Reviewers: Please see the "Note to Reviewers" on page 14.

Toward a More Carefully Specified Metanotation

author information omitted

for double-blind reviewing

omitted.email.address@nodomain

Abstract

POPL is known for, among other things, papers that present formal descriptions and rigorous analyses of programming languages. But an important language has been neglected: the *metanotation* of inference rules and BNF that has been used in over 40% of all POPL papers to describe all the other programming languages. This metanotation is not completely described in any one place; rather, it is a folk language that has grown over the years, as paper after paper tries out variations and extensions. We believe that it is high time that the tools of the POPL trade be applied to the tools themselves.

Examination of many POPL papers suggests that as the metanotation has grown, it has diversified to the point that problems are surfacing: different notations are in use for the same operation (substitution); the same notation is in use for different operations; and in some cases, notations for repetition are ambiguous, or require the reader to apply knowledge of semantics to interpret the syntax. All three problems present substantial potential for confusion. No individual paper is at fault; rather, this is the natural result of language growth in a community, producing incompatible dialects.

We back these claims by presenting statistics from a survey of all past POPL papers, 1973–2016, and examples drawn from those papers. We propose a set of design principles for metanotation, and then propose a specific version of the metanotation that can be always interpreted in a purely formal, syntactic manner and yet is reasonably compatible with past use. Our goal is to lay a foundation for complete formalization and mechanization of the metanotation.

Categories and Subject Descriptors D.3.1 [*Formal Definitions and Theory*]: Syntax; F.3.1 [*Specifying and Verifying and Reasoning about Programs*]: Assertions; F.4.3 [*Formal Languages*]

Keywords assertion, BNF, context-free grammar, ellipses, inference rule, judgment, macro, macro expansion, metanotation, nested repetition, overline notation, repetition, substitution, template

1. Introduction

What is the most popular programming language at POPL? Not C, nor Java, nor even Haskell, but the "POPL metanotation" consisting of *inference rules* plus the data description language *BNF*—and, make no mistake, this metanotation is indeed a programming language, or could be, if properly formalized and mechanized. But it is not completely described in any one place; rather, it is a folk language that has grown over the years, as one paper after another tries out small variations and extensions. Each paper largely assumes metanotational conventions established by past papers as customary and explicitly describes only a few features of specific interest.

But after four decades of this, we now face three problems:

(1) So far, 27 different notations for substitution have been used in POPL papers, and 14 of them are still in current use.
(2) Many of those same notations are also used for other purposes.
(3) In some cases, repetition notation is ambiguous, or requires the reader to apply knowledge of semantics to interpret the syntax.

All three problems present substantial potential for confusion.

In this paper, we back these claims by presenting summary and detailed statistics from a survey of all past POPL papers. We also present and explain examples drawn from those papers. We propose a set of design principles for metanotation, and then propose a specific version of the metanotation that can be interpreted in a purely formal, syntactic manner and yet is reasonably compatible with past use (and where it cannot be compatible, we explain why).

In §2, we present the results of our survey. In §3, we analyze the data and discuss difficulties with the metanotation, with examples drawn from past POPL papers. In §4, we present design principles for metanotation and novel extensions to the existing metanotation intended to address the observed difficulties. In §5, we present a careful specification of a complete metanotation and a formal explanation of how to expand it. In §6, we present examples of the use of this extended metanotation. In §7, we present recommendations to future authors of POPL papers and suggest future work.

The specific novel contributions of this paper are:

(1) A survey of the use of inference rules, substitution notations, and repetition notations in POPL papers, 1973–2016.
(2) A careful specification and formal interpretation of a complete metanotation that includes inference rules, BNF, substitution notation, and repetition notations, including:
    (a) certain common uses of ellipses to indicate repetition, and
    (b) the widely used overline notation that indicates repetition.
(3) Use of underlines in the overline notation to suppress indices.
(4) Use of harpoons as overlines to indicate repetition without separating punctuation and to notate certain kinds of recursive expression heretofore expressible only by using two ellipses.

