymbol{ip}_3 \rightsquigarrow \boldsymbol{ip}\}, \boldsymbol{ip}\rangle$, and $\mathbf{a}_3 = \langle\{\boldsymbol{ip}_3\}, \{\boldsymbol{ip}_3 \rightsquigarrow \boldsymbol{ip}_4\}, \boldsymbol{ip}_4\rangle$. Figures 1(a) and 1(b) illustrate the argument network from these three arguments, resectively, without labels, and with labels. In Figure 1(b), $\mathbf{a}_1$ and $\mathbf{a}_2$ are both labeled U, and $\mathbf{a}_3$ is labeled D, so that $\mathbf{a}_1$ and $\mathbf{a}_2$ are justified, and $\mathbf{a}_3$ is not justified. Figure 1(c) gives a labeled argument network obtained after adding an argument $\mathbf{a}_4$, which attacks $\mathbf{a}_1$.

Proposition IV.22. *Given a finite and connected argument network, each node has either the label* U *or* D*, and the labeling procedure (cf., Definition IV.19) will find it.*

Proof: By Definition IV.16, any argument network is acyclic. Moreover, Proposition IV.22 assumes the argument network to be finite and connected. The first rule in the labeling procedure assigns U to all leaves. Starting from each leaf, an ancestor node is considered, and is labeled according to the labels on its children and the labels A or S on lines from its children. The labeling stops when the nodes without ancestors are reached and labeled. Since the argument network is finite and acyclic, there is a finite number of (simple) paths from each leaf to each node without ancestors. It is apparent that no inner node will remain unlabeled. To see why, suppose that there is an inner node, and remains unlabeled. This is only possible if there is no path from a leaf to that inner node, which is a contradiction, since it is assumed an inner node. ∎

B. Obligations

Definition IV.23. **NL*-refinement.*** *Given a potentially inconsistent set I of $\boldsymbol{i}$-wffs and a set Δ of defeasible rules, let $\boldsymbol{A}$ be the maximal set of arguments from I and Δ, and $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$ an argument network with $N(\boldsymbol{G}) \equiv \boldsymbol{A}$. We say that $I' \subseteq I$* NL*-refines ψ if and only if:*

1) *$\langle I', \Delta', \psi\rangle$ is an argument, and $\langle I', \Delta', \psi\rangle \in \boldsymbol{A}$, and*
2) *$\langle I', \Delta', \psi\rangle$ is justified in the labeled $\boldsymbol{G}(N(\boldsymbol{G}), L(\boldsymbol{G}))$.*

Definition IV.24. ***Maximal Set of Arguments.*** *$\boldsymbol{A}$ is the maximal set of arguments from a potentially inconsistent set I of $\boldsymbol{i}$-wffs and a set Δ of defeasible rules if and only if there is no argument $\langle I', \Delta', \phi\rangle \notin \boldsymbol{A}$ such that $I' \subseteq I$ or $\Delta' \subseteq \Delta$.*

