Abstraction, Refinement, Enrichment

Stefan Gruner

Department of Computer Science
University of Pretoria
South Africa
E-mail: sg@cs.up.ac.za

Abstract: In the no longer existing South African journal Quæstiones Informaticæ, *An Approach to Defining Abstractions, Refinements and Enrichments* was published by Derrick Kourie more than twenty years ago. At some occasion, about two years ago, Derrick Kourie had asked and encouraged me to review his original topics, such as to re-construct and re-present them from a different perspective. In this festschrift chapter, in honour of Derrick Kourie's 65th birthday, I outline the results of my attempt at fulfilling Derrick Kourie's collegial request. For this purpose I shall first recapitulate the key concepts of Kourie's original paper, since that paper has more or less fallen into oblivion and cannot be easily retrieved from the public domain any more. Thereafter Kourie's notions of abstraction, refinement and enrichment are re-defined in a different (more classical) theoretical framework. Finally those notions are contextualised with respect to the related notion of retrenchment developed since the mid-1990s by Banach, Poppleton, et al.

1 Motivation and History

Notions of abstraction, refinement and enrichment are relevant for many practical activities in software engineering, including (for example): super-classes in object-oriented programming (abstraction), model-driven software development from specifications to implementations (refinement), or the definition and construction of software product lines as families of related software systems (enrichment).

In a 1989-article published in the no longer existing journal Quæstiones Informaticæ, Derrick Kourie had noticed the practical relevance of those above-mentioned notions as well

as lack of a unifying theory to bring those tightly related notions under one consistent and coherent theoretical roof [14]. In that paper Kourie had presented his early ideas about such a theoretical framework, which was then based on first-order predicate logic together with a tuple-structured universe (extension) and a non-standard *trivalent* truth function in $\{\mathsf{T}, \mathsf{F}, \bot\}$.

In the early approach of [14], however, important background notions (such as the pragmatic notion of relevance) were presented with much appeal to intuition, and the notion of abstraction was overloaded with ambiguity insofar as Kourie had spoken in [14] about abstract *entities* as well as about abstract *properties*. Also the crucial notion of enrichment was only with much appeal to intuition. Last but not least the use of a *trivalent* truth function in [14] was not elaborated in detail —what was the role and purpose of the undefined truth value $\bot$ in that theoretical framework?— and the structure of a specific, particular universe of discourse —containing specifically *tuples* instead of simply anything— as the underlying semantic domain was also not compellingly motivated in [14].

With those or similar considerations in mind, Kourie had occasionally (about two years ago) asked me to revisit his original article, and to present a reconstruction of his original ideas from a different theoretical perspective. The remainder of this Festschrift chapter in honour of Derrick Kourie's 65^{th} birthday describes my attempt at fulfilling his collegial request. For this purpose I shall in the following section recapitulate the main concepts of Kourie's original paper because that paper had more or less fallen into oblivion and cannot be easily retrieved any more. In the subsequent sections I outline the notions of abstraction, refinement and enrichment in a classical-logical manner and bring them into context with the related, more recent, notion of *retrenchment* specifically developped for the domain of software engineering by Banach, Poppleton, et al.

2 Recapitulation of Kourie's 1989-Paper

Paper [14] was motivated by Kourie's often-expressed desire for theoretical consolidation of hitherto unrelated concepts and notions in the fields of computer science and software engineering: *"It would seem, therefore, that there is a need to develop a unified formal theory of abstractions and refinements so as to sharpen and shape the ideas of those who are involved in software development and in the practice of AI. Towards this end, the present paper proposes generic definitions for these concepts, and for several concepts directly associated with these. In particular, the notion of enrichment seems to be long naturally with that of refinement"* [14](p.174).

2.1 Preliminaries

The elaboration of [14] was based on first-order predicate logic together with a set-theoretic interpretation, in which particularly the union of sets, $X \cup Y$, was relevant. "$\vdash P$" was explicated as *"Predicate P is true"*, and "$P \rightarrow Q$" was explicated as *"Predicate Q logically*

follows from predicate P" [14] without particular differentiation between a syntactic versus a semantic notion of logical implication or consequence.

Notation and treatise in [14] were, however, only semi-formal and appealed to its reader's intuition at several places. Such semi-formality of expressions does not only make it hard to grasp the subtleties of the ideas and intentions 'behind' the original paper [14] in its full length: it makes it even harder to *reproduce* those ideas adequately and understandably in this Festschrift chapter in which Kourie's original paper cannot be reproduced as a whole, line by line, for obvious reasons.

For these reasons, also one of the reviewers of the draft of this chapter (before its publication) had a number of subtle questions concerning some finer theoretical details of this elaboration. Not all of questions I was able to answer fully, because I would not have been able to 'ground' such answers solidly in the positively given historical material of [14] as it stands. This is a hermeneutical problem well known to any historian working on the basis of texts in the faculties of the humanities. Nevertheless I was surprised to recognize —by means of the reviewer's questions and comments— that "history" (including all its hermeneutical problems) in the field of computer science does not seem to be much farther than just about twenty years away from now.

2.2 Domain of Discourse

"A domain of discourse is represented by a pair of sets: an entity set and a property set respectively. The pair is denoted by <Ent,Prop>. Apart from being non-empty, no assumption about the cardinality of these sets is made" [14](p.174). This entity-property-approach in [14] seems to hint already at Kourie's later, well-known, long-term research interests into the topic of lattice-theoretical *formal concept analysis* [10], which is also entity-property-based. In that context, 'entities' E were intuitively regarded as apparently un-typed constants (of whatever kind) —no type system was mentioned in [14], in spite of Russell's well-known paradox for arbitrary sets of un-typed entities— whereas properties were represented by clauses $p(E)$ in which each literal sub-clause comprised E as one of its arguments. From a pragmatic (i.e.: purposeful) modelling point of view it was taken for granted that *"only properties which are considered directly relevant in the domain of discourse are included in Prop. However, it will be assumed that if a clause is considered relevant then each separate literal in the clause together with its negation is also in Prop"* [14](p.175).

Interpretation was defined as $I : (Ent \times Prop) \rightarrow \{\mathsf{T}, \mathsf{F}, \bot\}$ whereby three notations "$\vdash p(E)$", "$\vdash \neg p(E)$", "$?p(E)$" were used accordingly for the three possibilities of $I(E, p(E))$. Consequently: *"Each entity, E_i, thus partitions Prop into three sets called the true, unknown or false property sets of E_i"*. Moreover: *"It can be easily shown, using the assumption of consistency in 3-valued logic, that if Prop and true(E_i) are given, then it is possible to infer false(E_i) and unknown(E_i)"* [14]. Thus, for clarification: The semi-formal expression "*true*(E)" in [14] was obviously meant to denote, intuitively, some set of entities for which some given property p holds true, such that $\forall e \in E\ \exists I\colon\ I(p(e)) = \mathsf{T}$ — accordingly also for *false*(E) and *unknown*(E). Those considerations also presumed *"that the truth value of a property assertion in a domain of discourse is fixed over time, space, environment, etc. It is*

either true (or false) under all circumstances (or modalities), or else it has value 'unknown' — even" (if) *"it is known to be true (or false) in some circumstances"*. In other words: Kourie's system was intended for the modelling of static systems, (though it might —perhaps— have been possible to capture dynamic systems via some static representation of their completed trace trajectory from start to termination).

Last but not least, in this sub section: *"It is interesting to note that entities may be characterized as either deterministic or non-deterministic in terms of the foregoing model. An arbitrary entity $E_i \in Ent$ is deterministic iff $unknown(E_i) = \emptyset$. An entity which is not deterministic is non-deterministic"* [14](p.175). There the term 'deterministic', which is usually applied to dynamic processes, not to static entities, was somewhat mis-used: obviously the intended meaning was that an entity could be completely 'determined', or 'known', as far as all its properties are concerned. From a computational point of view I may also remark that such an explicitly trivalent logical model is not so easy to implement. Typically the 'unknown' property $\perp$ is 'characterized' only implicitly by the non-termination of a model-checking algorithm in a bivalent logic. Alternatively to a trivalent domain of truth values one can always lift the 'unknown'-information up onto the level of predicates (by introducing a suitable meta-predicate U) and thus reduce the trivalent to a classical bivalent logic in T and F.

2.3 Property Definitions

As mentioned above, notation and treatise in [14] were only semi-formal and appealed to the reader's intuition at several occasions. For the understanding of this sub section —as well as for this entire chapter— it must always be kept in mind that Kourie had *used* in [14] some of the usual well-known elements or snippets of logic syntax in order to support the expression of his novel ideas, but he had *not* elaborated a logical 'theory' in a strictly formalistic sense of the term 'theory'. This is also the reason why one can find in [14] a peculiar ambivalence as far as the *type* of logic is concerned: Though Kourie expressed himself in [14] in the *symbols* of pure propositional logic (on the lexical level), i.e.: without '$\forall$' and '$\exists$', he seems to have had a first-order logic in mind (on the hermeneutical level). Even the use of the syntactical symbols was occasionally 'non-standard' in [14].

Anyway: *"In this section two arbitrary properties $p_x(E)$ and $p_y(E)$ in Prop are considered, and for conciseness they are denoted by x and y respectively. The notation $x \rightarrow y$ is used to assert that y logically follows from x. A necessary condition for this to hold is that every entity in Ent which has property x (and by assumption there will be at least one such entity) also has property y. Conversely, $\neg[x \rightarrow y]$ asserts that y does not logically follow from x. A sufficient condition for this to hold is that some entity in Ent which has property y (and by assumption there will be at least one such entity) does not have property x"* [14](p.175). At this point one can see again a peculiar ambiguity between semantic (or model-theoretic) and syntactic (or axiomatic-deductive) notions of 'follows logically'. Nevertheless, the following definitions were then provided on the basis of those preliminaries:

- *"Property x refines property y iff* $\vdash ([x \rightarrow y] \wedge \neg[y \rightarrow x])$

- *Properties x and y are equivalent iff* $\vdash ([x \rightarrow y] \wedge [y \rightarrow x])$
- *Properties x and y are independent iff* $\vdash (\neg[x \rightarrow y] \wedge \neg[y \rightarrow x])$
- *Properties x and y enrich one another iff they are independent, but are common properties of at least one entity in Ent.*
- *Properties x and y are mutually exclusive iff they are independent but do not enrich one another"* [14](p.175).