2. Survey of POPL Papers

We examined manually (well, ocularly) every page of every paper of every past POPL, 1973–2016 (there was no POPL conference in 1974, so that is 43 volumes). There were 1,401 papers, a total of 17,160 pages. We examined a paper copy of each volume, working in chronological order; when we identified a paper of interest then we pulled it up onto a screen from the ACM Digital Library so that we could search the text for such words as "substitution" (actually, "subst"). While the OCR is good (and gets better over time), it does not always distiguish certain pairs of symbols such as $\rightarrow$ and $\mapsto$, and it does not capture at all the horizontal lines that indicate inference rules and repetition notation. Fortunately such horizontal lines are easy to spot even when flipping through pages quickly.

$e\frac{v}{x}$	1	[P77A]	$e[v/x]$	133	[P79D]–[P16Y]	$e(v/x)$	1	[P78B]
$[v/x]e$	67	[P77B]–[P16S]	$e[{}^{v}/x]$	6	[P96O]–[P07I]	$e\{v/x\}$	25	[P90A]–[P16W]
$[{}^{v}/x]e$	1	[P97G]	$e[{}^{v}/_{x}]$	2	[P13J]–[P15L]	$e\{{}^{v}/x\}$	5	[P01G]–[P16V]
$[x := v]e$	2	[P09F]–[P10S]	$e[v\backslash x]$	1	[P14T]	$e\{{}^{v}/_{x}\}$	4	[P03D]–[P16L]
$[x \mapsto v]e$	9	[P94M]–[P16B]	$e[x/v]$	5	[P89C]–[P15Y]	$e\{x \leftarrow v\}$	4	[P88D]–[P95D]
$[x \to v]e$	1	[P08A]	$e[x := v]$	21	[P88G]–[P16B]	$e\{x \mapsto v\}$	1	[P02B]
$[\![v/x]\!]e$	2	[P08V]–[P12Y]	$e[x \leftarrow v]$	7	[P89E]–[P11F]	$e\{x \to v\}$	1	[P02K]
$\{v/x\}e$	6	[P86D]–[P15P]	$e[x \mapsto v]$	12	[P94D]–[P15S]	$e\{\!\vert{}^{v}/x\vert\!\}$	2	[P98R]–[P99G]
$\{x \mapsto v\}e$	4	[P95B]–[P16Q]	$e[x \to v]$	2	[P12Σ]–[P15Θ]	$e\{\!\{x \leftarrow v\}\!\}$	1	[P04G]

Table 1. Twenty-seven substitution notations and the number of POPL papers in which they were observed, also citing the earliest and latest. Prefix notations are shown right-justified, and postfix notations are shown left-justified. Eight different notations were used in 2016 alone.

At first we tried to identify every paper that used either inference rules, repetition notation, or substitution notation. By about 1982 it became clear that most papers using either repetition or substitution notation also contained inference rules, so from that point on, with only one or two exceptions that happend to catch our eye, we examined carefully only papers that contained at least one inference rule. During such careful examination, we scanned the entire paper, not just the inference rules themselves, to try to find uses of repetition notation or substitution notation. By "repetition notation" we mean either the use of an ellipsis, such as "$x_1, \ldots, x_n$", or the use of an iterator notation, such as "$\{x_i\}^{i\in 1..n}$", or the use of an overbar or overarrow or other such symbol to indicate either (1) repetition of a symbol or syntax fragment, or (2) a vector of elements of which only one representative is shown. Example of this last category are $\overline{x}$, $\overrightarrow{x}$, $\widetilde{x}$, and $e[\overline{v/x}]$. By "substitution notation" we mean an explicit three-argument notation such as $[v/x]e$ (not merely an application such as σe of a substitution denoted by σ to an expression e) that is intended to represent the standard capture-free substitution of (a copy of) v (or, more typically, an expression denoted by the metavariable v) for every free occurrence of a variable x (or a variable denoted by the metavariable x) within the expression e (or an expression denoted by the metavariable e).

Of the 1,401 papers, we identified 609 for careful examination. For each paper, we recorded an actual example of the substitution notation used (if any) as well as a schematic template using the three variables v, x, and e so that the specific symbols used in the notation could be easily identified. We also recorded whether inference rules were used (because of our sampling bias, nearly all did, but we wanted to have an accurate count of how many papers out of the 1,401 use inference rules), and whether and how those inference rules were labeled. We recorded whether ellipses, iterators, and/or over-symbols were used to indicate repetition; for over-symbol notation, we recorded the precise symbol used, whether explicit index variables and/or iterators were used as part of the over-symbols notation, whether the over-symbol was used only over single symbols or over larger syntax fragments, and whether the over-symbols were ever stacked or nested.