To contrast the definition above to goal refinement, recall that the obligations for AND-goal refinement are (cf., §I) logical consequence (C1), minimality (C2), consistency (C3), and (C4), which we shall call non-triviality. Within $\mathbb{A}$, the first condition in Definition IV.23 indicates that nonmonotonic consequence, minimality, and consistency obligations in NL-refinement. Intuitively, they can be seen as analogues to the conditions (C1)–(C3) in NL-refinement. There is no obligation in NL-refinement that is inspired or corresponds to non-triviality, which was introduced in AND-goal refinement "to avoid trivial refinements consisting in rewriting [a goal] into logically equivalent forms" [5]. The presumable intention behind the non-triviality constraint in AND-goal refinement is to avoid the use of different syntactical representations of the same relationships in the application domain. Roughly speaking, the aim is to ban the refinement that amounts to the simple rewriting of the same information. Non-triviality plays out in a rather different manner in NL-refinement. Consider an example: suppose that in the argument $\langle\{\boldsymbol{ip}\},\{\boldsymbol{ip} \rightsquigarrow \boldsymbol{iq}\},\boldsymbol{iq}\rangle$, $\boldsymbol{ip}$ is the proposition "Inform the customer of price changes via email" and $\boldsymbol{iq}$ is the proposition "Inform the customer of price changes". If this argument turns out to be justified within some given argument network, then we shall conclude that $\boldsymbol{ip}$ NL-refines $\boldsymbol{iq}$. Now, the non-triviality constraint in AND-goal refinement is a syntactic one: if we imposed it on the argument $\langle\{\boldsymbol{ip}\},\{\boldsymbol{ip} \rightsquigarrow \boldsymbol{iq}\},\boldsymbol{iq}\rangle$, $\boldsymbol{ip}$, then that argument would not be a refinement, because $\{\boldsymbol{ip}\}$ is a singleton and the non-triviality constraint asks that at least one additional piece of information be added to the set $\{\boldsymbol{ip}\}$. In contrast, we see that $\boldsymbol{ip}$ in fact is more detailed than $\boldsymbol{iq}$, and we may well be content with that refinement as it is. That we make no use of the syntactic non-triviality constraint does not mean that we have no way to avoid trivial refinements given by rewriting propositions. Suppose that we have the argument $\langle\{\boldsymbol{ip}'\},\{\boldsymbol{ip}' \rightsquigarrow \boldsymbol{iq}'\},\boldsymbol{iq}'\rangle$, so that $\boldsymbol{ip}'$ is the proposition "The customer should always be informed of price changes" and $\boldsymbol{iq}'$ is the proposition "Always inform the customer of price changes". Disregarding grammatical nuances leads to the conclusion that $\boldsymbol{ip}'$ is a simple rewriting of $\boldsymbol{iq}'$. If the argument $\langle\{\boldsymbol{ip}\},\{\boldsymbol{ip} \rightsquigarrow \boldsymbol{iq}\},\boldsymbol{iq}\rangle$ is a refinement in a given argument network, we can respond by adding another argument $\langle\{\boldsymbol{ir}\},\{\boldsymbol{ir} \dashrightarrow \boldsymbol{ip}\},\neg\boldsymbol{ip}\rangle$ to the same argument network. If the added argument turns out justified in the revised argument network, then former will not be justified, and will not be a refinement. In conclusion, the syntactic non-triviality constraint is replaced here with a, so to speak, pragmatic non-triviality constraint: we can voice arguments against a refinement via the appropriate update of the argument network, which can result in the failure of the trivial argument to be a refinement.

C. Reasoning

The question we are interested in is which arguments are justified and therefore refinements in a given argument network. The EVALUATE procedure in Algorithm 1 is applied to an argument network in order to label each argument with either U or D. EVALUATE implements the labeling procedure from Definition IV.19. The traversal of an argument network and the computation of the labels is straightforward: EVALUATE launches a modified (to accommodate labeling) breadth-first search (e.g., [11]) from each leaf node, and at each node applies the appropriate labeling rule from Definition IV.19.

Proposition IV.25. *When applied to a connected and finite argument network $\boldsymbol{G}(N(\boldsymbol{G}),L(\boldsymbol{G}))$, the procedure* EVALUATE*: (i) does not loop indefinetly; (ii) assigns labels according to the rules in Definition IV.19; and (iii) has the upper bound on the running time in $O(P(|N(\boldsymbol{G})|+|L(\boldsymbol{G})|))$, where P is the number of simple paths in $N(\boldsymbol{G})$ such that each of these paths starts in a leaf node of $\boldsymbol{G}(N(\boldsymbol{G}),L(\boldsymbol{G}))$.*

Proof: (i) Because $\mathbf{G}(V(\mathbf{G}),L(\mathbf{G}))$ is acyclic, the number of times a node will be traversed by the outer **for each** loop equals the number of simple paths from each of its leaf nodes. The number of simple paths in a finite acyclic directed graph is finite, so that each node will be traversed a finite number of times. At every traversal of a node, that node is removed from Q, so that Q will ultimately empty and the outer **for each** loop will terminate. EVALUATE will never loop indefinetly for an acyclic and finite $\mathbf{G}(N(\mathbf{G}),L(\mathbf{G}))$. (ii) Lines 5–7 correspond to the first rule in Definition IV.19. Lines 9–17 correspond to the second rule in Definition IV.19. Each time an inner node is visited, the conditional block in lines 9–17 is called, and the label on the visited node is replaced. Now, observe that it is only the last time an inner node is visited that all of the children will be labeled. That last visit will provide the definite label on the inner node. It is straightforward then that assigns labels according to the rules in Definition IV.19. (iii) The number of times a node v will enter the queue Q equals the total number of simple paths to v from all leaf nodes in $N(\mathbf{G})$. The upper bound on the running time is then in $O(P(|N(\mathbf{G})|+|L(\mathbf{G})|))$, where P is the number of simple paths in $V(N(\mathbf{G}))$ such that each of these paths starts in a leaf node of $\mathbf{G}(N(\mathbf{G}),L(\mathbf{G}))$. ■

Example IV.26. Consider the problem of defining a registration form for new users of an online flight booking application. Say that we are given the following statements:

$\boldsymbol{iq}$: Whenever an unregistered user selects to purchase a flight, a comprehensive registration form should be displayed.

