Literature survey on implementation techniques for type systems

Exploring name binding techniques

Hasan Kocakaya[1]

Supervisor(s): Jesper Cockx[1]**, Bohdan Liesnikov**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Hasan Kocakaya
Final project course: CSE3000 Research Project
Thesis committee: Jesper Cockx, Bohdan Liesnikov, <Examiner>

Abstract

Names are essential for structuring and reasoning about programs. However, the implementation of names differs across many programming languages. There is an abundance of choice between various implementation techniques with regards to name-binding techniques. As such, when designing a programming language it is not clear which technique one should choose. This paper attempts to give an exhaustive overview of the various techniques that exist, compares them on properties such as alpha-equivalence, ease of implementation and enforcing well-scopedness. Furthermore, the possibility of a one-fits-all solution is explored.

1 Introduction

Name binding is a crucial feature of type systems, and type systems are a crucial component of programming languages [21]. Name binding refers to the concept of binding identifiers to the associated entities(functions,variables,types) [23]. Thus, depending on the manner of implementation of name binding, a given program could have different behaviours. There are a variety of different techniques to implement name binding. Unfortunately, there isn't a technique that is considered the best, nor is there a consensus on which technique should be used.

Name binding is closely related to scoping. An identifier can be resolved to different associated entities depending on the scope it is resolved from. Given that this is a fundamental part of programming languages, it is valuable to do further research on this topic and explore all avenues available. Future languages, and already existing languages can benefit from having a clear overview of the currently available techniques, such that informed choices can be easily made with regards to name-binding techniques. In this literature survey paper we attempt to provide an overview of the currently existing research on name-binding and compare the proposed techniques, identifying their strengths and weaknesses.

Some surveys have already been done, however these surveys were not the main focus of the papers, and have therefore not attempted to be exhaustive [2; 1; 4; 22].

An example of a name-binding technique is de Bruijn indices [8]. In the domain of lambda calculus it can be used to represent terms without naming the bound variables. This concept can be used to implement name-binding by using nameless encoding [4] which has been used to solve common issues that are encountered when implementing name binding. It works as follows: a reference to a name-binding receives an index equal to the amount of bindings that are present in-between the reference and binding location. Thus, traversing an amount of steps equal to the index is how the reference can be resolved. This removes the need for having a string representation of the name.

Since the aim of this paper is to give a high level overview of the currently known name-binding techniques, details such as proofs will not be discussed, as that is beyond the scope of this paper. To give an effective overview of all the currently available implementation techniques in the literature, the following research questions are answered in the paper:

- What are the different techniques that exist to implement name binding?
- What are the advantages and disadvantages of these techniques?
- Is there a technique that can be identified as being superior?

The paper is structured as follows. First the research methodology is discussed in section 2. Then the general concept of name binding is introduced and various techniques that have been proposed are explored and discussed in section 3. Subsequently the techniques are analyzed and compared, as well as the comparison metrics are discussed in section 4. Then in section 5 scientific integrity and the applied research practices is discussed in the context of responsible research. In section 6 the findings of the comparisons are discussed, as well as advice for language implementers is discussed with regards to the findings. Finally the paper is concluded and future research is discussed in section 7.

2 Methodology

The set of papers that are considered in this survey were collected as follows: First a look was taken at the most common name binding technique, de Bruijn indices. Because most if not all papers mention de Bruijn, additional techniques were easy to identify from within the list of papers that cite this paper.

The grouping method for implementation techniques also follows from this method, as some papers of interest that were identified already made clear categorizations and groupings. Some groupings are: named, unnamed, first-order approaches, higher-order approaches, nominal approaches. These groupings are used and added on to with the discovered techniques.

The comparison dimensions were determined by looking at the common properties of techniques discussed in the found papers. For example, a technique being invariant under alpha-equivalence is an important notion mentioned in every single technique. Therefore, it was considered to be of the utmost importance and selected as one of the main dimensions for comparison.

3 Implementation Techniques

In this section, a general description of name-binding is given. Then the various techniques with which name-binding can be implemented are listed and explained.

3.1 Name-Binding

In programming languages, name-binding is the process of associating 'identifiers', which can be seen as names, with their respective values or bindings. It's the process of establishing the link between a name and what it represents, allowing programmers to refer to specific entities of a program using their given names.

When writing code, names are used to represent variables, functions, classes, and other elements of a program. These

names facilitate referring to these elements throughout a program. Name binding ensures that when a name is used, it points to the right underlying binding. A name cannot always be resolved, as the binding for a specific name might not have been created, or is not accessible. And thus, there are many different ways in which this association can be implemented.