Here are some simple clarifying examples:

- 'sky-blue' *refines* 'blue'.
- 'loud' and 'noisy' are *equivalent* (at least approximately just for the sake of this example, which immediately motivates the question whether or not any exactly equivalent properties can be discovered empirically in the physical world).
- 'blue' and 'noisy' are *independent*.
- 'noisy and 'silent' are *mutually exclusive*.

Considering the novelty of the concept of *enrichment* at the time of [14] the 'wordy' informality (hence: unclarity) of its definition was probably the weakest point of that paper, particularly since also no illustrative examples were provided in [14]. Moreover, as the definition of properties in [14] were always *entity*-based properties $p(E)$ (*extensional* approach: a kind of nominalism), and never regarded as entity-independent *universals* (which would have been an *intensional* approach), the definitions of above lacked accurate quantification ('$\exists$' or '$\forall$') as far as the underlying sets of entities are concerned. With the theoretical tools provided by the —later— formal concept analysis (FCA) [10] it would have been quite easy to resolve those early informality-ambiguities in a concise and elegant manner.

Anyway that sub section concluded with Kourie's assertion that *"any pair of properties that an entity has either enrich one another, or one of them refines the other"* [14](p.175). As it will shown below in subsequent sections of this chapter, already the classical logical terminology provided concepts for such semantic relations.

2.4 Abstraction

"Consider two distinct entities E_i and E_j in Ent. E_j is defined as an abstraction of E_i, iff true(E_j) is a subset of true(E_i)" [14](p.175). One of the reviewers of the pre-published draft of this chapter had asked why a similar notion of abstraction was not also defined in the domain of *false*(E), but this question cannot be answered on the basis of the factually given historical material of [14]. Anyway, because of the extensional approach in [14], this definition of the notion of abstraction also had implications as far as the previously defined sets *unknown*(.) and *false*(.), as well as their various sub set relations, are concerned. Those considerations particularly entailed (amongst others) Kourie's following informal assertions:

- *"Properties in true(E_i) but not in true(E_j) are in unknown(E_j). Similarly properties in false(E_i) but not in false(E_j) are also in unknown(E_j). Informally, therefore, there is a parallel between abstraction and non-determinism. The degree of abstraction is directly related to the degree of non-determinism.*

- *An entity may be an abstraction of several entities which may be, but need not be abstractions, of one another"* [14](p.176).

The first one of those two points indicates quite clearly that Kourie wanted to avoid any 'closed-world' assumptions characterized by the classical tertium-non-datur axiom, though Kourie's original work [14] did not motivate very explicitly the rationale behind his preference for an 'open-world' model — possibly he might also have had some particular AI scenarios in mind, for which the classical-logical tertium-non-datur models would have been too restrictive or inadequate.

Also the second point (of above) is interesting from nowadays perspective, because Kourie had defined the notion of *abstraction* on the basis of *entities*, whereas his notion of *refinement* was defined on the notion of *properties*. Because of those different basis notions —entities versus properties— Kourie's notions of abstraction and refinement appear somewhat 'orthogonal' againt each other, whilst the 'canonical' interpretation of abstraction and refinement had simply regarded them as *inversely* related concepts (with abstraction as a kind of anti-refinement).

Those informal definitions also motivate some philosophical-ontological questions about the 'nature' of those entities in the universe of discourse. 'Brute' entities, in their individual-substantial *uniqueness*, which John Duns Scotus had dubbed 'HAECCEITAS'), could never be abstractions of one another from an ontological point of view. The fact, however, that Kourie *has* postulated the possibility of some entities to be abstractions of other entities, implies that those entities must be of *composite* nature, such as —for example— software specification documents at various degrees of richness in detail. This ontological implication, however, is at odds with the earlier stipulation: *"Each $E_i \in Ent$ is a constant symbol which is interpreted as a distinct entity of interest in the domain of discourse"* [14](p.174). Those entities seem to be the kind of 'brute' individual substances to which the notion of 'HAECCEITAS' by Duns Scotus seems appropriately applicable, in logical and ontological contradiction to the assertion of above according to which an entity *"may be an abstraction of several entities"* [14](p.176). Indeed, as also this chapter's reviewer has noticed: Kourie's theoretical scheme is only consistent with a Scotian notion of entities if a universal for Duns Scotus may itself be an individual instance of some higher universal. However, it would be difficult to find evidence of such thinking in the texts of Scotus who had clearly stated that *"haecceitas est singularitas"* [12].

Last but not least in this sub section there seem to be conceptual problems with an explicit *implementation* of the postulated *unknown*-set from a constructive computational point of view (i.e.: if one could implement it, how could it then be 'unknown', and if it would be unknown in the genuine sense of the word, how would it then be possible to reason about it computationally effectively?), as well as with some peculiar equivocation in the

notions of 'determinism' and 'non-determinism' which are usually applicable to dynamic processes, (as opposed to static entities).

2.5 Refinement/Enrichment

"Based on these notions, an entity may sometimes be regarded as a refinement or an enrichment of one or more of its abstractions" [14](p.176). At this point it must be kept in mind that Kourie had unconventionally defined refinement as a relation between two properties universal over entities, whereas abstraction was defined as a relation between two entities universal over properties — I shall further clarify these relations by means of the classical notion of *residuum* in a subsequent section of this chapter. Based on a property-difference set D, which *"is the set of properties that E_r has in addition to those in E_a"*, whereby E_r and E_a were individual entities, too, the following definitions were stipulated in Kourie's paper:

- *"Entity E_r is a refinement of entity E_a iff every property in D refines some property in true(E_a).*
- *Entity E_r is a enrichment of entity E_a iff every property in D enriches all property in true(E_a)"* [14](p.176).

No further explanation was provided about the choice of $\exists$ (*"some"*) in the definition of refinement versus the choice of $\forall$ (*"all"*) in the definition of enrichment. Thus, without such motivations, those early definitions still had some flavour of ad-hoc-ness to them. That subsection was concluded with the following additional explanation: *"Note that if some (but not all) properties in D refine properties in true(E_a), and the remaining properties in D enrich all properties"* (in) *"true(E_a), then E_r neither refines nor enriches E_a"* [14](p.176). Only very recently, in his book on correctness by construction co-authored with Watson, Kourie exemplified those rather abstract ideas more concretely when he *restricted* the notion of enrichment to a more specific application domain, namely *object-oriented programming* (OOP), and suggested that some sub class $\mathcal{E}$ to some given class $\mathcal{A}$ should be regarded as an enrichment of $\mathcal{A}$ *"if and only if $\mathcal{E}$ introduces new members but does not override any inherited members of $\mathcal{A}$"* [15](p.47).

2.6 Corollaries

In the original text a sub-section entitled Corollaries succeeded those definitions and brought home a larger number of small lemmata about the various meta-relations between those basically set-theoretical concepts and relations, such as (non-) reflexivity, (non-) transitivity, etc. Most important in that sub section, from an ontological point of view, is a point in which Kourie now also conceded explicitly that his notion of abstraction implies a particular ontological committment as far as the entities $E_i \in Ent$ were concerned:

- *"The entity set may be partitioned into two sets: a set of entities which are abstractions of other entities and a set containing the remaining entities. Elements of the latter set*

may appropriately be called concrete entities in the domain of discourse, and elements of the former, abstract entities" [14](p.176).

This is obviously similar to the concept of abstract super-classes versus instantiatable sub-classes in the realm of object-oriented programming, which started to become popular during the time in which Kourie wrote and published his article, (though Kourie himself did not mention this analogy). Moreover:

- *"Entity refinement and entity enrichment are mutually exclusive notions.*
- *If an entity E_a is an abstraction of entity E_r, and"* (the) *"latter has exactly one more property in its property set than the former, then E_r either refines E_a or E_r enriches E_a"* [14](p.176).

Those lenghty and informal definitions leave the reader alone with some feeling of confusion, because 'intuitively' the two notions of enrichment and refinement seem to be 'somehow related' (via the notion of abstraction), in spite of Kourie's conjecture that they would be mutually exclusive. Also somewhat puzzling in the point of above is the emphasis on *"one"* more property — what would be the relation between E_a and E_r if the latter would have exactly *two* or *three* more properties in its property set than the former?

Without any motivating examples, all those early definitions and corollaries must have appeared to their readers as somewhat arbitrary. The newer notion of *retrechment*, which shall be discussed in one of the subsequent sections of this chapter, will shed some additional light on this conceptual problem.

2.7 Base Abstractions

Another conceptual difficulty in [14] arose from Kourie's terminological 'overloading' of his concepts of abstraction, refinement and enrichment for *both* entities *and* properties. The informal (and thus unclear) treatment in the original text's sub section entitled Base Abstactions reveals this confusing 'overloading':

"If a given property in Prop refines another in Prop, it will be called a refining property. All those properties in Prop which are not refining are called base properties. Note the following:

- *Every pair of base properties in the property set of an arbitrary entity enrich one another.*
- *A refining property always refines at least one base property, but may also refine one or more other properties.*
- *The property set of any entity E_i can be uniquely partitioned into a non-empty set of base properties, and a (possibly empty) set of refining properties. These sets will be denoted by bas(E_i) and ref(E_i) respectively.*

An abstraction E_j of E_i which is such that $true(E_j) = bas(E_i)$ will be called a base abstraction of E_i, and will be denoted by $ba(E_i)$" [14](p.176).

No motivation (nor illuminating example) for this introduction of the notion of base abstraction was provided. Only a subtle pragmatic hint at the intended purpose of base abstractions was given in the final paragraph of that sub-section, according to which *"in applying progressive refinement and/or enrichment to arrive at the properties of an entity E_i, much of the initial creative effort will go into determining $bas(E_i)$, and indeed into the base abstraction of E_i if it exists"* [14](p.176).