We recorded these details in a BIBTEX database, and then used a custom `bst` file to generate various distillations of the data, some of which were then pulled into a spreadsheet for further analysis. The data presentations in Tables 1, 2, and 3 were generated semi-automatically form this database (additional TEX commands were hand-inserted for formatting purposes).

For purposes of graphing the data, we divide time into lustra (five-year intervals), working backward from the present, producing nine bins of five conferences each, except that the first bin has just the first POPL three conferences (1973, 1975, and 1976).

2.1 Notations for Substitution

We found that (at least) 27 distinct notations for three-argument substitution have been used at POPL; these are summarized in Table 1. We found substitution notation in 327 of the 609 papers we examined closely; of these, 4 [P08V, P09E, P12Y, P16B] used two different kinds of substitution notation, so we have a total of 331 data points. All 331 data points are listed in Table 2; each paragraph is one year's worth of data.

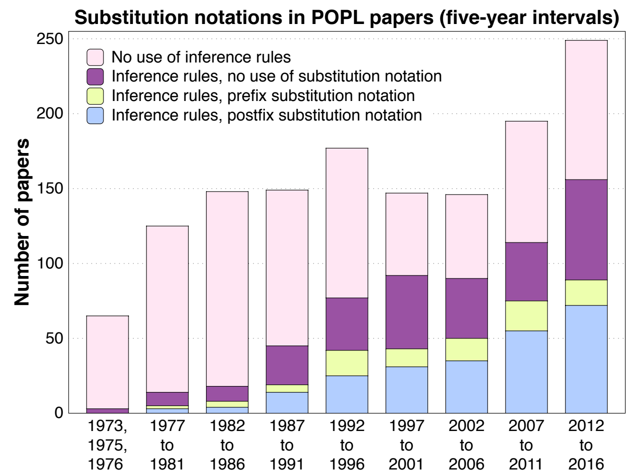

Figure 1. Substitution notations in POPL papers, 1973–2016

With the exception of the very earliest one, namely $e\frac{v}{x}$, all these notations can be characterized along five axes: (a) are they prefix (v and x occur to the left of e) or postfix (v and x occur to the right e)? (b) What symbols enclose v and x? (c) What symbol(s) separate v from x? (d) Does v occur to the left of x, or to the right of x? (e) Are v and/or x on the baseline, or are one or both raised or lowered? Not all possible combinations occur.

By far the most commonly used notation is $e[v/x]$ (postfix, brackets, slash, v before x, baseline): 133 out of 331. The second most common notation is $[v/x]e$ (prefix, brackets, slash, v before x, baseline): 67 out of 331. Two others, $e\{v/x\}$ and $e[x := v]$, were used more than 20 times each. Each of the 23 other notations observed was used in fewer than 10 papers.

Figure 1 shows the usage of inference rules and substitution notation in POPL papers within each lustrum, and furthermore breaks down usage of prefix and postfix notations. Observations: (1) At first only a small percentage of papers presented inference rules, but their use increased sharply after 1986, and over the last 20 years roughly 60% of POPL papers in each lustrum have presented inference rules. (2) Of papers that present inference rules, over half also use substitution notation. (3) Early on, prefix and postfix notations were used with roughly equal frequency, but after 1986 the relative percentage of postfix notations climbed sharply.

Figure 2 shows the breakdown of enclosers within the 331 data points for papers that use substitution notation. Plain brackets [ ] have always been the most commonly used encloser symbols, and

Each of these notations is intended to represent the result of capture-free substitution of v for all occurrences of the variable x within expression e. Twenty-seven different styles of notation are presented in this table. Each example is followed by a bibliographic citation (see the list of references); 327 papers are cited out of 1,401 papers that appeared in POPL from 1973 through 2016, and 4 of them [P08V, P09E, P12Y, P16B] each used two different notations. Papers using substitution notation were not found in years 1973, 1975, 1976, 1980, 1981, or 1984. We may have missed a few; we focused primarily on papers that also use inference rule notation.

