Novel Didactic Proof Assistant for First-Order Logic Natural Deduction

Jorge Pais* and Álvaro Tasistro

Universidad ORT Uruguay

Abstract. We present a proof assistant designed to help learning formal proof, particularly in the system of Natural Deduction for First-Order Logic. The assistant handles formulas and derivations containing meta-variables and allows to maintain a library of instanciable lemmas. It possesses a graphical interface presenting proofs as trees and handles multiple simultaneous derivations that can be dragged and dropped into one another.

1 Introduction

This paper is about technology that helps learning *formal proof*. Proof is the main constituent of the practice of Mathematics, but it is certainly also performed in other disciplines. Particularly, it is, as we argue below, of utmost importance in software construction, despite being therein generally neglected. *Formal* proof, on the other hand, is proof framed within a formal language, or as it is traditionally called, a *formal system* of inference rules. The use of these systems in practice has been awakened, precisely by the needs of the software construction activity of acquiring certainty about the behaviour of computing systems. Out of this situation a field of research within Computing Science has begun to thrive, namely that of the design and development of *proof assistants*.

Learning proof, and especially formal proof, is therefore acquiring increasing importance, especially within the software community. Learning proof, generally, is of course of great interest for students in several disciplines and also, at least in some countries, for those at High School. In this paper we make the claim that certain systems of formal proof, most notably Natural Deduction, are useful also for learning proof in the widest sense, i.e. for developing organized manners of approaching their understanding and construction. Therefore there is some interest too in investigating the technology helping to learn formal proof through practice, i.e. that of the *didactic proof assistants*.

In [1] we presented a novel didactic proof assistant, here to be named Andy 0, for the system of Natural Deduction of Propositional Logic. One of its most interesting features was the handling of a library of *lemmas* that could be reused by instantiating them within other proofs. As pointed out in the paper, the natural continuation of that work was to extend the system to First-Order Logic,

* Currently at Northeastern University, Boston.

P. Zaphiris and A. Ioannou (Eds.): LCT 2014, Part I, LNCS 8523, pp. 441–451, 2014.

which is a standard formal language for Mathematics. However, to correspondingly extend the capacity of instantiating lemmas was far from trivial, a fact which is reflected in the abscence of such feature in existent proof assistants for the mentioned language.

In this paper we present a solution to this problem, embodied in Andy $1\frac{1}{2}$, the successor to Andy 0. It keeps the general outlook of the latter, being based on a graphical interface that displays formal proofs as trees, allowing for any number of draft proofs that can be dragged and dropped into others, and admitting combined strategies of proof development. Its main advance is the ability to work at the level of *schematic* formulas, i.e. ones that are actual patterns containing meta-variables that stand for arbitrary formulas and terms. It is a fact that the actual practice of First-Order Logic is carried out at this level, as witnessed by any textbook on the matter. It is also this feature that allows to properly handle reusable general lemmas. To our knowledge, Andy $1\frac{1}{2}$ is the first proof assistant to allow for this level of abstraction in the formal language. Its name derives from the fact that it is actually an assistant for (an adaptation of) One-and-a-Halfth Logic, as presented in [2].

The structure of this paper is the following: In section 2 we extend ourselves on the relevance of learning proof and formal proof, which justifies our interest in the related technology. We also try to present the relevant features of First-Order Logic Natural Deduction in a manner accesible to the general reader. In section 3 we describe the features that are in our opinion to be required of a didactic proof assistant. In particular, we try to give a detailed account of the technical problem involved in handling schematic formulas and proofs in the context of First-Order Logic. Having given the preceding motivation, we describe in section 4 the design of Andy $1\frac{1}{2}$, focusing on its most prominent feature as explained above. Finally, the last section comments on the use that we have been able to make of the assistant, as well as on forthcoming work. In comparison to [1], the present paper is more detailed in motivations and of course on the technical aspects concerning the novel extension to schematic First-Order Logic. We enumerate the features common to the predecessor tool Andy 0, giving only summarized descriptions.

2 Learning (Formal) Proof

Proof is the activity of arriving at knowledge deductively, i.e. starting off from self-evident, postulated or simply supposed principles and performing successive inferences, each of which extracts a conclusion out of previously arrived at premises. It constitutes the identifying feature of Mathematics, but is of course also of central importance in other disciplines, like Philosophy and Science generally. Students of all these branches of knowledge surely will benefit from endeavours directed to ease their path towards an increased understanding and mastery of the practice of proof. This should include students not only at college or university level, but also those in high school.