Moreover it must be kept in mind that the 'property set of an entity' in the terminology of [14] is *not* identical with the set $true(E)$. From the perspective of some given property p there can be many entities which have this property, whereas from the perspective of a given entity e there can be many properties which this entity has: this multi-relation between many properties and many entities is further elaborated in the theory of formal concept analysis, FCA [10], which later became one of Kourie's favourite theories to work with in the fields of computer science and software engineering. Nevertheless, in spite of the vagueness of those hints (and particularly the informality of the notion of base abstraction), it seems fair to say that much of nowadays model-driven development (MDD) proceeds in some or another manner on the abstract-to-concrete path via refinement and/or enrichment, such that Kourie's early ideas expressed in [14] can be appropriately re-interpreted and re-understood within the 'hermeneutical horizon' of MDD and OOP.

The reminder of Kourie's paper [14], which cannot be (and also does not need not be) recapitulated in detail for the purpose of this Festschrift chapter, consisted in yet another section on algebraic properties (particularly ordering relations $>$ and $\geq$) and finally a literature discussion section entitled 'Other Work', which ended with Kourie's words: *"To the author's knowledge the theme of enrichment has not been formally addressed in the literature on software development"* [14](p.178).

2.8 Intermediate Summary

According to [14] —though not always explicated formally and clearly— both refinement and enrichment are meant to add details to some abstract entity. Particularly the notion of enrichment was Derrick Kourie's novel contribution to the body of knowledge of software modelling in the year 1989. Though both refinement and enrichment add details to some abstract entity, they do so in specifically different manners. This difference is related in [14] to the notion of *implication* (or logical consequence) in a positive as well as in a negative mode:

- *Refinement*, according to [14], adds details to some abstract entity (in such a manner that a *formal* (logical) relation, (namely $\rightarrow$), will hold between the abstract entity and its corresponding refining entity.
  - For example: an entity which is red and hot is certainly red, whereas an entity which is red is not necessarily red and hot.

- *Enrichment*, on the contrary, adds information to some abstract entity in such a manner that this logical relation, (namely $\rightarrow$), does *not* hold between the abstract entity and its corresponding enriching entity.

In the philosophy of science [7] [8], this difference is well-known as the difference between 'The Formal' (logical) and 'The Material' (empirical), whereby 'new' empirical knowledge in the material sciences consists exactly in those statements which *cannot* simply be deduced mechanically from some a-priori axioms via the rules of a formal calculus. Philosophically, 'The Formal' is also associated with the notion of *neccessity*, whilst 'The Material' relates to the notion of *contingency*. For comparison: Kourie's logico-deductive notion of refinement in [14] would thus also relate to some old ideals of reasoning in Kant and Hegel in which new knowledge (with more and finer details) was hoped to emerge purely rationalistically, and with necessity, by some kind of logical conceptual unfolding, from some primordial a-priori concepts to their further necessary implications, in the mind of the philosopher.

To further illustrate the relation between refinement and enrichment somewhat aphoristically I might say that, according to 'the spirit' (though not literally 'the letter') of Kourie's 1989-paper, *refinement is a kind of formal-logical enrichment, whereas enrichment is a kind of material-empirical refinement*. Both are, of course, variations of *specialisation*, which diminish the extent of un-certainty of something, (whilst abstraction is related to the opposite of specialisation, namely generalisation). Therefore, from my science-philosophical perspective, it remains debatable whether or not refinement and enrichment are really and categorically that much *"mutually exclusive"* as they had been portrayed in [14]. My elaborations in the subsequent sections shall shed some additional light on this issue.

3 Related Work

Whilst Kourie's notion of enrichment in computer science and software modelling has remained rather unique since the publication of his 1989-paper [14], various notions of abstraction and refinement are now commonplace and appear in countless publications. Contrary to Kourie's static universe of discourse *Ent* in [14], refinement is now a well-established notion also in the formal treatment of dynamic systems with methods and languages such as *CSP*, *B*, *Event-B*, and the like, where it often refers to relational properties of the *traces* which such dynamic systems can observably emit. I shall keep the discussion of these matters brief in this section, because they are nowadays commonplace in the computer science and software engineering literature. The refinement-related notion of *retrenchment* (by Banach, Poppleton, et al.), however, deserves some additional attention because —similar to Kourie's notion of enrichment— retrenchment, too, is related to the notion of refinement in a subtle manner which is not easy to grasp at a superficial first-glance level.

3.1 Abstraction

A recent paper by Colburn and Shute [9] is interesting because it discusses abstraction from a perspective of meta-informatics, i.e.: the sub-field of philosophy of science *about* informat-

ics. Paper [9] is thus not a technical computer science or software engineering paper which would somehow simply 'use' or 'apply' some particular notion of abstraction as 'given'. This paper will therefore be recapitulated first. Thereafter follows a brief overview of the notion of abstraction in the seminal trilogy on software engineering by Bjørner, as an example of a technical (rather than philosophical) contribution from within the domain of computer science and software engineering. According to [9], *"philosophy of computer science should be able to characterize abstraction as it occurs in computer science, and also relate it to abstraction as it occurs in its companion, mathematics"*. In their paper the authors show *"that the fundamental nature of abstraction in computer science is quite different from that in mathematics. In doing so, we appeal to the ways in which abstraction actually facilitates the creation of software"* [9]. In other words, those two different notions of abstraction are related to the different practical goals to which mathematics and computer science are committed. *"In summary, abstraction has been characterized in philosophy, mathematics, and logic as the process of eliminating specificity by ignoring certain features. We will show that abstraction in computer science is fundamentally different by first focusing of mathematics and computer science"* [9]. Subsequently the authors specifically identified *information hiding* as the notion of abstraction particularly suitable for computer science, though the mathematical notion of abstraction can also occur in computer science so far as mathematical concepts and techniques are also used as auxiliary means in the domain of computer science. Unlike in pure mathematics, which is a formal as opposed to a material science [7](p.27), *"there are aspects of computer science which cannot be 'abstracted away' to make them cleaner, as"* (it) *"is done in mathematics. This point is all too often ignored by those who emphasize a mathematical paradigm for computer science"* [9]. In other words, *"abstraction in mathematics facilitates inference, while abstraction in computer science facilitates the modeling of interaction"* [9]. Abstraction in mathematics is twofold: On the one hand it is used to remove material meaning for the sake of pure form, and on the other hand it is used to filter the relevant from the irrelevant details. *"As all students of logic learn, for example, it is the form of modus ponens that makes it valid, and not the meaning of the sentences that comprise it. So mathematical abstraction in this sense eschews any nonlogical meaning"* [9]. On the other hand in mathematics *"another kind of abstraction is employed when deciding that certain properties, the color of a right triangle, or the processing speed of a computing device, are irrelevant when reasoning about the properties of the class of such things as a whole. This kind of abstraction makes judgments about what is essential to a concept and what is not"* [9]. Consequently *"there are at least two kinds of abstaction in mathematics: the emphasis of form over content, and the neglect of certain features in favor of others"* [9]. Colburn and Shute have characterized this meta-mathematical notion of abstraction as *information neglect*. As mentioned above, information neglect also occurs in computer science wherever mathematics is applied in computer science as an auxiliary tool. *"It is our contention, however, that computer science is distinguihed from mathematics in the use of a kind of abstraction that computer scientists call information hiding"* [9]. Information hiding, as opposed to information neglect, does not delete any information which would be needed for the purpose of constructing operational technical systems: one only constructs another

layer of indirection, with less details, on top of a layer which contains those details. Much of the remainder of Colburn's and Shute's article was dedicated to the illustration of this argument by many examples from various sub-fields of computer science and software engineering, including data abstraction, procedural abstraction, class design patterns, software architecture patterns, layered operating systems, virtual machines, and the like [9]. In this context of informatics Colburn and Shute have even regarded a Byte as an information hiding abstraction of several Bits [9], which is particularly interesting because their Byte/Bits example comes very close to classical philosophical notions of concept abstractions, for example the abstraction of an accumulation of somehow related ships as a 'fleet'. Whether such a fleet (Byte) 'really' exists, or whether 'fleet' ('Byte') would merely be a name for a collection of ships (Bits), was an issue of dispute over the existence or non-existence of *universals* since the scholastic philosophy of the middle ages.

In his seminal trilogy on software engineering, Bjørner has discussed his notion of abstration particularly in volumes 1 [4] and 2 [5]. *"By an abstraction we shall understand a formulation of some phenomenon or concept of some universe of discourse such that some aspects of the phenomenon or concept are emphasised (i.e., considered important or relevant while others are left out of consideration (i.e., considered unimportant or irrelevant)"* [4](p.231). In fact his entire chapter 12, with over 30 pages, in volume 1 is dedicated to the notion of abstraction, such that it cannot be concisely summarized here. Here it is only important to note a further conceptual distinction in [4] between the two sub-notions of *property*-oriented versus *model*-oriented abstraction:

- *"By a property-oriented abstraction we shall understand an abstraction of some phenomenon or concept of some universe of discourse such that the abstraction is primarily or solely expressed in terms of logical properties.*
- *By a model-oriented abstraction we shall understand an abstraction of some phenomenon or concept of some universe of discourse such that the abstraction is primarily or solely expressed in terms of mathematical entities such as abstract tokens, sets, Cartesians, lists, functions, etc."* [4](p.231).