$e\frac{v}{x}$ [P77A], $[v/x]e$ [P77B]

$e(v/x)$ [P78B]

$[v/x]e$ [P79B], $e[v/x]$ [P79D]

$[v/x]e$ [P82A]

$e[v/x]$ [P83B], $e[v/x]$ [P83C]

$[v/x]e$ [P85B], $[v/x]e$ [P85D], $e[v/x]$ [P85E]

$\{v/x\}e$ [P86D], $e[v/x]$ [P86E]

$[v/x]e$ [P87G]

$[v/x]e$ [P88A], $e[v/x]$ [P88B], $e\{x \leftarrow v\}$ [P88D], $e[x := v]$ [P88G], $e[x/v]$ [P89C], $e[v/x]$ [P89D], $e[x \leftarrow v]$ [P89E], $e[x \leftarrow v]$ [P89F]

$e\{v/x\}$ [P90A], $e[v/x]$ [P90B], $e[v/x]$ [P90D], $e[v/x]$ [P90I], $[v/x]e$ [P90J], $[v/x]e$ [P90K], $e[x \leftarrow v]$ [P90M]

$e[x := v]$ [P91F], $[v/x]e$ [P91J], $e[v/x]$ [P91K]

$e\{v/x\}$ [P92D], $e[v/x]$ [P92E], $e[v/x]$ [P92H], $[v/x]e$ [P92I], $\{v/x\}e$ [P92K]

$e[v/x]$ [P93C], $e[v/x]$ [P93E], $e[v/x]$ [P93F], $e[x := v]$ [P93H], $[v/x]e$ [P93I], $e\{x \leftarrow v\}$ [P93J], $e[v/x]$ [P93K], $[v/x]e$ [P93L], $e[x/v]$ [P93N], $[v/x]e$ [P93P], $e\{v/x\}$ [P93Q]

$[v/x]e$ [P94A], $[v/x]e$ [P94B], $e[x \mapsto v]$ [P94D], $e\{x \leftarrow v\}$ [P94F], $[v/x]e$ [P94G], $e[x := v]$ [P94H], $e[v/x]$ [P94I], $e[v/x]$ [P94K], $[x \mapsto v]e$ [P94M], $e[v/x]$ [P94P], $e[x := v]$ [P94R]

$[v/x]e$ [P95A], $\{x \mapsto v\}e$ [P95B], $[v/x]e$ [P95C], $e\{x \leftarrow v\}$ [P95D], $[v/x]e$ [P95H], $e[v/x]$ [P95K]

$[v/x]e$ [P96C], $[v/x]e$ [P96D], $[v/x]e$ [P96H], $e[v/x]$ [P96I], $e[v/x]$ [P96L], $e\{v/x\}$ [P96M], $[v/x]e$ [P96N], $e[{}^{v}\!/x]$ [P96O], $e[v/x]$ [P96Q]

$e[v/x]$ [P97B], $e[v/x]$ [P97C], $[v/x]e$ [P97D], $e[x := v]$ [P97E], $e[v/x]$ [P97F], $[{}^{v}\!/x]e$ [P97G], $[v/x]e$ [P97H], $e[x \leftarrow v]$ [P97L], $e[v/x]$ [P97Q], $\{x \mapsto v\}e$ [P97S], $[v/x]e$ [P97T]

$e[v/x]$ [P98A], $e[v/x]$ [P98E], $e[v/x]$ [P98H], $e[v/x]$ [P98J], $[v/x]e$ [P98K], $e[x \mapsto v]$ [P98M], $e[v/x]$ [P98Q], $e\{\!|{}^{v}\!/x|\!\}$ [P98R]

$e[x := v]$ [P99B], $[v/x]e$ [P99C], $e[{}^{v}\!/x]$ [P99D], $e[x \leftarrow v]$ [P99F], $e\{\!|{}^{v}\!/x|\!\}$ [P99G], $e[v/x]$ [P99H], $e[x := v]$ [P99I], $e[v/x]$ [P99K], $e[x := v]$ [P99L], $e[v/x]$ [P99O]

$e[x \mapsto v]$ [P00G], $e[v/x]$ [P00I], $\{x \mapsto v\}e$ [P00K], $e[v/x]$ [P00N]

$e[{}^{v}\!/x]$ [P01A], $e[v/x]$ [P01B], $[v/x]e$ [P01C], $e\{{}^{v}\!/x\}$ [P01G], $[v/x]e$ [P01I], $[v/x]e$ [P01J], $e[v/x]$ [P01K], $\{v/x\}e$ [P01M], $e[x/v]$ [P01P], $e\{v/x\}$ [P01Q]