Formal Logic, on the other hand, is the *theory* of the activity of proving. As such it has, since the very beginning, striven to put forward the rules that

govern such activity, i.e. the rules of correct reasoning. In its contemporary mathematical variety, it has formulated several artificial languages into which to formulate the (deductive) practice of Mathematics. According to such scientific programme, there should be a language for expressing every conceivable mathematical proposition and also a language (or, as it has been called, proof system) for expressing proofs —of the true propositions.

It was Frege who first formulated such a device, in 1879 [3]. His intent was to complete the work of the foundational movement that had taken over Mathematics since the early XIX century. This movement was firstly directed towards providing a firm basis to the differential and integral calculus. Frege, in turn, intended to reach the most fundamental level by axiomatizing the very concept of proof. It is worhtwhile to spend some effort trying to understand the implications of such endeavour, for they are remarkable. Systems of axioms were already a established practice in Mathematics, e.g. in Geometry, and their accomplishment was to make explicit *the starting points* of the deductive activity. But of course each proof had still to be understood, individually. That is to say, the *logic* validating such proof was left implicit, as something that any mathematician (any human being with adequate training) would certainly possess. Now what Frege was pursuing was a manner of making such logic explicit. Hence, and this is the important consequence, *no logic* ought to be necessary to validate a proof as arranged in his system. That is to say, validation or rejection of purported proofs would be immediate by just *reading* them, i.e. what we now call a purely *formal* procedure. Frege could not put it in these terms, but his ideal amounted to providing a language for Mathematics whose correct use could be checked by an automaton. Moreover, such a system ought also to ensure that syntactic validity entailed logical validity, for a correctly written proof should be also logically correct. Hence such a language could be characterized in current computational jargon as one satisfying the lemma *if it compiles, it works*[1].

Formal proof systems are of course of prime importance to logicians, be they of the mathematical or the philosophical extraction. However, these scholars take towards them a stand that is generally external or "from above", i.e. they are mainly interested in the properties of the formal machinery as a whole, not in its actual use. They need only a basic dexterity and certainly not an efficient method for their exploitation. Now, on the other hand, it is a readily perceived fact that the use of formal proof in reasoning about the correctness of computer programs brings about the possibility of having *verifying compilers*, i.e. of programming systems in which syntactic correctness entails functional or behavioural correctness, and that thus realize for software the seemingly utopic motto *if it compiles, it works*. Several such systems have already been developed and constitute a toolkit for practitioners of the branch of Software Engineering called Formal Methods. The approach certainly requires to develop not just executable code, but also *mathematical code*, i.e. formally verifiable evidence of

[1] Unfortunately, Frege's system did not satisfy this property, as its logical principles were actually defective. But of course many other sound proof systems have been devised since then.

the correctness of the former. The cost of development is thus increased, but there is still certainly a pay off in many important cases, mainly as the cost of failure approaches infinity.

Thus Computing Science and Software Engineering do already provide an increasingly ample field of demand for adequate systems of formal proof together with appropriate technology for their efficient use. A useful tool for composing formal proofs should of course verify their correctness, but also offer appropriate assistance in their construction. It could be asked whether such assistance could get as far as to just the fully automatic proof of the desired theorems, but this is generally impossible already for the most basic branches of Mathematics that bear some interest. And, moreover, it is arguable not desirable in many cases, for proving is but understanding and therefore essential to the activity of the practitioner, be her a mathematician, a scientist or an engineer. Nevertheless, certainly not all proofs are equally interesting and one would expect effective technology to efficiently take care of the more clerical obligations, among other services. All this has then given rise to the interesting research field of *proof assistants* within Computing Science.

Now, a further interesting side of formal proof systems to which we would like to call attention is their potential to contribute to the understanding and mastery of proof, generally. As said in the beginning this could be of interest to a large audience of students, from high school to college, spanning a diverse range of disciplines. Certainly included among the latter is software construction, since it is a fact that understanding of program code and therefore certainty about its behavior amounts to nothing less than proof. Particularly useful in connection to this educational aspect of formal proof systems is the Calculus of Natural Deduction [4], devised with the aim of closely mimicking common practice in informal, natural proof. We proceed now to give a summary description of its most important features.