From there on, suppose that we have the following statements:

$\boldsymbol{ip}_1$: A user is not registered.

$\boldsymbol{ip}_2$: A user selects to purchase a flight.

$\boldsymbol{ip}_3$: A comprehensive registration form is displayed.

$\boldsymbol{Cr}$: Satisfying together the conditions in $\boldsymbol{ip}_1$, $\boldsymbol{ip}_2$, and $\boldsymbol{ip}_3$ is prima facie reason for the satisfaction of the condition in $\boldsymbol{iq}$.

The statements above lead us to the argument $\mathbf{a}_1 = \langle\{\bigwedge_{i=1}^{3}\boldsymbol{ip}_i\},\ \{\bigwedge_{i=1}^{3}\boldsymbol{ip}_i \rightsquigarrow \boldsymbol{iq}\},\boldsymbol{iq}\rangle$. If we are not content with the formulations of $\boldsymbol{ip}_3$, because "comprehensive" is a vague term, we can suggest the argument

Algorithm 1 Evaluation

```
Input: An argument network $\mathbf{G}(N(\mathbf{G}), L(\mathbf{G}))$;
Output: For each node in $N(\mathbf{G})$, either the label U or D;
 1: procedure EVALUATE($\mathbf{G}(N(\mathbf{G}), L(\mathbf{G}))$)
 2:     Empty the queue $Q$
 3:     Add all leaf nodes in $N(\mathbf{G})$ to the queue $Q$
 4:     for each node $v$ in $Q$ do
 5:         if $v$ has no offspring then
 6:             Replace the label on $v$ with U
 7:         else
 8:             for each $w \in N(\mathbf{G})$ s.t. $\exists vw \in L(\mathbf{G})$ do
 9:                 if $w$ is an attacker and is labeled U then
10:                     Replace the label on $v$ with D
11:                 else if $w$ is an attacker and is labeled D then
12:                     Replace the label on $v$ with U
13:                 else if $w$ is a supporter and is labeled U then
14:                     Replace the label on $v$ with U
15:                 else if $w$ is a supporter and is labeled D then
16:                     Replace the label on $v$ with D
17:                 end if
18:                 Add $w$ to $Q$
19:             end for
20:         end if
21:         Delete $v$ from $Q$
22:     end for
23: end procedure
```

$$\begin{tikzcd}\mathbf{a}_2 \arrow[r, dashed, "S"] & \mathbf{a}_1 \\ \mathbf{a}_3 \arrow[u, dashed, "A"] & \mathbf{a}_4 \arrow[l, dashed, "A"']\end{tikzcd}$$

(a)

$$\begin{tikzcd}\mathbf{a}_2^{\mathtt{U}} \arrow[r, dashed, "S"] & \mathbf{a}_1^{\mathtt{U}} \\ \mathbf{a}_3^{\mathtt{D}} \arrow[u, dashed, "A"] & \mathbf{a}_4^{\mathtt{U}} \arrow[l, dashed, "A"']\end{tikzcd}$$

(b)

Figure 2. Example IV.26.

$\mathbf{a}_2 = \langle\{\bigwedge_{i=4}^{15} \boldsymbol{ip}_i\}, \{\bigwedge_{i=5}^{15} \boldsymbol{ip}_i \rightsquigarrow \boldsymbol{ip}_3\}, \boldsymbol{ip}_3\rangle$, where each $\boldsymbol{ip}_5, \ldots, \boldsymbol{ip}_{15}$ indicates a single piece of information that the form should ask the user to fill in. Suppose that among them, we have the following:

$\boldsymbol{ip}_9$: The user should provide her birth date.

We may have another argument, $\mathbf{a}_3 = \langle\{\boldsymbol{is}\}, \{\boldsymbol{is} \dashrightarrow \boldsymbol{ip}_9\}, \boldsymbol{ip}_9\rangle$, in which $srtis$ is:

$\boldsymbol{is}$: Information that is not absolutely necessary when purchasing a plane ticket should not be asked in the registration form.

If we know, however, that there a regulation demands that no minor can be a registered user, we can further offer the argument $\mathbf{a}_4 = \langle\{\boldsymbol{ip}_{16}\}, \{\boldsymbol{ip}_{16} \dashrightarrow \boldsymbol{is}\}, \neg\boldsymbol{is}\rangle$, where $\boldsymbol{ip}_{16}$ is:

$\boldsymbol{ip}_{16}$: The age of the user must be known in order to comply with the regulation that no minors can register.

