# Capture-Avoiding Program Transformations with *name-fix*

Sebastian Erdweg[1] Tijs van der Storm[2,3] Yi Dai[4]

[1] TU Darmstadt, Germany [2] CWI, Amsterdam, The Netherlands
[3] INRIA Lille, France [4] University of Marburg, Germany

**Abstract:** We present an algorithm called name-fix that automatically eliminates variable capture from a generated program by systematically renaming variables. name-fix is guided by a graph representation of the binding structure of a program, and requires name-resolution algorithms for the source language and the target language of a transformation. name-fix is generic and works for arbitrary transformations in any transformation system that supports origin tracking for names.

Program transformations find ubiquitous application in compiler construction to realize desugarings, optimizers, and code generators. While traditionally the implementation of compilers was reserved for a selected few experts, the current trend of domain-specific and extensible programming languages exposes developers to the challenges of writing program transformations. In our paper [EvdSD14], we address one of these challenges: capture avoidance.

A program transformation translates programs from a source language to a target language. In doing so, many transformations reuse the names that occur in a source program to identify the corresponding artifacts generated in the target program. This bears the danger of variable capture where variables from the source program are captured by synthesized variable declarations or vice versa. In a study of current language workbenches, we found that developers in 8 out of 9 workbenches failed to address variable capture when implementing a simple domain-specific language [EvdSD14]. However, a general solution is difficult to obtain. Existing approaches either rely on naming conventions and fail to guarantee capture avoidance, or they require a specific transformation engine and affect the implementation of transformations [SB99, SPG03, LE13].

We propose a generic solution called *name-fix* that guarantees capture avoidance and does not affect the implementation of transformations. *name-fix* compares the name graph of the source program with the name graph of the generated program to identify variable capture. If there is variable capture, *name-fix* systematically and globally renames variable names to differentiate the captured variables from the capturing variables, while preserving intended variable references among original variables and among synthesized variables, respectively. *name-fix* requires name analyses for the source and target languages, which often exists or are needed anyway (e.g., for editor services, error checking, or refactoring), and hence can be reused. *name-fix* treats transformations as a black box and is independent of the used transformation engine as long as it supports origin tracking for names [VvdSE14, vDKT93].

*name-fix* enables developers of transformations to focus on the actual translation logic and to ignore variable capture. In particular, *name-fix* enables developers to use simple naming schemes for synthesized variables in the transformation and to produce intermediate open terms. Transformations of this kind fall into the class of transformations for which

$name\text{-}fix$ guarantees hygiene, that is, $\alpha$-equivalent source programs are always mapped to $\alpha$-equivalent target programs.

Our current definition of $name\text{-}fix$ renames not only synthesized names but also names that originate from the source program. This may break the expected interface of the generated code. Accordingly, $name\text{-}fix$ currently is a whole-program transformation that does not support linking of generated programs against previously generated libraries, because names in these libraries cannot be changed. In order to apply $name\text{-}fix$ in the context of our extensible programming language SugarJ [Erd13, ER13], we currently investigate a modular variant of $name\text{-}fix$ that supports separate compilation.

Finally, the current implementation of $name\text{-}fix$ requires repeated execution of the name analysis of the target language. As a result, $name\text{-}fix$ can be expensive in terms of runtime performance. When a compiler is run continuously in an IDE, this penalty can be an impediment to usability. Fortunately, incrementality [MEK+14] and in particular incremental name analysis [RTD83, WKV+13] are well-studied topics that are likely to yield benefits for $name\text{-}fix$ because (i) we know the delta induced by $name\text{-}fix$ (renamed variables) and (ii) new variable capture can only occur in references that have changed.

## References


[ER13] Sebastian Erdweg and Felix Rieger. A Framework for Extensible Languages. In *GPCE*, pages 3–12. ACM, 2013.

[Erd13] Sebastian Erdweg. *Extensible Languages for Flexible and Principled Domain Abstraction*. PhD thesis, Philipps-Universiät Marburg, 2013.

[EvdSD14] Sebastian Erdweg, Tijs van der Storm, and Yi Dai. Capture-Avoiding and Hygienic Program Transformations. In *ECOOP*, pages 489–514. Springer, 2014.

[LE13] Florian Lorenzen and Sebastian Erdweg. Modular and Automated Type-Soundness Verification for Language Extensions. In *ICFP*, pages 331–342. ACM, 2013.

[MEK+14] Ralf Mitschke, Sebastian Erdweg, Mirko Köhler, Mira Mezini, and Guido Salvaneschi. i3QL: Language-Integrated Live Data Views. In *OOPSLA*, pages 417–432. ACM, 2014.

[RTD83] Thomas Reps, Tim Teitelbaum, and Alan Demers. Incremental context-dependent analysis for language-based editors. *TOPLAS*, 5(3):449–477, 1983.

[SB99] Yannis Smaragdakis and Don S. Batory. Scoping Constructs for Software Generators. In *GCSE*, volume 1799 of *LNCS*, pages 65–78. Springer, 1999.

[SPG03] Mark R. Shinwell, Andrew M. Pitts, and Murdoch J. Gabbay. FreshML: Programming with Binders Made Simple. In *ICFP*, pages 263–274. ACM, 2003.

[vDKT93] Arie van Deursen, Paul Klint, and Frank Tip. Origin tracking. *Symbolic Computation*, 15:523–545, 1993.

[VvdSE14] Pablo Inostroza Valdera, Tijs van der Storm, and Sebastian Erdweg. Tracing Model Transformations with String Origins. In *ICMT*, pages 154–169. Springer, 2014.

[WKV+13] Guido Wachsmuth, Gabriël D. P. Konat, Vlad A. Vergu, Danny M. Groenewegen, and Eelco Visser. A Language Independent Task Engine for Incremental Name and Type Analysis. In *SLE*, volume 8225 of *LNCS*, pages 260–280. Springer, 2013.