First, we shall take First-Order Logic as the formal language for expressing propositions. This is just standard, in the sense that any mathematician, when urged about the ultimate foundations of her activity may, and generally will, safely lean on Set Theory and its formalisation (encoding) in First-Order Logic. It is also quite natural, as its symbolism has permeated normal mathematical practice. For instance, $(\forall x)P(x)$ means "all individuals satisfy the property P" and $\alpha \to \beta$ means "the proposition α entails (implies) the proposition β". It is the fact that $\forall$ can only refer to the universe of individuals, and not to collections of propositions or properties, that is responsible for the "First-Order" part of the name of this language. $\forall$ is the universal *quantifier*, whereas $\to$ is a *connective* (specifically, the implication). Quantifiers and connectives are collectively named the *logical constants*. The letters $P, Q, \ldots$ used to represent properties form a *vocabulary* for a particular application, together with symbols for specific individuals and functions defined over the latter. The expressions representing propositions are the *formulas*, whereas those representing individuals are the *terms* of the language.

Now, as to the language or system of proofs, the system of Natural Deduction presents as characteristic feature the existence of rules that allow the use of temporary additional assumptions. For instance, to prove a formula that states an implication $\alpha \to \beta$ it is possible to use a rule that allows to assume α in order to prove β from it, just as in common practice. Specifically, this rule states that if one possesses a proof of β which depends on the assumption of α, then one can infer that $\alpha \to \beta$, thereby obtaining a proof which does not anymore depend on the aforementioned assumption. This assumption is said to be discharged, i.e. dispensed with in the newly formed proof.

Further, the rules of inference are organized around the logical constants and are of one of two classes in each case, i.e. a rule is either:

- An *introduction* rule stating how a formula having the logical constant in question as principal operator can be derived in a direct, canonical manner, or
- An *elimination* rule, stating how such a formula can be employed to derive further consequences from it.

For example, the rule described just above is the introduction rule for the implication $\to$. The elimination rule for this connective is:

$$\frac{\alpha \to \beta \qquad \alpha}{\beta}$$

which states that from $\alpha \to \beta$ one can infer β provided α is also proven. Another interesting rule is that of introduction of $\forall$: To conclude $(\forall x)\alpha$, where the formula α will in general depend on the variable x, it is enough to prove just α, provided that no assumption used in this latter proof depends on x. The proviso ensures that x represents an indeed *arbitrary* individual, i.e. one about which no particular property has been assumed. The elimination rule for $\forall$ makes use of the (meta-)operation of *substitution*:

$$\forall \mathsf{e}\ \frac{\forall x \alpha}{\alpha[x := t]}$$

Here $\forall \mathsf{e}$ is just the name of the rule, included for later reference. The conclusion of the rule says that the (universally) quantified variable can be just instantiated at any term (representing some individual). This is of course evidently correct, since we have just proven that the proposition α holds for any such individual. The instantiation is realized by the operation of substitution. The intent here is that this operation is defined outside the proof system and performed "silently". That is to say, it is assumed that the user knows how to perform it (because the explanation is given at some place during the setting up of the language of formulas) and, therefore, she just needs to execute it for applying this inference rule.

Rules have in general several premises and always one conclusion and therefore the formal proofs (technically called *derivations*) are naturally arranged as trees. It is natural to read inference as proceeding from the premises above to the

conclusion below, and therefore the trees are written with the root, which is the conclusion of the theorem, at the bottom, and the initial assumptions at the leaves on the top. Generally, the derivations prove judgements of the form $\alpha_1, \ldots, \alpha_n \vdash \alpha$ where the formulas to the left of $\vdash$ are the assumptions (or hypotheses) of the theorem and α its conclusion (thesis). The use of the rules inside a tree follows a quite characteristic pattern: Reading the tree from the top, one first applies elimination rules to obtain information from the given premises in a phase that could be called of *analysis*. At some level during the derivation one starts a *synthesis* forming new conclusions from the data obtained, by employing the introduction rules. The full system is well explained in [5].