These two sub notions do not logically exclude each other, such that a property-oriented abstraction (in the terminology of Bjørner) could at the same time also be a model-oriented abstraction. Whereas Bjørner's notion of property-oriented abstraction is obviously in close proximity to Kourie's earlier theory (including even the very term 'universe of discourse') [14], Bjørner's notion of model-oriented abstraction seems to stand in closer proximity to the notion of information-hiding abstraction as identified in the above-mentioned paper by Colburn and Shute [9]. In his volume 2 [5], Bjørner treated yet two further sub notions of abstraction, namely *hierarchical* versus *compositional* abstraction, defined as follows:

- *"By a hierarchical abstraction we mean a description (or a development) which initially emphasises the overall structure of the phenomenon ('thing', system, language) being described (or developed) as decomposable into parts and which then proceeds to emphasise the further decomposition of parts into subsidiary such, etc., descending downwards a final emphasis on the atomic parts of the phenomenon or concept.*

- *By a compositional abstraction we mean a description (or a development) which initially emphasises (i.e., presents or develops) the atomic parts of the phenomenon or concept being described (or developed) and which then proceeds to emphasise the composition of concepts from atomic parts, etc., ascending towards a final emphasis on the whole phenomenon or concept as composed from parts"* [5](p.38).

Hierachical abstraction is thus associated with a *"Top-down"* approach whereas compositional abstraction is associated with a *"Bottom-up"* approach of software construction [5]. Those definitions from volume 2, however, which Bjørner wrote for software engineering practitioners, are so lengthy and 'wordy' that they cannot easily be formalised. Consequently it is not only difficult to analyse in which type of relation compositional abstraction and hierarchical abstraction stand to each other —e.g.: are they mutually exclusive from a formal-logical point of view?— but it is also hard to relate both of them back to the two previous sub-notions, property- versus model-oriented, of abstraction from volume 1 — e.g.: is compositional abstraction always also a model-oriented abstraction? (and the like).

There is no need to emphasize that much more literature on the notion of abstraction could be cited and discussed at this point. In many cases, however, the definitions and notions of abstraction given in the literature are *domain-specific* rather than science-philosophically general notions. A famous example is the notion of λ-abstraction in the Lambda Calculus [3](p.6). Liu, in his textbook on formal engineering [17](p.243), mentioned 'abstract design' without having provided any definition of abstraction as such, whereas Kourie briefly mentioned —also without going into details— the notion of object-oriented 'class abstraction' in his textbook on correctness by construction co-authored with Watson [15](p.46).

3.2 Refinement

The literature on refinement has grown so vastly during the decades since the publication of [14] that there is no hope of presenting a comprehensive synopsis of it in this chapter. *Process* refinement (dynamic) as well as *data* refinement (static) are known, whereby data refinement is also known as *reification* [16] [4](p.587, p.628). The experts further distinguish between refinement as a software engineering *activity* and refinement as a result or *product* of such activity.

Bjørner has provided this very general and informal definition: *"Refinement is a* relation *between two* specifications*: One specification, D, is said to be a refinement of another specification, S, if all the properties that can be observed of S can also be observed in D. Usually this is expressed as $D \sqsubseteq S$. (Set-theoretically it works the other way around: in $D \supseteq S$, D allows behaviours not accounted for in S.)"* [4](p.628). A long list of further refinement literature references, considering conceptually and technically different notions of 'refinement', can be found in [2]: *"Not all of these are compatible, in the sense that a development step which is a refinement according to one notion may not be one according to another; see [...] for some relevant discussion"* [2](p.302).

In his most recent textbook co-authored with Watson, Kourie has (re-)defined his notion of refinement in the domain of specifications in terms of a logical satisfaction prob-

lem: *"Spec*$(P, S, Q) \sqsubseteq$ *Spec*(P', S', Q') *if and only if* $\forall C : GCL \cdot$ *Sat*$(C$,*Spec*$(P', S', Q'))$ $\Longrightarrow$ (*Sat*$(C$,*Spec*$(P, S, Q))$*"* [15](p.35), and then linked this definition with the well-known refinement *rules* about *stronger pre*-conditions [15](pp.35-36) and *weaker post*-conditions [15](p.36). Here it is interesting to note that Kourie did *not* directly define the notion of refinement in terms of those pre- and post-condition rules. A few pages further, Kourie then defined, rather informally, a domain-specific notion of 'class refinement' as follows: *"Suppose* $\mathcal{R}$ *is a subclass of class* $\mathcal{A}$*. Then* $\mathcal{A} \sqsubseteq \mathcal{R}$ *if and only if* $\mathcal{R}$ *does not introduce new procedures as members and* $\mathcal{A}.P \sqsubseteq \mathcal{R}.P$ *for every procedure* $\mathcal{A}.P$ *in class* $\mathcal{A}$ *that is overridden by procedure* $\mathcal{R}.P$ *in class* $\mathcal{R}$*"* [15](p.47), however without demonstrating that such a definition of class refinement is a special case or instance of the more general logical *Sat*-definition of refinement as shown above. What makes this double example from Kourie's most recent textbook so interesting is that it shows that the question of refinement does not only occur at object level (i.e.: refinement amongst two system specifications S and S') but also at the logical meta level, such that one can reasonably ask whether or not one (specific) *definition of* 'refinement' is a refinement of another (more general) *definition of* 'refinement'. In [15], however, Kourie has not carried out such a conceptual self-application of refinement amongst definitions of 'refinement'.

In [16], Liu has used the pre-weakening- and post-strengthening-rules to directly define a notion of *operations* refinement. Thus he did *not* first define 'refinement' and then relate the weakening and strengthening rules to the initial definition as it was done in [15]. With reference to Morgan, Liu defined that for $P \sqsubseteq Q$ *"the following two conditions must be satisfied:*
(1) $pre\text{-}P \Rightarrow pre\text{-}Q$
(2) $pre\text{-}P \wedge post\text{-}Q \Rightarrow post\text{-}P$*"* [16].

Later in [17], Liu has characterised refinement informally not as a timeless logical relation but as *"an activity of improving a specification by resolving non-determinism. The result of a refinement is a concrete specification or program that does exactly what is required in the abstract specification"* [17](p.252). In this case, however, the question arises why Liu has spoken of 'improvement' when the result 'does *exactly* what is required' — where is then room for 'improvement'? The formal definition of refinement somewhat further down the lines in [17](p.253) was still the same as in [16].

With reference to the classical notion of operations refinement, a more precise characterisation of process refinement was presented in [2] on the basis of a formal definition of the notion of *simulation*. This approach requires a distinction between an 'abstract step' (of some abstract process) and a 'concrete step' (of some concrete process). Similar to the notion of refinement in the framework of CSP (see below), the notion of refinement in that paper was meant to *"stay within a forward simulation formulation"* [2] with a strong 'flavour' of operational (rather than denotational) semantics.

In an old reference handbook on the *Z* notation [23], the above-mentioned distinction between *operational* refinement and *data* refinement was already made. In the context of *Z*, both types of refinement are instances of a property-preserving specification-implementation-relation. In the context of *CSP*, *Roscoe* has defined the notion of refinement as follows: *"The*

process P can be used in any place where $P \sqcap Q$ would work, since there is nothing we can do to stop $P \sqcap Q$ behaving like P every time anyway. If R is such that $R = P \sqcap Q$ we say that P is more deterministic than R, or that it refines R" [19].

Both refinement notions, in *Z* as well as in *CSP*, are thus based on the practical purpose of diminishing uncertainty, or diminishing the degree of non-determinism towards 'more determinism', (if I may sloppily say so), or 'reducing non-determinism' (as the practitioners of *Z* would say). In Roscoe's example, taking into account the disjunctive behavoural semantics of the *CSP* symbol '$\sqcap$', the logical relation is clear:

- R '(behaves like P)' $\Longrightarrow$ R '(behaves like $P \sqcap Q$)'.

Because the property $p :=$ '(behaves like $P \sqcap Q$)' is a formal logical consequence of the property $p' :=$ '(behaves like P)', i.e.: $p' \Longrightarrow p$, property p' refines property p as it was also stipulated by Kourie's definition in [14]. But what about the *entities* — in this *CSP* example: the processes? According to Kourie's definition [14], a property-difference set D must first be determined. Let entity E_a be the process which behaves like $P \sqcap Q$ —thus: $p(E_a)$ is true— and let entity E_r be the process which behaves like P — thus: $p'(E_r)$ is true. Now, according to Kourie, one must find for D *"the set of properties that E_r has in addition to those in E_a"* [14]. But in the chosen example process E_r, which behaves like P, has actually *less* properties (i.e.: less behavioural possibilities) than process E_a which can behave like $P \sqcap Q$. This leads us to the tricky philosophical-ontological question whether there is anything such as *negative* properties?

In his original paper with his trivalent logic, Kourie had spoken about *"false"* properties [14] in the following sense: a 'false property' p^- of an entity E is a property p which entity E is *known not to have*. But this is —philosophically speaking— *not* the same as a 'negative property'. The ground of this subtle difference is the difference between ontology, about 'what *is*', and epistemology, about 'what we *know*'. The properties, particularly the 'false' ones, in Kourie's treatment clearly belong to the epistemological category, whereas in the chosen *CSP* example of above the 'negative' property clearly belongs into the ontological category. To make both of them the same, and to bridge the philosophical gap between ontology and epistemology, would require an additional meta-theory about the essence of truth, i.e.: a philosophical truth theory.

The 'negative property' in the chosen *CSP* example of above can be stated as follows: the refined process E_r *does not behave like* Q. This allows me to define now, formally: $p^- :=$ '(behaves NOT like Q)', such that for the refined process $p^-(E_r)$ is true. Any *ontological committment* (in the terminology of Quine) for or against the existence of negative properties has implications on the contents of Kourie's D-set which must, according to Kourie's formalism, be taken into account now to conclude this related-work-based example. With *tertium non datur* there are only the following two possibilities, OC versus OC', for such an ontological committment.

OC: *Negative properties are forbidden.* In this case, since p^- was the only difference between processes E_a and E_r in the example of above, $D = \emptyset$. Consequently it is *trivially true*, from a strictly formal-logical perspective, that *"every property in D refines some*

property in true(E_a)*"* [14], on the basis of the mathematical standard convention according to which 'nothing' ($\emptyset$) shall be interpreted as a special case of 'everything'. With this OC, and with the standard convention of interpreting 'nothing' as a special case of 'everything', it follows indeed that the process entity E_r, with property p', *is* a refinement of the process entity E_a, with property p. In this case, Kourie's theory of refinement [14] is indeed *consistent* with Roscoe's definition and example [19] of process refinement.