Name binding techniques can be categorized in several ways. A distinction can be made between First-Order approaches and Higher-Order approaches. Another way to distinguish them is to consider approaches where names are present, approaches where names are abstracted away and approaches where a combination of the two are applied.

3.2 de Bruijn Indices

De Bruijn indices provide a way to implement name-binding without relying on variable names. Instead, variables are represented using numeric indices that encode their binding information. When using de Bruijn indices, variables are represented by numbers that indicate the number of binders between the reference to the binder and its binding site. An index of 0 represents that no binding is in between the binding reference and its binding site. This numeric system represents the relative positions of these bindings. By using de Bruijn indices, we eliminate the need for variable names and represent all binding information using numeric indices relative to their binders. This approach simplifies the manipulation and analysis of lambda terms, as it avoids issues related to variable capture, renaming, and conflicts. It also enables efficient operations such as substitution, scoping, and checking for alpha equivalence by directly encoding the binding structure within the indices. For example, substitution does not require renaming operations, since there are no names present. To substitute, the operations that are performed are simply the traversal and updating of indices.

3.3 Locally named

The essence of the locally named [1; 13] technique is to make a distinction between a bound and unbound variable. This representation of syntax eliminates the difficulties associated with reasoning about capture-avoiding substitutions. By making a distinction between (bound) variables and (free) parameters in syntax, the occurrence of a parameter being captured by a variable binder during substitution is completely prevented. Additionally, because both variables and parameters are represented as names, it is a very human readable format.

3.4 Locally nameless

The locally nameless [1; 3] representation of syntax with binding is a technique which combines de Bruijn indexes with names. The fundamental idea is to make use of two seperate syntactic classes to represent variables depending on whether they are bound or free. Bound variables are represented using de Bruijn indices, and free variables are represented using names. An advantage of this approach is that it combines the best of both named and de Bruijn indices. By using the named syntactic class to display names for human readability, and by using de Bruijn indices for internal manipulations, substitutions and other operations.

3.5 Well-scoped de Bruijn

Well-scoped de Bruijn syntax [18; 10; 17] refers to using de Bruijn indices to represent name-binding. However an additional constraint is added, such that bindings that are not in scope cannot be referenced. This is done by introducing a set of bound variables. Any element not in the set of bound variables, is not in scope. Thus, this representation is well-scoped.

3.6 Higher-Order Abstract Syntax

Higher-Order Abstract Syntax (HOAS) [15; 9] refers to using the abstraction rules present in the host/meta-language to represent the binders in the object language. By doing so, this allows the underlying host language type system to help catch potential errors during the development process.

3.7 Nominal Logic

Nominal logic [16; 19] is a technique that takes a different approach to name-binding. Using nominal sets, a mathematical theory of atomic names, with properties invariant under permuting names [6]. It uses the concept of ‘atoms‘ which are singular syntactic units. An operation exists to swap two atoms. An operation also exists to determine the ‘freshness‘ of an atom, to determine whether the name of that atom is unique. Using these concepts, it is possible to reason about names and binders, their freshness and equality between atoms.

3.8 Nameless Painless

The nameless painless [17] technique is an approach that makes use of de Bruijn indices, however some additional restrictions apply. A ‘world‘ would represent the scope. In Pouillard’s implementation, a world is represented as a list of boolean values. To determine whether a variable is within scope, a simple lookup of the nth value in the list would return either true or false, indicating whether it is in scope or not. This concept allows for simple reasoning about variable binding and scoping, due to offering a systematic way to distinguish between scopes.

3.9 Scope Graphs

Scope graphs are a different way to think about the name-binding structures of a program. Instead of reasoning about names, bindings and environments based on abstract syntax trees; scope graphs allow us to reason about the structure of a program based on graphs [14]. ‘In scope graphs, nodes represent scopes and declarations, which are connected by labeled edges. References are resolved by finding paths to eligible declarations, subject to visibility and shadowing policies expressed in terms of edge labels.

Up till now, techniques have all defined name-binding based on lexical scoping. Lexical scoping refers to using the program’s structure to determine scoping rules. However, for features such as imports and class inheritance, using non-lexical scoping is simpler.

Using this formalism, many different (non-lexical) binding patterns can be encoded‘ [23, p. 32:2]. Scope graphs focus on non-lexical binding, in particular this facilitates a more

straightforward method to implement name-binding for imports and class inheritance [20].