So Natural Deduction actually consists in an orderly arrangement of rules of inference, where the organizing concept is the collection of logical constants. Indeed, with basis on the understanding of the meaning of each logical constant one can easily reconstruct the logical system of rules. That is to say that there is really no need of going through a painful recollection of an at first sight long list of inference rules when using the system. It is rather more convenient to *think* about the *meaning* of the formula at hand, and specifically about that of its principal logical constant. Further, the general structure of the proofs as explained earlier reflects the existence of a well founded strategy for selecting introduction or elimination rules at each step: At first, one goes backwards from conclusion to new conclusions-to-be-proven, generally by selecting appropriate introduction rules. This is carried out until no further rules of this kind can be applied or they would lead to conclusions which are not sensible to try. At such point one starts combining the available hypotheses in a forward manner, i.e. going from known premises to further inferred conclusions, trying to obtain the pending results. This is done by applying elimination rules. Hence the tree is generally constructed first from the root and upwards, until some point which roughly corresponds to the completion of the synthetic part of the proof as depicted above in the description of the general strcture of derivations. Then one proceeds from the assumptions and downwards, constructing the analytic part of the proof. Such order of proceeding looks somehow opposite to the one suggested by the analytic-synthetic structure of the proofs that has been mentioned earlier. But then one must keep in mind that the way a proof is presented or read is generally different from the way it is conceived. And also that the strategy just explained is not claimed to be the only one appropriate to employ but just a generally useful one, especially to start finding one's way about carrying out this kind of constructions.

We believe that all these features bear a favourable effect not only as to the mastery of the formal system, but also as to the understanding of mathematical propositions generally, especially as to what concerns the comprehension and development of their proofs.

3 Technology for Learning Formal Proof

What a student should learn about the use of Natural Deduction is not just to be able to come out with syntactically well formed derivations, but also to endow

each of the rules with *meaning*, which is to say that they carry out the proofs according to a well determined *intent*. Teaching, i.e. helping to learn, such mastery, conveniently begins with a detailed explanation of the system of rules, since it is important that the student gets to see the entire structure of the system, identifying the organizing principle. But it must quickly follow on promoting extensive practice. Such practice has to be, to the greatest possible extent, autonomous, because the students should not depend on the teacher's judgement for achieving certainty about the correctness of their constructions. Needless to say, this kind of practice is greatly favoured by the use of a computerized tool, which has given rise to the investigation of *didactic* proof assistants.

Clearly, the requirements on didactic proof assistants are different from those on those destined to professional use. Generally, it could be said that a student ought to be plainly in command of the proof process and only get help at those points where it does not stand in the place of understanding. More specifically, a number of tasks that are a nuisance for a professional constitute significant learning items for a student. A didactic assistant should first of all provide motivation. In this respect, a good deal is already achieved by the mere possibility of having the derivations constructed and checked. But there are also a number of other desired features that rapidly turn out to be necessary: The assistant should significantly improve over the paper-and-pencil experience, which means that it should allow:

1. Draft derivations to be initiated, discarded and connected at any moment.
2. Alternating backwards and forward derivation at will.
3. To manipulate the formalism at the right abstraction level.
4. Proof reuse, i.e. storage and instantiation of lemmas.

A detailed explanation of all these points has been given by the authors in [1]. However, the third point needs now some expansion, since the work cited reports on an assistant for the much simpler system of Natural Deduction for Propositional Logic. The point is that the right level of abstraction in using First-Order Logic is that of *schematic* formulas and theorems. The idea can be easily explained by the following claim: The theorems that one proves when teaching and learning First-Order Logic are *not* of the form $\forall x(P(x) \wedge Q(x)) \vdash \forall x P(x) \wedge \forall x Q(x)$, but of the form $\forall x(\alpha \wedge \beta) \vdash \forall x\alpha \wedge \forall x\beta$, i.e. they are formulated schematically (that is to say, generically) on formulas (and terms). This is due to the simple fact that the interesting results in First-Order Logic are independent of the particular predicate or function symbols employed, i.e. independent of any specific vocabulary. So one is naturally led to reason generically on formulas and terms, i.e. to work at the schematic level. It is important to remark that this *is* the normal practice, as witnessed by direct experience or by the contents of textbooks.

Moreover, we want the schematic theorems to be used directly in inference steps in other proofs. Specifically, each schematic theorem stands for an infinite family of theorems (its instances). We naturally wish to justify any of these by direct appeal to a schematic theorem of which it is an instance. This is equivalent to saying that (schematic) theorems can be used as rules of inference.