OC': *Negative properties are allowed.* Since the negative property p^-, according to which E_r does *not* behave like Q, constitutes the only difference between E_r and E_a in this example, $D = \{p^-\}$. Now, however, it is *no longer the case* that *"every property in D refines some property in true*(E_a)*"*, because:
E_a behaves like $P \sqcap Q$.
$\Longrightarrow E_a$ behaves like P, *or* E_a behaves like Q.
However $D \ni p^- :=$ 'does *not* behave like Q'.
Thus $p^- \not\Longrightarrow$ '(behaves like P)', *and* $p^- \not\Longrightarrow$ '(behaves like Q)'.
Therefore p^- does *not* refine any of the properties of E_a.

Under the assumption of the alternative ontological committment, OC', there is thus a *counter-example*, as shown above, in such a manner that Kourie's notion of entity refinement from [14] is *not* fully consistent with Roscoe's [19] notion of process refinement. In that case, under the assumption of OC', Roscoe's processes can thus *not* be regarded as proper 'entities' in the sense of Kourie's early theory.

However, why should anybody assume OC' and admit the existence of negative properties? Here, however, it must be noted that Kourie's epistemological theory of existing properties which can be known to be *"false"* [14], i.e.: $\exists(E \in Ent) : \exists(p \in Prop) : p(E) = false$, *can* indeed be philosophically reconciled with an ontological theory according to which negative properties can exist. To build this reconciling bridge accross the philosophical gap between ontology and epistemology it is only neccessary to *postulate* one additional meta-physical philosophical axiom, A, which stipulates the following relation.

A: 'It is *known* (epistemologically) that $p(E)$ is false' $\Longleftrightarrow$ '$p^-(E)$ *is* (ontologically) true'

The left-hand-side of axiom A is already provided for by Kourie's theory. The right-hand-side of axiom A would then lead us to the prerequisite of the alternative ontological committment OC', on the basis of which there can be found at least one example —see above— in which Kourie's notion of entity-refinement differs formal-logically from another notion of entity-refinement found in the related literature [19]. The deep ground for this subtle problem can be found in the manner in which the *two sub*-notions of refinement in Kourie's theory, namely *property*-refinement and *entity*-refinement [14], had been conceptually connected with each other by Kourie's definitions. In Roscoe's definition, on the other hand, this problem does not even occur, since Roscoe's notion of refinement did not make any such distinction between entity- and property-refinement.

3.3 Enrichment

Unlike the notions of abstraction and refinement, which are part and parcel of many standard textbooks in computer science and software engineering nowadays, Kourie's notion of enrichment has not yet become a familiar or widely-distributed textbook standard.

Volume 1 [4] of Bjørner's software engineering trilogy provides a brief explication of that notion in a glossary appendix, but does not treat enrichment explicitly and systematically in the main body of the text. Bjørner's glossary definition is given as follows. *"Enrichment: The addition of a property to something already existing. (We shall use the term enrich in connection with a collection (i.e., a* RSL scheme *or a* RSL class*) — of definitions, declarations and axioms — being 'extended with' further such definitions, declarations and axioms.)"* [4](p.594). As above, this definition provided by Bjørner is so vague and 'wordy' that it does not lend itself easily to a logical formalisation of its contents. Consequently it is not only hard to compare against the definition provided by Kourie [14], but it also does not show any comparison against the conceptually related notion of refinement. In other words: in Bjørners definition of enrichment —unlike in Kourie's [14]— the *specific difference* between enrichment and refinement has not been pointed out.

In his own most recent textbook, which he co-authored with Bruce Watson [15], Kourie has re-defined a notion of *class enrichment*, on the basis of his own notion of *base abstraction* [14], as follows: *"(Class Enrichment). Suppose $\mathcal{E}$ is a subclass of class $\mathcal{A}$. Then $\mathcal{E}$ is an enrichment of $\mathcal{A}$, written as $\mathcal{A} \sqsubseteq_e \mathcal{E}$, if and only if $\mathcal{E}$ introduces new members but does not override any inherited members of $\mathcal{A}$"* [15](p.47).

In other words: the old notion of enrichment from [14], which had been defined on *any* entities, has lately been *restricted* in [15] to a universe of discourse in which the entities consist specifically of 'classes' from the domain of object-oriented programming. Such a restriction on the domain of discourse also entails a modification of the mutual relation between refinement and enrichment also with regard to the relation of each of them to the notion of abstraction. *"In terms of this definition, an abstract class can be specialised in a stepwise fashion in one of two complementary directions at each step: either by refinement or by enrichment"* [15](p.47). Here, both enrichment and refinement appear thus as (different) instances of the notion of specialisation in the context of object-oriented programming. *"Because refinement and enrichment as defined above are mutually independent,"* —note the subtle difference between 'mutually *independent*' here and and the assertion: *"Entity refinement and entity enrichment are mutually exclusive notions"* in [14](p.176)— *"one could arrive at $\mathcal{S}$ by first carrying out all the necessary enrichments and then carrying out the required refinements. In principle, this could happen in the following three steps: $\mathcal{A} \sqsubseteq_e \mathcal{E} \sqsubseteq \mathcal{S}$. In [...], $\mathcal{E}$ is called the base abstraction of $\mathcal{S}$ and is characterised by the fact that none of its members are refinements"* [15](p.47).

In summary it seems fair to say that Kourie's original notion of enrichment had been rather 'dormant' during the past two decades —with only very few exceptions, such as the one in [4]— until Kourie himself has revived that notion for a more specific domain of discourse, namely OOP, in his most recently published book [15]. However, because his new notion of enrichment was defined only 'wordy'-informally in [15], it is not possible here in

this chapter to present an in-depth formal-logical analysis of the differences (or equivalence) between Kourie's new [15] and old [14] versions of that notion.

3.4 Retrenchment

Retrenchment *"is a more liberal formal technique, based on the main ideas of refinement, and is intended so that some pairs of models, which cannot be related within a development by refinement alone, can nevertheless be included within a formal development by its help"* [2](pp.301-302). Retrenchment is thus a formalised approximation to refinement, whilst refinement is, vice versa, a special case —the exactest possible one— of retrenchment.

Already much earlier, in [18], Poppleton and Banach have identified two types of refinement theories, namely *lattice-theoretic* ones (such as Kourie's in [14]) versus *relational* ones which come close to the formal notion of *simulation*. This latter one, which also relies on a notion of abstraction, is the refinement theory of concern for Poppleton and Banach: *"Relationally, refinement is characterised as a development step requiring the concrete precondition to be weaker than the abstract, and the concrete transition relation to be stronger, or less nondeterministic, than the abstract"* [18]. From a practical software engineering perspective, however, such a 'pure' notion of refinement is often not satisfactory. This type of refinement *"is too restrictive to describe all but a fraction of many realistic developments"* [1]. For this reason their theory of *retrenchment* has been developed in a long sequence of papers of which [1] had been the first. The following paragraphs briefly recapitulate Banach's and Poppleton's *early* retrenchment theory according to [1], such that the conceptual relation (similarity and difference) between this notion of retrenchment and Kourie's early notion of enrichment [14] should become evident and obvious to the attentive reader. Generally it should be noted that refinement, as a form of specialisation, stands in a complementary relation to the notion of generalisation or abstraction, such that there can be at least as many different notions of refinement as there are different notions of abstraction: a similar point was also made in [1]. On these premises retrenchment turns out to be a somewhat more permissive variant of refinement.

In [1] the authors *"address the main problem posed by the second use of the 'refinement' word, namely that there is not a refinement relation in the strict sense between the idealised high level specification, and the 'real' but lower level specification"*. One reason, amongst others, for such a refinement gap could be the difference between the so-called *"divine"* numbers (of which the authors of [1] seemed to presume essential Platonic existence), such as $\sqrt{2}$ or e or π, and their so-called *"mundane"* counterparts as they can be bit-represented in some finite digital computing machinery. Whereas an abstract mathematical machine could virtually carry out an operation such as $\pi \cdot \sqrt{2}$, the concrete engineering-variant could only approximate this operation *though* the *program code* of that machine would be considerably *more complex* —and in this sense a kind of 'refinement'— than the formal specification of its corresponding abstract machine. Retrenchment will thus be defined as *"an engineering variation on refinement"* [1]. A similar argument could be made about the *"divine"* natural numbers with $|\mathbb{N}| \approx \infty$, versus the *"mundane"* integer numbers with $|\mathsf{INT}_+| < \infty$ in the realm of engineered machines, such that no concrete I-implementation could ever be regarded

as a refinement of a corresponding abstract $\mathbb{N}$-specification, at least not in that sense of the notion of 'refinement' as it was defined in [1], namely the notion of simulation with a diminished degree of non-determinism.

Note however that the integer set INT_+ *is* nonetheless a *specialisation* of $\mathbb{N}$, because from an extensional point of view: $\mathsf{INT}_+ \subset \mathbb{N}$, whereas from an intensional point of view an *additional* logical constraint condition must be stipulated in order to define INT_+, on the basis of $\mathbb{N}$, in the classical Aristotelian technique of definition by genus and specific difference as $\mathsf{INT}_+ := \{n \mid n \in \mathbb{N} \ \wedge \ n < l \in \mathbb{N}\}$, whereby the additional predicate '$< l$' represents a specific difference in the Aristotelian sense. Now however, because of the logical implication $(x \in \mathsf{INT}_+) \Rightarrow (x \in \mathbb{N})$, this Aristotelian specialisation *is* a refinement in Kourie's property-and-implication-based notion of refinement [14], whereas this Aristotelian specialisation is *not* a refinement in the sense of Banach and Poppleton [1]. This difference in their underlying notions of refinement must be kept in mind when comparing the notions of retrenchment [1] versus enrichment [14] against each other: Whereas a refinement in the sense of [1] *"weakens the precondition and strengthens the postconditions of an operation"* in a simulation-preserving manner, an Aristotelian specialisation —which is closely related to the notion of refinement in [14]— *increases* the information-contents of the defining *intension* (intent) of a concept, and thereby typically *descreases* the cardinality of its *extension* (extent), i.e.: the set of individual entities which can be subsumed under that intensional concept.