3.10 Hypergraphs

Name-binding with hypergraphs is a technique which has been implemented to represent name binding using a generalization of graphs [22]. To represent name-binding in a hypergraph, each vertex/node corresponds to a specific name, while the hyperedges represent the references or associations between names and their respective entities. A hyperedge connects a set of vertices/nodes, indicating that the names within that set are bound together. This allows for more flexible and expressive name-binding relationships, as multiple names can be bound simultaneously.

3.11 Co-de Bruijn indices

Another variant of de Bruijn indices is the Co-de Bruijn [11] variant. It is a nameless representation, however with additional constraints. In this variant, unused variables are discarded. The scope is reduced to a minimal version, where only variables that actually occur within an expression are present. By doing so, it avoids unnecessary shifting of indices.

4 Comparison of implementation techniques

In this section the dimensions on which the implementation techniques are compared are listed and expanded upon. Also the selection criteria for the comparison dimensions themselves are given. The advantages and disadvantages of each technique listed in section 3 are discussed.

4.1 Selection of dimensions

The dimensions to compare on are selected while keeping in mind the limitations of a literature survey. Therefore aspects that are mentioned in the original papers, as well as easily identifiable properties of name-binding techniques are selected for the comparison.

4.2 invariance under alpha-equivalence

Alpha-equivalence [7] is useful when two expressions only differ in the names of bound variables, while maintaining the same syntactic structure and scope. In such cases, alpha equivalence facilitates comparison on the similarity of the expressions' structure instead of the specific names associated with variables. The concept of alpha equivalence is essential as it allows us to reason about expressions without being concerned about variable naming. By treating alpha-equivalent expressions as equivalent, it allows for reasoning about the structure and meaning of expressions, while excluding irrelevant details such as the arbitrarily chosen variable names.

Alpha equivalence is important when talking about capture-avoiding substitution. When a substitution of a variable occurs in an expression, it is critical to avoid variable capture, which is when a free variable is unintentionally captured by a bound variable.

Alpha equivalence facilitates the comparison of expressions while disregarding the naming of the bound variables. It allows reasoning about expressions based on syntactical structure and behavior.

4.3 Ease of implementation

Whether an implementation exists of a particular technique is an important factor to consider. For example, it shows whether actually implementing a technique is feasible. Additionally the lack of such an implementation being present could point to potential short-comings of the technique. Furthermore it is important to distinguish between theoretical techniques and techniques that have been successfully implemented in practice.

4.4 Enforces well-scopedness

Well-scopedness is an important aspect for name-binding techniques. If a technique guarantees intrinsic well-scopedness, it means that it is impossible to use variables outside of their scope. It saves effort during the development of a programming language if the name-binding technique already ensures this.

4.5 de Bruijn Indices

Basic De Bruijn indices enforce alpha-equivalence. Since it is one of the most fundamental techniques used in nameless representations, an existing implementation is present. This basic form of De Bruijn does not enforce well-scopedness, thus ill-scoped terms can be constructed.

This name-binding technique is a very basic one, and is therefore easy to implement. However, it is difficult to reason about due to its index-based nature. Another drawback of de Bruijn indices is the need for constant bookkeeping and index shifting when introducing new bound variables.

4.6 Locally named

The locally named representation does not enforce alpha-equivalence, as the names of free variables matter [13]. Locally named has been used in 'Pure Type Systems' (PTS), for formal meta-theory. This representation does not enforce well-scopedness.

The main advantage of this technique is the ease of differentiating between bound and unbound variables. Another is that, at some level, the name of a variable must be present in one way or another, whether that be internally or for parsing/printing. It should be noted that this technique was mainly developed with metatheory in mind, so it remains unknown whether this technique is practical for more traditional programming languages.

4.7 Locally nameless

The locally nameless representation uses 2 seperate syntactic classes for free and bound variables. It does enforce alpha-equivalence, but does not enforce well-scopedness. The locally nameless representation has been used in practice, for example Epigram [12] a dependently-typed language has used the locally nameless representation in its implementation.

This technique combines the human readability of names, and the convenience of de Bruijn indices for internal manipulations. Another benefit is that by representing global variables and constants as names, environments can be implemented in a simplistic manner [3].