$e\{v/x\}$ [P02A], $e\{x \mapsto v\}$ [P02B], $e\{v/x\}$ [P02D], $[v/x]e$ [P02G], $e[x \mapsto v]$ [P02H], $[v/x]e$ [P02I], $e[v/x]$ [P02J], $e\{x \rightarrow v\}$ [P02K], $e[x \leftarrow v]$ [P02L], $[v/x]e$ [P02M]

$[v/x]e$ [P03B], $e[x := v]$ [P03C], $e\{{}^{v}\!/_{x}\}$ [P03D], $e[x \mapsto v]$ [P03E], $e[x := v]$ [P03F], $e[v/x]$ [P03G], $e[v/x]$ [P03I], $e[v/x]$ [P03J], $e[x \mapsto v]$ [P03K], $e[v/x]$ [P03L]

$[v/x]e$ [P04B], $e[v/x]$ [P04C], $e[{}^{v}\!/x]$ [P04D], $e[{}^{v}\!/x]$ [P04E], $[x \mapsto v]e$ [P04F], $e\{\!\{x \leftarrow v\}\!\}$ [P04G], $e\{v/x\}$ [P04I], $[v/x]e$ [P04J], $e[v/x]$ [P04L], $e[v/x]$ [P04M], $[v/x]e$ [P04Q], $e[v/x]$ [P04R]

$[v/x]e$ [P05A], $e[v/x]$ [P05D], $[x \mapsto v]e$ [P05E], $[v/x]e$ [P05F], $[v/x]e$ [P05G], $e\{v/x\}$ [P05I], $[v/x]e$ [P05J], $e[x \mapsto v]$ [P05N], $e[v/x]$ [P05O], $e[v/x]$ [P05Q]

$e[v/x]$ [P06A], $e[v/x]$ [P06G], $[v/x]e$ [P06H], $e[x := v]$ [P06N], $e[v/x]$ [P06R], $e[v/x]$ [P06S], $[v/x]e$ [P06U], $e[v/x]$ [P06V]

$e[v/x]$ [P07B], $e[v/x]$ [P07C], $e[v/x]$ [P07D], $e[v/x]$ [P07F], $e\{v/x\}$ [P07H], $e[{}^{v}\!/x]$ [P07I], $[v/x]e$ [P07J], $e[v/x]$ [P07K], $e\{v/x\}$ [P07N], $e[v/x]$ [P07R]

$[x \rightarrow v]e$ [P08A], $e[v/x]$ [P08D], $e[v/x]$ [P08E], $e[v/x]$ [P08F], $[v/x]e$ [P08G], $e[v/x]$ [P08H], $e[v/x]$ [P08I], $e[x := v]$ [P08J], $e[v/x]$ [P08K], $e[v/x]$ [P08N], $[v/x]e$ [P08O], $e[v/x]$ [P08Q], $e[v/x]$ [P08R], $[v/x]e$ [P08V], $[\![v/x]\!]e$ [P08V], $e[v/x]$ [P08W], $e[x/v]$ [P08X]

$e[v/x]$ [P09A], $e[v/x]$ [P09D], $e[v/x]$ [P09E], $e\{v/x\}$ [P09E], $[x := v]e$ [P09F], $e[v/x]$ [P09H], $\{v/x\}e$ [P09I], $[v/x]e$ [P09K], $e[v/x]$ [P09L], $[v/x]e$ [P09O], $e[v/x]$ [P09P], $[v/x]e$ [P09Q], $[x \mapsto v]e$ [P09R], $[v/x]e$ [P09S]

$e[x \mapsto v]$ [P10C], $e[x \mapsto v]$ [P10E], $e[v/x]$ [P10G], $e[v/x]$ [P10H], $e[v/x]$ [P10I], $e[v/x]$ [P10J], $e[x \mapsto v]$ [P10K], $e[v/x]$ [P10L], $e[v/x]$ [P10M], $e\{v/x\}$ [P10O], $[x \mapsto v]e$ [P10P], $e\{{}^{v}\!/_{x}\}$ [P10Q], $e[x := v]$ [P10R], $[x := v]e$ [P10S], $e[x \mapsto v]$ [P10U], $e[v/x]$ [P10Y], $[v/x]e$ [P10Z]