"Retrenchment is, very loosely speaking, the strengthening of the precondition and the weakening of the postcondition though technically it's more subtle than that. This is like the opposite of refinement, except that we avail ourselves the opportunity to liberalise the connection between abstract and concrete operations more widely. For instance not only will we allow changes of data type in the state component of an operation, we will also allow flexibility in the input and output components" [1](p.133). By comparison, it is this precisely such kind of 'flexibility' —expressed in terms of additional information which cannot be merely logically deduced— that Kourie had in mind when defining his notion of enrichment in [14]. Aphoristically I may thus say that *retrenchment is an enriched inversion of refinement*, whereby the enrichment is the reason for the semantic difference between a merely formal-logical inversion of refinement and a genuine retrenchment in the sense of [1]. Formally the notion of retrenchment is not defined 'directly' in simple logical terms. It is defined only indirectly in a more complicated manner *"by its proof obligations"* [18] in the context of mutually related B-machines. This implies that the very specific notion of 'retrenchment' is somewhat context-sensitive and might vary slightly from case to case, depending on the specific B-machines between which some relations shall be formally proven. This should not be too surprizing, since the very notion of retrenchment had been introduced particularly to capture those subtle technical details which distinguish the material-real realm of computer engineering from the formal-ideal realm of pure mathematics, and which cannot be formal-logically deduced or inferred from an abstract initial specification. In a later paper by the same research group a formal notion of retrenchment was presented more generally, i.e.: not bound to the specific technique of B-machines any more [2], but still capturing the 'gist' of retrenchment as an *"almost-refinement"* [2](p.314).

From a *science*-philosophical point of view this 'ontological gap' between the technical-empirical and the formal-theoretical realms, which motivated the introduction of the notion of retrenchment in the first place, is conceptually related to the type difference between 'nomological' and 'nomopragmatic' statements in Bunge's philosophy of technology [8](pp.149-150), because the nomopragmatic statements can also not be formal-logically deduced from the nomological ones, though they are materially-empirically related to each other. Beheld from a *language*-philosophical perspective the notion of retrenchment is characterized more 'operationally' (in terms of 'doing') rather than essentially (in terms of 'being'). However, in a similar manner by which enrichments can be *composed* (because of their logical independence) according to the theory outlined in [14] and [15], also retrenchments can be composed according to the theory outlined in [18] and further papers. In particular, both theories provide some laws of transitivity: in [18], if some S is retrenched by some T and T is further retrenched by some U then (under some additional conditions) S is also retrenched by U, whilst *"enrichment of entities is a transitive and irreflexive relation"* such that it *"defines a strict but partial order $>_e$ on entities of Ent"* in [14](p.177). However, since every refinement can be regarded as a retrenchment (with further properties), there are lattices of refinements and retrenchments, too.

3.5 Evolution

An early predecessor to Banach's and Poppleton's notion of retrenchment has been the notion of 'evolution' introduced and (semi-)formalised by Liu. Similarly to Banach's and Poppleton's, also Liu's work was motivated by certain infeasibilities of 'pure' refinement in the practical workflow of software engineering: *"We observe that software development in general is an evolution process. By evolution we mean a change of the operation (or system) under its current functional constraints. For example, evolution of operation P means that P is improved by adding more information to its pre- and/or postconditions, but <u>not</u> necessarily weakening the precondition and strengthening the postcondition. We also believe that it is necessary to distinguish evolution and the widely used term modification, where the latter means, in this paper, that the original idea of the operation is changed to a different one"* [16](p.143). Consequently, *"refinement is indeed a special case of evolution, and evolution is transitive"* [16](p.142). On the premise that *"$\mathcal{R}(P, Q)$ denotes a property that predicate P is equivalent to a predicate that contains predicate Q as its component but* (does) *not occur as either $Q \vee \neg Q$ or $Q \wedge \neg Q$"* [16](p.146), the notion of evolution had been formalised as follows: *"Let P and Q be two operations. If they satisfy all the conditions:*
(1) $\mathcal{R}(pre{-}Q, pre{-}P)$,
(2) $\vdash \neg(pre{-}Q \Leftrightarrow \mathbf{false})$,
(3) $\mathcal{R}(post{-}Q, post{-}P)$, *and*
(4) $\vdash \neg(post{-}Q \Leftrightarrow \mathbf{false})$,
then we say that Q is an evolution of P (or P is evolved to Q)" [16](p.146). Subsequently: *"If operation P is changed to Q and Q is neither an evolution nor a refinement of P, Q is called a modification of P (or P is modified to Q)"* [16](p.146). The 'adding of some useful information', regardless of the strict pre- and postcondition rules in pure refinement, is thus

a common concept in the works of Liu [16] and Kourie [14]. In Liu's work, however, as remarked by Banach and Poppleton, *"the concept of evolution addresses this by demanding that the pre- and post-conditions of the abstract specification in a development step be semantically equivalent to subformulae of those at the more concrete level. Unfortunately since* $((P \wedge Q) \vee (P \wedge \neg Q))$ *is equivalent to* P *for any* Q*, the 'is a subformula of something equivalent to' property enables us to go from any* Q *to any* P *uncritically, tending to rob the evolution notion of its semantic bite"* [2](p.314). In his later textbook [17] seems to have revoked or revised his earlier notion of evolution when he wrote: *"An evolution of a specification can be one of the following three activities:*

- *Refinement*
- *Extension*
- *Modification"*

[17](p.252), in contrast to his earlier concept according to which modification *"is neither an evolution nor a refinement"* [16].

3.6 Intermediate Summary

The literature review in this section has shown that Kourie's notion of abstraction in [14] was a quite 'canonical' and often-found one, though it must not be forgotten that —as Colburn and Shute have pointed out [9]— there can be subtle differences in the notions of abstraction in the different fields of mathematics on the one hand and informatics or software engineering on the other hand. To make matters even more complicated, the mathematical notion of abstraction, which comes close to the notion of abstraction in [14], can *co-exist* in informatics together with the genuine informatical notion of abstraction understood as information-hiding. In matters of refinement, however, the variety of notions found in the literature is considerably wider, such that the notion of refinement in [14] cannot be so easily reconciled with, for example, the notion of refinement in [1] and [18]. Of particular philosophical interest in the context of Kourie's notion of refinement is the question of negative properties (yes or no), as it was discussed in comparison to Roscoe's notion of process refinement [19]. Finally, Kourie's formal notion of enrichment has remained by-and-large unique and rare during the past two decades: The seminal textbook trilogy on software engineering by Bjørner, for example, which covers almost any conceivable aspect of software engineering on approximately two thousand pages, explicitly mentions enrichment only briefly and informally in a glossary appendix to volume 1 [4]. *Implicitly*, however, some tacit notion of enrichment seems to be at work in Banach's and Poppleton's related notion of retrenchment [1] [18] which I have aphoristically dubbed as 'enriched inversion of refinement'.

4 Further Considerations

This section introduces alternative definitions for the three main notions from [14], i.e.: abstraction, refinement, and enrichment. By doing so, further semantic subleties in these three notions shall be illuminated. The definition of 'abstraction' will be a traditional one according to [25]. Both 'refinement' and 'enrichment' will then be defined w.r.t. this traditional notion of abstraction. This section will conclude with a few (however sketchy) considerations regarding the applicability of these notions in the application domain of *software product lines*.

4.1 Abstraction

According to Weingartner, already Boethius (around 480-525 A.D.) and Thomas of Aquino (1225-1274 A.D.) had distinguished *two* diffeent types of abstraction, namely [25]:

- abstraction of *properties from individuals* (in Aristotelian/Scholastic terminology: abstraction of 'form' from 'matter'), as well as
- abstraction of *the general from the specific*.

Rephrased in modern terminology, their first type of abstraction may correspond both to the notion of *class* abstraction (or *extensional* abstraction) [25] as well as to the notion of *property* abstraction (or *intensional* abstraction) [25], whereas their second type of abstraction relates to the notion of generalisation through quantification [25]. For example: if $S := \{\mathsf{apple}, \mathsf{ball}, \mathsf{car}\}$, then the number $3 = |S|$ would be an abstraction of S in that second sense of the term. (The previously mentioned example, in which 'fleet' is regarded as a 'set of ships', is not covered by this notion of abstraction. Nominalistically one could simply regard 'fleet' as a *name* for 'set of ships'.)

In any type of abstraction one can further distinguish the *activity* of abstracting (something from something) and the *result* of such an activity. The latter one is called an *abstractum* [25], whereas the *relictum* [25] remains as the 'left-over' of such an abstraction. To be even more precise, Weingartner distinguished *two* types of relictum, namely a less concrete 'relictum$_1$' and a more concrete 'relictum$_2$', for *both* extensional *and* intensional types of abstraction [25]. These concepts are illustrated by the following example cases.

Extensional Abstraction (for enumerable *things*): the cars.

Relictum$_1$: the other vehicles (e.g.: bikes).

Relictum$_2$: the roadsters.

Intensional Abstraction (for non-quantifiable *concepts*): CULTURE.

Relictum$_1$: NATURE.

Relictum$_2$: LITERATURE.

From these examples it can be learned there is always some kind of '*is-a*' (inclusion or implication) relation between a relictum$_2$ and its corresponding abstractum, however *not* between a relictum$_1$ and its corresponding abstractum, and also *not* between an abstractum and its relictum$_1$. Relictum$_2$ is thus always 'more concrete' than its corresponding abstractum, because from relictum$_2$ the abstractum has actually been abstracted. Also note that, according to the definition in [25], relictum$_1$ is not necessarily simply the negation of the abstractum: as shown by the vehicle example of above, relictum$_1$ ('the other vehicles') still shares a vehicle-ness property with its corresponding abstractum ('the cars'). Also note that the notions of relictum, as in [25], are *relative* (not absolute) notions, namely in relation to some 'universe of discourse' out of which entities and abstractions can be taken.