4.8 Well-scoped de Bruijn

Well-scoped de Bruijn is similar to regular de Bruijn in the aspect of how alpha-equivalence is treated. Therefore, well-scoped de Bruijn is stable under alpha-equivalence. Because well-scoped de Bruijn is implemented by adding additional constraints on de Bruijn in the form of a scope or 'world'. This increases the difficulty of the implementation, however the pay-off is an intrinsically well-scoped name-binding technique.

Overall this technique is a more refined variant of the regular de Bruijn technique, with a slight cost in implementation difficulty, but an added feature of well-scopedness.

4.9 Higher-Order Abstract Syntax

Because HOAS uses the binding constructs of the host language [5], if the host language is stable under alpha-equivalence, so will the object language. For the same reason, the object language is well-scoped, as the host languages scoping rules are applied on the object language.

As for the ease of implementation, being able to re-use the host language binding constructs makes it potentially convenient to use. However, the drawback is that HOAS is mainly useful for doing metatheory, and less so for programming language design [1].

4.10 Nominal Logic

Nominal logic is stable under alpha-equivalence. it is not inherently well-scoped. Nominal logic is not as simple to implement as other techniques that rely on de Bruijn. As Yasen said 'the formalization of name binding with swapping and freshness constraints, which are the fundamental part of the nominal logic, seems somewhat difficult to understand for non-experts' [22, p. 1139].

4.11 Nameless Painless

The nameless painless approach is stable under alpha-equivalence. Additionally, the technique also enforces well-scopedness. The nameless painless approach is implemented as a library, written in Agda [17]. Because of this, making use of the nameless painless approach using the library is rather simple.

4.12 Scope Graphs

The scope graph technique is not stable under alpha-equivalence. However, the technique does enforce well-scopedness, because there must exist a path between a reference and the declaration of the reference within a graph.

This technique is complicated to implement. However, there exists implementations and case studies [20]. The advantage of this technique is that it also allows for non-lexical scoping, it facilitates simpler implementation of imports and the technique attempts to standardize the treatment of name-binding in programming languages.

4.13 Hypergraphs

The hypergraphs technique is not stable under alpha-equivalence. However, the technique does enforce well-scopedness. This technique has only been implemented in 'HyperLMNtal', and requires familiarity with the language and the graph type 'hlground' [22].

A major downside of this technique is that it is not efficient yet. Yasen mentions that it is planned as future work to improve this inefficiency and to improve upon this technique.

4.14 Co-de Bruijn indices

The co-de Bruijn indices technique is stable under alpha-equivalence. It is also well-scoped. Co-de Bruijn indices are difficult to work with, as they are very unintuitive and 'unsuited to human comprehension'. The main disadvantage of this technique is the complexity, but the advantage that comes with it is the precision and minimal scopes.

5 Responsible Research

In this section the ethics and repeatability of this paper are discussed. Since this paper does not use any experimental data, nor human test subjects, the focus will mostly be on academic integrity.

An effort has been made to not misrepresent the original papers cited in this literature survey, such that misinformation is not spread, nor the authors' words twisted. Additionally all direct quotes are attributed to their original papers, and indirect quotes are paraphrased sufficiently such that they deviate adequately from the original work.

6 Discussion

In section 4 the various techniques implementation techniques for name-binding are compared on the metrics such as ease of implementation and stability under alpha-equivalence. Each technique has both advantages and disadvantages. There seems to be no obvious one-size-fits-all solution. For example if simplicity and ease of implementation is the focus, one might find that a de Bruijn implementation of name-binding is sufficient. If however the focus lies on implementing various types of imports and class inheritance, scope graphs should be opted for.

The advice to programming language implementers would therefore be to assess the situation and requirements of the programming language they are implementing, and make an informed choice based on the comparisons in this paper.

There are limitations to this literature survey. This research paper attempts to be exhaustive, however this is not guaranteed and new developments should be considered.

Another limitation is the paper is descriptive, and not critical of the work presented. Therefore, the correctness of the contents of the paper are limited to what the papers volunteer as information.

Finally, the comparison dimensions selected in this paper provide some insights on the strengths and weaknesses of the techniques. However they may not capture the full range of considerations for evaluating name-binding techniques.

7 Conclusion and Future work

In this paper, the currently existing name-binding techniques are explored, discussed and a comparison was made on several aspects. The advantages and disadvantages of each tech-

nique are explored, and the research question 'Is there a technique that can be identified as being superior?' is answered.