These latter considerations bring about a considerable technical problem, whose solution is explained below and turns our tool into a significant innovation in the field of assistants for First-Order Logic. The point is that the logical language is being modified, as can be inferred after consideration of the following examples ($\vee$ is the symbol for disjunction):

$$\forall x(\alpha \vee \beta) \vdash \alpha \vee \forall x\beta \quad (\alpha \text{ not depending on } x)$$

$$\forall x(\alpha \to \beta), \alpha[x := t] \vdash \beta[x := t].$$

The two preceding judgements are actual theorems that we wish to be able to derive using the assistant. They present:

1. Meta-variables (α and β ranging on formulas, t ranging on terms).
2. Side-conditions (restricting dependence on variables of certain formulas, that generally are or contain meta-variables).
3. Explicit substitution (since meta-variables are symbols standing for yet unknown formulas the substitution can no longer just silently operate on formulas or terms).

Already these observations lead to the necessity of formulating another language and proof system. In addition, a rule of *equality* of formulæ has to be available for explicit use, as in:

$$\dfrac{\dfrac{\forall x\alpha}{\alpha[x := x]}\,\mathsf{eq}}{\alpha.}\,\forall\mathsf{e}$$

Notice that one cannot skip the intermediate judgement $\alpha[x := x]$ because the substitution cannot be "silently" performed at the meta-level. Therefore the last step is necessary, in which the rule eq is applied. This allows to infer a formula from another if it can be checked that they are in fact equal modulo a theory that embodies the properties of the substitution together with the interchange of bound names. This rule is mostly left implicit in ordinary practice, i.e. it is used "silently", performing the necessary conversions "on the fly". But to do this correctly requires mastery of the properties of the language, so we think it advantageous that the rule has to emerge explicit in the new formal system.

The problem of formulating this new logic has been solved in [2] where *One-and-a-Halfth Order Logic* is introduced. It is precisely the logic of the schematic theorems of First-Order Logic as described above. One-and-a-Halfth Order Logic is founded in turn upon the syntax of Nominal Terms [6] which is a framework for languages with binding operators (e.g. the logical quantifiers) that incorporates meta-variables with explicit substitutions.

From a strictly technical point of view, the problem of handling schematic theorems could have been solved by stepping up to Second-Order Logic, for then a meta-variable ranging over formulæ can be represented as an object variable ranging over propositional functions. But this approach conducts to modifying the logic itself, i.e. we would be teaching Second-Order instead of First-Order Logic. We would rather stay at First-Order Logic or, as we should say, at the normal practice with First-Order Logic, which is actually the intermediate One-and-a-Halfth level.

4 The Design.

The design of Andy $1\frac{1}{2}$ rests upon the preceding considerations. Most of its features are similar to those of its ancestor Andy 0 and so we just summarize them here, giving also a comment about their presence in other proof assistants. We give some more detail on the new significant feature, which is the ability to deal with schematic First-Order logic.

1. Tree-like display of derivations. This is preferred because it is the notation that corresponds to writing rules of inference in the most natural way, i.e. with the premises on top of a bar which represents the act of inference of the conclusion appearing below. It is rather uncommon in elsewhere available systems.
2. Main proof and draft proofs. There exists a main proof together with draft subsidiary proofs which can be as many as desired. Draft proofs can be adjoined to the main proof by doing just *drag-and-drop*. Not known to be present in other proof assistants.
3. Backwards and forward reasoning. These have already been explained and can be alternated without restriction. They are pretty common in other proof assistants.
4. Library of Lemmas. This has also already been explained. Lemmas are generally schematic theorems and proper instantiation is carried out by the assistant, resting on the algorithms to be described below for the manipulation of the schematic formulas. This feature is rare in other assistants. At most they allow recovering lemmas but instantiation has to be done manually.

As already said, the main contribution of the present work consists in allowing to construct and check *schematic derivations*. For accomplishing this we have based ourselves on the already mentioned One-and-a-Halfth Order Logic. The formal system in [2] is a so-called Sequent Calculus, rather than a Natural Deduction system. We therefore have given it the latter formulation, proving it equivalent to the original one. As a consequence of this result, the logical soundness proven in [2] for the original system is inherited by ours. We also proved two basic properties of our system, which altogether justify the use of lemmas as inference rules. These are a so-called Generalized Cut and an Instantiation result that proves inference closed under substitution of formulas and terms for corresponding meta-variables.