In OOP the relation between super-class and sub-class is similar to the relation between abstractum and relictum$_2$ in extensional abstraction *if* the super-class is also instantiatable and not an abstract class from which no objects can be generated. Un-instantiatable abstract classes in OOP would belong to the category of intensional abstraction. Without any need for constructing particular entity-property-tuples for a particular domain of discourse (as proposed in [14]) those various relations of abstraction can be formalised simply for arbitrary models (extensions). Similar to the notation in [25], let $\mathcal{A}$ be a set and $\mathcal{A}^-$ its complement set. Moreover, let the somewhat un-canonical notation '$\{e|e \in \mathcal{A}\}$' be understood as a so-called *comprehension construction* according to [20], alternatively written as '$\{e|\mathcal{A}(e)\}$', which yields a set of entities for which some attribute or assertion holds true [20] (similar to the property-based considerations found in Kourie's paper [14]). Then one can define, according to [25],

Extensional Abstraction: "$\{e|e \in \mathcal{A}\}$", see [20]. In this case an abstraction is characterised as a (sub) set of *elements* with

"Relictum$_1$: $\{r|\ \exists R : (R \subset \mathcal{A}^- \wedge r \in R)\}$", see [25].

"Relictum$_2$: $\{r|\ \exists R : (R \subset \mathcal{A} \wedge r \in R)\}$", see [25].

Intensional Abstraction: "$\{P|\mathcal{A} \subseteq P\}$", see [25]. In this case an abstraction is characterised as a (sub) set of *properties* (or qualities or attributes) with

"Relictum$_1$: $\{Q|\ \exists A : (A \subset \mathcal{A}^- \wedge A \subseteq Q)\}$", see [25].

"Relictum$_2$: $\{Q|\ \exists A : (A \subset \mathcal{A} \wedge A \subseteq Q)\}$", see [25].

In the philosophical definitions of [25] the scopes of $\mathcal{A}$ and Q (i.e.: the 'universe of discourse') were not specified. Of course such specifications would need to be provided for the computable technical applications of logics in software engineering. The examples of above —with the cars, the roadsters and the other vehicles— should suffice to illustrate the intended meaning of the relictum definitions from [25], even though also those definitions were not completely formalised in [25]. Particularly in the case of extensional abstraction, which is the relevant notion of abstraction for this chapter, it must be demanded that $R \neq \emptyset \neq \mathcal{A}^-$, otherwise the word 'abstraction' would not be appropriate to the notion it is meant to capture. Particularly by $\emptyset \neq \mathcal{A}^-$ the trivial-universal 'anything'abstraction is excluded.

Those classical definitions of 'abstraction' are clearer and more consistent than the explication found in [14](p.175) which stated that *"E_j is defined as an abstraction of E_i iff true(E_j) is a subset of true(E_i)"* — such a definition is ontologically and categorially questionable, because two *elementary things* (E_i, E_j such as: my red car, your blue van) in an extension $\mathcal{E}$ (or 'domain of discourse' in the terminology of [14]) cannot sensibly be 'abstractions' of each other, unless one would be willing to do violence to our common use of language as far as the typical meaning of the term 'abstraction' [25] in the philosophy of science is concerned. Thus, the common use of the word 'abstraction' only relates to *sets* of entities, not to individual entities, in a similar manner in which the word 'temperature' in physics relates meaningfully only to sets of molecules (e.g., a gas) but not to individual molecules as such. Indeed, one reason for the conceptual vagueness in [14] was insufficient distinction between intensional and extensional abstraction as well as unclarity as far as the two different types of relictum$_{(1,2)}$ from [25] are concerned.

In the following, as mentioned above, the notion of abstraction will be restricted to *extensional* abstraction, because software engineering is meant to construct 'things' (e.g.: programs, code-objects, etc.) which can be explicitly contained and enumerated in concrete sets. Intensional notions, since the era of the Scholastic philosophy also known as *universals*, cannot be materialised in a software-engineering manner.

4.2 Refinement

Colloquially one may say that *refinement makes under-determined definitions* (descriptions, specifications, stipulations) *more determined*. Refinement is thus a step from the more abstract to the more concrete, by being more specific in the descriptions of the objects to be captured with logical concepts — an example was given in the CSP-discussion of above. Having defined a notion of abstraction formally in the previous sub-section, the formalisation of a related notion of refinement now follows easily.

Definition: Given an abstractum $\mathcal{A}$, a relictum$_2$ of $\mathcal{A}$ is a *refinement* of $\mathcal{A}$.

Note that this is a purely *semantic* definition of the notion of refinement which does not stipulate anything about how any such refinement shall be represented syntactically in the notation of a formal specification of a desired extensional artefact. Here is thus another difference with regard to [14] where the notion of refinement had been defined in a calculus of logical implications ($\Longrightarrow$) and predicates. Typically one can achieve semantic refinement, whereby an extension *shrinks*, by means of some syntactic 'enrichment' whereby its lexical-syntactic description (i.e., its intension) *grows* [26], through the conjunctive insertion of further conditions and constraints into its notational specification. However it is also well-known that not every explicitly given concrete set S is characterised analytically by a defining membership predicate P_S —try, for example, to find a concise 'formula' for a logical predicate $P_S(n)$ to define exactly the set $S := \{\sqrt{2}, 7\}$ without petitio principii— such that the logical restriction $P'_S := P_S \wedge C$ of P_S with a conjugated constraint C is only a sufficient —not a necessary— way of yielding semantic refinement in a given extension. Anyway, as shown in the definition of above, the two notions of abstraction and refinement are closely (namely

inversely) related to each other through the mediating notion of the second relictum. An illustrative example is shown in a sub section further below.

4.3 Enrichment

As previously discussed, a theoretical notion of 'enrichment' was the main novel contribution of [14] at the time of its publication. Alas, due the peculiar choice of a trivalent logic in [14], the effects of the third truth value $\perp$ on the mathematical-logical semantics of the term 'enrichment' remained unclear (in contrast to its intended intuitive meaning).

With the notions of 'abstraction' and 'refinement' on the basis of [25] as provided in the previous sections, it will be easy to re-define and formalise 'enrichment' precisely and concisely, too. Thereby, however, care must be taken not to miss the originally intended meaning of 'enrichment' in [14]. Whereas a refinement of some extension $\mathcal{E}$ simply yields a sub set $\mathcal{E}_R \subset \mathcal{E}$, it is *not* the case that an enrichment would simply yield a super set $\mathcal{E}_E \supset \mathcal{E}$ in such a way that enrichment would naively appear as inversion of refinement. As indicated in [14], the relations between those notions are somewhat tricky.

Definition: Let an extension of entities, E, be a (second) relictum$_2$ of some abstractum $\mathcal{A}$, and let $E' := E \uplus E''$. If E'' is a proper subset ($\subset$) of the (first) relictum$_1$ of $\mathcal{A}$, then E' is an *enrichment* of E.

Once again this is a purely *semantic* (extensional) definition of the notion of enrichment which does not stipulate anything about how such enrichments can or must be denoted syntactically in terms of a formal system specification or calculus. There may be different ways of yielding semantic enrichment through the various syntactic features of a formal specification system.

Note that E'' must not be allowed to be identical with relictum$_1$ of $\mathcal{A}$, because otherwise $\mathcal{A}$ itself would be the (trivial) 'enrichment' of E —contrary to the original ideas and intentions expressed in [14]— whereby empty extensions $E = \emptyset$ are (at least formally) allowed, (such that $(E = \emptyset) \uplus \mathcal{A}$ would be the formal characterisation of such a trivial pseudo enrichment). Because the first and the second relictum of any abstractum $\mathcal{A}$ are disjoint, the definition of above respects the requirement of 'independence' in Kourie's original definition [14] of 'enrichment'. Moreover, also Kourie's requirement of common properties in his original notion of enrichment is fulfilled by the definition of above, namely by the abstractum $\mathcal{A}$ to which both the first and the second relictum are semantically related.

Last but not least also note that 'proper subset' is itself a (second) relictum$_2$ —i.e.: a 'refinement'— if one regards the (first) relictum$_1$ of $\mathcal{A}$ as an abstractum $\hat{\mathcal{A}}$ in its own right. This implies that the two notions of refinement and enrichment are closely related and interwoven with each other, in contrast to Kourie's idea that "*entity refinement and entity are mutually exclusive notions*" [14].

4.4 Example

Let $\mathbb{Z}$ be the abstractum of all integer numbers, (with symbol $\mathbb{Z}$ for German *Zahl*, i.e. number).

- Relictum$_2$ $\mathbb{N}$ is a *refinement* of $\mathbb{Z}$ which is characterized analytically by the following intensional refinement condition (specification): if $n \in \mathbb{N}$ then $n \in \mathbb{Z}$ and $n > 0$. This example shows that, in refinement, the extension is shrinking whilst the intension is lexically growing by the logical conjunction (never disjunction) with additional constraints [26].

- Moreover, $\mathbb{N}_0 := \mathbb{N} \uplus \{0\}$ is an *enrichment* of $\mathbb{N}$ with $0 \in \mathbb{Z} \backslash \mathbb{N}$. If $\mathbb{N}$ is regarded as an abstractum on its own, then $0 \in$ relictum$_1$ of $\mathbb{N}$.

Note that, at least from a classical (non-constructivist) point of view, $\mathbb{N}$ (as a set) would still be a (semantic) refinement of $\mathbb{Z}$ (as a set) even if nobody could find a concise analytic characterisation of a logical refinement condition (such as $n > 0$) at the syntactic level of a specification. Thus it is *not* necessary to define the notion of refinement in terms of predicates and material implications (P, $\Longrightarrow$) as it was done originally in [14]. For related opinions about replacing intensional by extensional characterizations see [27]. For example the *pseudo* predicate $P(x) :=$ '*the input x which drives the Universal Turing Machine into divergence*' is not well-defined (otherwise the general halting problem would be solved), or, as Frege would have said: it has got 'sense' but it does not have 'meaning' [11] — though it is known that at least one such number x must exist and that the set of all those xs is indeed a proper subset of $\mathbb{Z}$.