At this point, we have four arguments, for which the argument network is shown in Figure 2(a). Applying the EVALUATE procedure gives the labels shown in Figure 2(b). We see that both $\mathbf{a}_1$ and $\mathbf{a}_2$ are refinements.

V. CONCLUSIONS

Stakeholders in an RE process communicate their requirements in natural language. It is a longstanding conceptual and methodological challenge to systematically increase the precision of early requirements expressed in NL to the point where they can be straightforwardly rewritten in a predicate formalism. Although Darimont and van Lamsweerde's goal refinement [5] was initially defined within a (monotonic) linear temporal Boolean first-order logic via a series of proof obligations, we have argued that many are misusing it by applying it to early requirements expressed in NL. One typically sees propositions designating desired conditions being represented as nodes in an AND/OR goal graph, in which links indicate the instances of AND- or OR-goal refinement. By drawing such graphs, one subsumes that proof obligations of goal refinement verify. We have argued (cf., §III) that this assumption is not acceptable when refining requirements expressed in NL, because doing so suffers from the soundness and maturity paradoxes.

Our response in this paper is a proposal for a new kind of refinement, called NL-refinement, specifically designed for the refinement of early requirements expressed in NL. Its salient point is that its obligations are defined to avoid the soundness and maturity paradoxes. Obligations in NL-refinement are in a sense "weaker" than those of goal refinement: an instance of the refinement relationship is not proved to hold in a monotonic formal system, and is therefore not certain and indubitable, but tentative, preliminary, and open to revision. Whether a refinement relationship stands between some information must be verified again as new information becomes available. This is reflected in our use of a nonmonotonic formal system when defining the obligations of NL-refinement, in contrast to defining obligations in a monotonic formalism, as in goal refinement.

In his discussion of program development by stepwise refinement, Wirth argued that "each refinement implies a number of design decisions based upon a set of design criteria. [...] Students must be taught to be conscious of the involved decisions and to critically examine and to reject solutions [...]. In particular, they must be taught to revoke earlier decisions, and to back up, if necessary to the top. Relatively short sample problems will often suffice to illustrate this important point; it is not necessary to construct an operating system for this purpose." ([22]; p.227) NL-refinement reflects these principles via its proof obligations and extends them to the refinement of early requirements expressed in NL.

REFERENCES

[1] A. Antonelli. Non-monotonic logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2008.

[2] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the purse: The balance enquiry quandary, and generalised and (1, 1) forward refinements. *Fundam. Inform.*, 77(1-2):29–69, 2007.

[3] R. Banach and M. Poppleton. Retrenchment: An engineering variation on refinement. In *B'98: Int. B Conf.*, pages 129–147, 1998.

[4] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[5] R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *SIGSOFT FSE*, pages 179–190, 1996.

[6] W. P. de Roever and K. Engelhardt. *Data Refinement: Model-oriented Proof Theories and their Comparison*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[7] E. W. Dijkstra. Chapter I: Notes on structured programming. In *Structured programming*, pages 1–82. Academic Press Ltd., 1972.

[8] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[9] J. A. Goguen and C. Linde. Techniques for requirements elicitation. In *Proc. Int. Symp. Req. Eng.*, pages 152–164, 1993.

[10] C. A. R. Hoare. Proof of correctness of data representations. *Acta Inf.*, 1:271–281, 1972.

[11] D. E. Knuth. *The Art Of Computer Programming*, volume 1. Boston: Addison-Wesley, 3rd edition, 1997.

[12] H. E. Kyburg, Jr. Real logic is nonmonotonic. *Minds and Machines*, 11(4):577–595, 2001.

[13] E. Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Université catholique de Louvain, 2001.

[14] J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artif. Intell.*, 13(1-2):27–39, 1980.

[15] M. McGrath. Propositions. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2008.

[16] J. Pollock. A recursive semantics for defeasible reasoning. In I. Rahwan and G. Simari, editors, *Argumentation in Artificial Intelligence*. Springer, 2009.

[17] J. L. Pollock. Defeasible reasoning. *Cognitive Science*, 11(4):481–518, 1987.

[18] J. L. Pollock. Justification and defeat. *Artif. Intell.*, 67(2):377–407, 1994.

[19] G. R. Simari and R. P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artif. Intell.*, 53(2-3):125–157, 1992.

[20] R. Sorensen. Vagueness. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2008.

[21] C. Swoyer. Relativism. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2008.

[22] N. Wirth. Program development by stepwise refinement. *Commun. ACM*, 14(4):221–227, 1971.