Checking validity of rule application requires to check:

1. Side-conditions concerning the non-dependence of formulas on variables.
2. Equality of formulas, under laws that characterize (explicit) substitution and renaming of bound variables.

These checks are decidable and therefore performed by the assistant. Now, on didactic considerations, we have chosen not to do so silently, i.e. to force the student to explicitly command the assistant to perform the checks. In that way, she is permanently in command of the proof process. The corresponding algorithms

pertain to a module dealing with nominal abstract syntax which, as already mentioned, is the framework within which the present language is formulated.

Further, in order to implement the instantiation of lemmas it is necessary to *match* a schematic formula (the one to be proven) against another one (the conclusion of the chosen lemma, officiating as a pattern). Matching requires defining a criterion of identity of formulas, which within this language can be formulated in the following three manners:

1. Up to the laws of explicit substitution and renaming of bound variables.
2. Up to renaming of bound variables only.
3. Just as plain syntactic identity.

Our choice is for the latter, again based on didactic considerations: We want the student to explicitly indicate the need of renaming bound variables or using a susbtitution law. If the matching employed by the assistant were more powerful we would be saving the student from applications of the eq rule, already commented on. This is an example of an arguably clerical task, certainly uninteresting in professional practice, that bears some significance in educational terms. It is also worthwhile to mention that matching algorithms for the most powerful notion of formula equality above (number 1) are yet not known.

5 Conclusions

Andy $1\frac{1}{2}$ is then, to our knowledge, the first proof assistant to allow handling of schematic First-Order Logic, particularly within the calculus of Natural Deduction. It is also quite novel in offering the possibility of developing a library of reusable, instantiable lemmas, and maintains from its predecessor Andy 0, a graphical interface with proofs as trees, and several draft proofs that can be dragged and dropped into one another.

Currently it is being tested in seminars with advanced students. This experience is being used to adjust a number of details concerning especially the graphical interface. The system will be fully operating next semester in the course on Logic pertaining to the Software Engineering programme of our university. This is a standard course comprising Propositional and First-Order Logic. As reported in [1] the experience with Andy 0 is quite satisfactory as can be concluded from several interviews with the students involved. In particular, they were coincident in highlighting the motivating potential of the assistant, as it makes proof development and experimentation much more comfortable. We expect therefore a still better outcome from the experience with Andy $1\frac{1}{2}$, especially since it can be used in both parts of the course. Practice with actual formal proof takes two weeks within the 16 weeks of the entire course, but the students go back to it at the time of preparing the final tests. Our main concern is the permanent risk that the tool becomes the theory, in the sense that they take logical validity to be "Andy" validity. This is another manifestation of the purely formalistic attitude of composing proofs by just fitting trees into one another and missing the sense of the rules. We are also in the search for methodologies for properly assessing

the effect of this practice on the general understanding and handling of proofs by the students. Finally, we are currecntly planning to introduce programming practice into the course, posing the students problems of manipulation of formulas and proofs. It will be interesting to study the effect of this activity in their understanding of proof in the widest sense.

References

1. Pais, J., Tasistro, Á.: Proof Assistant Based on Didactic Considerations. Journal of Universal Computer Science 19(11), 1570–1596 (2013)
2. Gabbay, M.J., Mathijssen, A.: One-and-a-Halfth-Order Logic. Journal of Logic and Computation 18, 521–562 (2008)
3. Frege, G.: Beggriffsschrift, eine der aritmetischen nachgebildete Formelsprache des reinen Denkens. Louis Nebert, Halle A.S. (1879); Translated as Bauer-Mengelberg S.: Concept Script: A formal language of pure thought modeled upon that of Arithmetic. In: van Heijenoort, J. (eds.) From Frege to Gödel: A Source Book in Mathematical Logic (1879-1931). Harvard University Press, Cambridge (1967)
4. Gentzen, G.: Untersuchungen über das logische Schliessen. Matematische Zeitsschrift 39 (1935); Translated as Investigations into logical deduction. In: Szabo, M. (ed.) Collected Papers of Gerhard Gentzen. North Holland (1969)
5. van Dalen, D.: Logic and Structure. Springer (2008)
6. Urban, C., Pitts, A.M., Gabbay, M.J.: Nominal Unification. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 513–527. Springer, Heidelberg (2003)