This paper attempts to compare all currently existing name-binding techniques. Since these techniques vastly differ on their approach to name-binding, they are sometimes not easily comparable. It is difficult to say in which cases, if at all, a certain technique is more favourable to choose for a potential programming language. It is also the case that these name-binding techniques, and their respective papers use different setups, languages, workbenches or frameworks to achieve their goal.

Therefore an interesting future work might look towards implementing all these name-binding techniques within the same language. By doing so, external factors are isolated, and the true merits of the name-binding technique can be explored. A vastly superior comparison would be possible in such a study. Currently in this paper, a comparison is made on the claims and theory of the techniques, without any real practical comparison.

Undoubtedly it is also important to look at the practical aspect, because the main purpose of a name-binding technique is to be used in practice, as part of a type-system for a programming language.

References

[1] Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 3–15, San Francisco California USA, January 2008. ACM.

[2] J.-P. Bernardy and N. Pouillard. Names for free - polymorphic views of names and binders. pages 13–24, 2013.

[3] Arthur Charguéraud. The Locally Nameless Representation. *Journal of Automated Reasoning*, 49(3):363–408, October 2012.

[4] James Cheney and Christian Urban. Prolog: A Logic Programming Language with Names, Binding and -Equivalence. In Bart Demoen and Vladimir Lifschitz, editors, *Logic Programming*, Lecture Notes in Computer Science, pages 269–283, Berlin, Heidelberg, 2004. Springer.

[5] Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceedings of the 13th ACM SIGPLAN international conference on Functional programming*, pages 143–156, Victoria BC Canada, September 2008. ACM.

[6] Ranald A. Clouston and Andrew M. Pitts. Nominal Equational Logic. *Electronic Notes in Theoretical Computer Science*, 172:223–257, April 2007.

[7] Roy L. Crole. Alpha equivalence equalities. *Theoretical Computer Science*, 433:1–19, May 2012.

[8] N.G De Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

[9] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 193–202, Trento, Italy, 1999. IEEE Comput. Soc.

[10] Daniel R. Licata and Robert Harper. A universe of binding and computation. In *Proceedings of the 14th ACM SIGPLAN international conference on Functional programming*, pages 123–134, Edinburgh Scotland, August 2009. ACM.

[11] Conor McBride. Everybody's Got To Be Somewhere. *Electronic Proceedings in Theoretical Computer Science*, 275:53–69, July 2018.

[12] Conor Mcbride and James Mckinna. The view from the left. *Journal of Functional Programming*, 14(1):69–111, January 2004.

[13] James McKinna and Robert Pollack. Pure type systems formalized. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, volume 664, pages 289–305. Springer-Verlag, Berlin/Heidelberg, 1993. Series Title: Lecture Notes in Computer Science.

[14] Pierre Neron, Andrew Tolmach, Eelco Visser, and Guido Wachsmuth. A Theory of Name Resolution. In Jan Vitek, editor, *Programming Languages and Systems*, volume 9032, pages 205–231. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. Series Title: Lecture Notes in Computer Science.

[15] F. Pfenning and C. Elliott. Higher-order abstract syntax. *ACM SIGPLAN Notices*, 23(7):199–208, July 1988.

[16] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2):165–193, November 2003.

[17] Nicolas Pouillard. Nameless, painless. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, pages 320–332, Tokyo Japan, September 2011. ACM.

[18] Nicolas Pouillard and François Pottier. A fresh look at programming with names and binders. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 217–228, Baltimore Maryland USA, September 2010. ACM.

[19] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1-3):473–497, September 2004.

[20] Hendrik Van Antwerpen, Casper Bach Poulsen, Arjen Rouvoet, and Eelco Visser. Scopes as types. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–30, October 2018.

[21] Larisse Voufo, Marcin Zalewski, and Andrew Lumsdaine. Scoping rules on a platter: a framework for understanding and specifying name binding. In *Proceedings of the 10th ACM SIGPLAN workshop on Generic*

programming, pages 59–70, Gothenburg Sweden, August 2014. ACM.

[22] Alimujiang Yasen and Kazunori Ueda. Name Binding is Easy with Hypergraphs. *IEICE Transactions on Information and Systems*, E101.D(4):1126–1140, 2018.

[23] Aron Zwaan and Hendrik van Antwerpen. Scope Graphs: The Story so Far. In Ralf Lämmel, Peter D. Mosses, and Friedrich Steimann, editors, *Eelco Visser Commemorative Symposium (EVCS 2023)*, volume 109 of *Open Access Series in Informatics (OASIcs)*, pages 32:1–32:13, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 2190-6807.