4.5 Software Product Lines

According to Carnegie Mellon's Software Engineering Institute (SEI), *"a software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"* [29]. In this definition three occurrences of the term 'set' can be found, namely:

- set of systems,

- set of features,

- set of core assets,

such that this set-based definition of SPL should be amenable to a rational reconstruction in terms of set-based notions of abstraction and/or enrichment as discussed in the previous sections of this chapter. Most important in SEI's definition of SPL are the set of systems and the set of features, as they form the synchronous part of that definition (whereas the diachronous-genealogical explication that these two sets had their origin in some common set of assets is so not relevant in the context of this chapter). SEI's definition, however, does

not clarify whether or not *all* systems in the system set *together* have at least one common feature, or if only *any two* systems in the system set have at least one common feature *pairwise*. The latter situation would be an instance of Wittgenstein's notion of 'family resemblance' (in the German original: *Familienähnlichkeit*) in his posthumously published Philosophical Investigations [28]. The difference between the two notions, about which SEI's definition of SPL does not make a very clear statement, is illustrated in the following figures. If a system is simply regarded as a set of features, then a SPL as a set of n systems $\{S_1, \ldots S_n\}$ could have the property $\bigcap_i S_i \neq \{\}$ $(i = 1, \ldots, n)$ which could be called a *tightly related* SPL. Alternatively it could be characterised by the property $\bigcap_i S_i = \{\}$ $(i = 1, \ldots, n)$ whereby $\forall i \in \{1, \ldots n\}$ $\exists j \in \{1, \ldots n\}$ with $i \neq j$ and $S_i \cap S_j \neq \{\}$ pairwise, which could be called a 'Wittgensteinian' or *family-resemblance*-SPL. Anyway, in both cases one can clearly interpret each system S_i of a given SPL as an *abstractum* (of its feature extension) in terms of the definition of above. Consequently one can also say that a system S_i *refines* another system S_j in a given SPL, if the extension of features of S_i is a proper sub set of the extension of features of S_j — or, in software engineering jargon: S_i is 'more elegant' and 'less cluttered' than S_j. Vice versa one can say that a system S_j *enriches* another system S_i in a given SPL, if both the feature extensions of S_i and S_j fulfill the *relictum* conditions as formulated above.

Last but not least, however, I must admit that the foregoing considerations on SPL are somewhat idealised. Moreover, the 'features' listed in SPL need not always be immutual atomic entities: there may also be abstract features in the form of object-oriented classes with polymorphic types which call for specific instantiations in the production of particular products or product variations.

5 Summary and Outlook

After having discussed Derrick Kourie's notion of abstraction [14] and several related concepts, I have presented in this chapter a more classical notion of abstraction on the basis of some standard texts from philosophy of science [22]. Also these concepts were presented from an extensional perspective under the classical presumption that any item belongs somehow decideably to some set. Thereby the question, *how* to decide *constructively* to *which* set a given item belongs, was not asked. It was indeed not even necessary to ask that question for the simple purpose of defining or re-defining the notions of refinement and enrichment in the context of this chapter, because I have taken the items as given and have defined any abstractions on the basis of the given items — not, as in [14], starting with some abstract property predicates P to which one would then possibly not find any extensional items in existence. Nevertheless it should not be forgotten that there exist sets which are *recursively inseparable* [21], though such inseparable mathematical sets —of which the famous Mandelbrot and Julia sets are some particularly intuitive examples— are typically not relevant in the practical software engineering domain with its countable number of finite and discrete entities.

Related to the notion of (extensionally) inseparable sets is the notion of (intensionally)

vague predicates [24] for the formalisation of which also trivalent logics —similar to Kourie's suggestion [14]— had earlier been proposed [24]. However, a trivalent logic is not the only possible logic to capture the intensional notion of vague predicates: as further described in [24], even multivalent logics (with numerical truth values) had already been proposed for that purpose. Further related to the classical notion of vague predicates is nowadays also the notion of *rough sets* [13] which are receiving more and more attention in the field of information systems and their so-called 'ontologies'. Rough sets are sets with 'borderline cases' of membership, i.e.: entities for which it is not possible to distinguish sharply whether or not they belong (perhaps even partly) to a given rough set. As further explained in [13], the notion of a rough set consequently leads to a *modified notion* of (rough) *inclusion*, $\subseteq_r$, which differs slightly in its semantics from the classical notion of set inclusion, $\subseteq$. On the basis of such a modified notion of inclusion, $\subseteq_r$, a new axiomatised calculus of rough set theory can be formulated [13] which may have a variety of its own semantic models: typically discrete models, but also over-countably continuous ones. Because the notions of abstraction, refinement and enrichment were presented in this chapter as *semantic* and *extensional* (not axiomatic-syntactic or intensional) notions on the basis of classical entity sets and their inclusion relation, $\subseteq$, it follows intuitively that it should not be too difficult (for future work) to provide some new related notions of *rough refinement* and *rough enrichment* on the basis of the rough set theory and its particular notion of rough set inclusion, $\subseteq_r$.

Finally I may remark that the set inclusion relation, $\subseteq$, which forms the basis of the definitions of refinement and enrichment in this chapter, can —obviously— be used as a model (interpretation) of a lattice order relation, $\preceq$, with the empty set $\{\} := \bot$ and the universal set $\{x|\ \exists P : P(x) = \mathit{true}\} := \top$ as the lattice's extreme elements. Indeed, the two figures from above's sub-section on product lines can be easily transformed into their corresponding formal concept lattices, as per [10], with the SPL's systems S_i as FCA's objects and with the SPL's features as FCA's properties or attributes. The related notions of relictum, refinement and enrichment can thus all be expressed and explained in Kourie's favourite theory, FCA, as well.

References

[1] R. Banach, M. Poppleton: *Retrenchment, an Engineering Variation of Refinement.* LNCS 1393, pp. 129-147, 1998.

[2] R. Banach, M. Poppleton, C. Jeske, S. Stepney: *Engineering and Theoretical Underpinnings of Retrenchment.* Science of Computer Programming 67, pp. 301-329, Elsevier Publ., 2007.

[3] H. Barendregt: *The Lambda Calculus, its Syntax and Semantics.* Series Studies in Logic and the Foundations of Mathematics 103, Revised ed., North-Holland Publ., 1984.

[4] D. Bjørner: *Software Engineering 1, Abstraction and Modelling*. EATCS Series Texts in Theoretical Computer Science, Springer-Verlag, 2006.

[5] D. Bjørner: *Software Engineering 2, Specification of Systems and Languages*. EATCS Series Texts in Theoretical Computer Science, Springer-Verlag, 2006.

[6] D. Bjørner: *Software Engineering 3, Domains, Requirements, and Software Design*. EATCS Series Texts in Theoretical Computer Science, Springer-Verlag, 2006.

[7] M. Bunge: *Philosophy of Science 1, From Problem to Theory*. Revised ed., Transaction Publ., 1998.

[8] M. Bunge: *Philosophy of Science 2, From Explanation to Justification*. Revised ed., Transaction Publ., 1998.

[9] T. Colburn, G. Shute: *Abstraction in Computer Science*. Minds and Machines 17/2, pp. 169-184, Springer-Verlag, 2007.

[10] B. Ganter, R. Wille: *Formale Begriffsanalyse, Mathematische Grundlagen*. Springer-Verlag, 1996.

[11] R. Haller, W. Grassl: *Bedeutung*, pp. 49-60 in [22].

[12] J. Hoffmeister (ed.): *Wörterbuch der philosophischen Begriffe*. Series Philosophische Bibliothek 225, art. 'haecceitas', p. 287, 2nd ed., Felix Meiner Publ., 1955.

[13] C. Keet: *Rough Subsumption Reasoning with rOWL*. Proceedings SAICSIT'11, pp. 133-140, 2011.

[14] D. Kourie: *An Approach to Defining Abstractions, Refinements and Enrichments*. Quæstiones Informaticæ 6/4, pp. 174-178, Rekenaar Vereniging van Suid Afrika, 1989.

[15] D. Kourie, B. Watson: *The Correctness-by-Construction Approach to Programming*. Springer-Verlag, 2012.

[16] S. Liu: *Evolution, a more Practical Approach than Refinement for Software Development*. Proceedings ICECCS'97, pp. 142-151, IEEE, 1997.

[17] S. Liu: *Formal Engineering for Industrial Software Development, using the SOFL Method*. Springer-Verlag, 2004.

[18] M. Poppleton, R. Banach: *Retrenchment, Extending the Reach of Refinement*. Proceedings ASE'99, pp. 158-165, IEEE, 1999.

[19] A. Roscoe: *The Theory and Practice of Concurrency*. Pearson/Prentice-Hall, 2005.

[20] K. Schütte: *Beweistheorie*, pp. 95-99 in [22].

[21] R. Smullyan: *Undecidability and Recursive Inseparability.* Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 4, pp. 143-147, Deutscher Verlag der Wissenschaften, 1958.

[22] J. Speck (ed.): *Handbuch wissenschaftstheoretischer Begriffe*, Vol. 1 (A-F). Series UTB 966, Vandenhoeck & Ruprecht Publ., 1980.

[23] J. Spivey: *The Z Notation, a Reference Manual.* 2nd ed., International Series in Computer Science, Prentice Hall, 1992.

[24] R. Trapp: *Exaktheit*, pp. 202-205 in [22].

[25] P. Weingartner: *Abstraktion*, pp. 3-4 in [22].

[26] P. Weingartner: *Extension/Intension*, pp. 217-222 in [22].

[27] P. Weingartner: *Extensionalitätsthese*, pp. 222-223 in [22].

[28] L. Wittgenstein: *Philosophische Untersuchungen.* Post. by G. Anscombe (ed.), Blackwell Publ., 1953.

[29] world-wide web, http://www.sei.cmu.edu/productlines/