$[x \mapsto v]e$ [P11B], $e[x := v]$ [P11C], $[x \mapsto v]e$ [P11D], $e[x \leftarrow v]$ [P11F], $e[x := v]$ [P11G], $\{v/x\}e$ [P11H], $e[x \mapsto v]$ [P11I], $e[v/x]$ [P11J], $e[v/x]$ [P11L], $e\{v/x\}$ [P11M], $e[v/x]$ [P11N], $e[v/x]$ [P11O], $e[v/x]$ [P11S], $e\{v/x\}$ [P11T], $[v/x]e$ [P11U], $e\{v/x\}$ [P11W], $e[v/x]$ [P11Z]

$e[v/x]$ [P12A], $[v/x]e$ [P12C], $e[v/x]$ [P12D], $e[v/x]$ [P12E], $e[x \mapsto v]$ [P12F], $e\{v/x\}$ [P12G], $e[x \mapsto v]$ [P12I], $e[v/x]$ [P12K], $e[v/x]$ [P12M], $e[x \mapsto v]$ [P12Q], $e[x \mapsto v]$ [P12S], $e[v/x]$ [P12T], $e[v/x]$ [P12U], $e[v/x]$ [P12V], $e[v/x]$ [P12W], $e[v/x]$ [P12X], $[v/x]e$ [P12Y], $[\![v/x]\!]e$ [P12Y], $e[x := v]$ [P12Π], $e[x \rightarrow v]$ [P12Σ], $e[v/x]$ [P12ϒ]

$e[v/x]$ [P13A], $e[v/x]$ [P13C], $[v/x]e$ [P13D], $e[v/x]$ [P13E], $e[v/x]$ [P13F], $e[v/x]$ [P13I], $e[{}^{v}\!/_{x}]$ [P13J], $e[v/x]$ [P13K], $e[v/x]$ [P13N], $e[v/x]$ [P13O], $e\{v/x\}$ [P13P], $e[v/x]$ [P13Q], $e[v/x]$ [P13T], $e\{v/x\}$ [P13U], $e\{v/x\}$ [P13V], $e[v/x]$ [P13X], $e[v/x]$ [P13Y], $[x \mapsto v]e$ [P13Γ], $[v/x]e$ [P13Δ]

$e\{{}^{v}\!/_{x}\}$ [P14A], $e[v/x]$ [P14B], $[v/x]e$ [P14C], $e[v/x]$ [P14D], $e[x := v]$ [P14L], $e\{{}^{v}\!/x\}$ [P14N], $e[v/x]$ [P14O], $[v/x]e$ [P14R], $e[v\backslash x]$ [P14T], $e[v/x]$ [P14W], $e[v/x]$ [P14X], $[v/x]e$ [P14Y], $e\{v/x\}$ [P14Z], $e[v/x]$ [P14Γ], $e[v/x]$ [P14Δ], $e[v/x]$ [P14Σ], $e[v/x]$ [P14ϒ], $e\{v/x\}$ [P14Φ], $e[v/x]$ [P14Ψ]

$[v/x]e$ [P15B], $e[v/x]$ [P15E], $e[v/x]$ [P15H], $e\{v/x\}$ [P15I], $e[{}^{v}\!/_{x}]$ [P15L], $[v/x]e$ [P15M], $e\{{}^{v}\!/x\}$ [P15N], $\{v/x\}e$ [P15P], $e\{v/x\}$ [P15Q], $e[v/x]$ [P15R], $e[x \mapsto v]$ [P15S], $e\{{}^{v}\!/x\}$ [P15T], $e[x := v]$ [P15X], $e[x/v]$ [P15Y], $e[x \rightarrow v]$ [P15Θ], $[v/x]e$ [P15Π]

$e[x := v]$ [P16A], $[x \mapsto v]e$ [P16B], $e[x := v]$ [P16B], $[v/x]e$ [P16D], $e[v/x]$ [P16G], $e[v/x]$ [P16K], $e\{{}^{v}\!/_{x}\}$ [P16L], $\{x \mapsto v\}e$ [P16Q], $[v/x]e$ [P16S], $e[v/x]$ [P16T], $e\{{}^{v}\!/x\}$ [P16V], $e\{v/x\}$ [P16W], $e[v/x]$ [P16X], $e[v/x]$ [P16Y]

Table 2. Substitution notations used in POPL papers (mostly considering only papers that also use inference rules; see text), 1977–2016

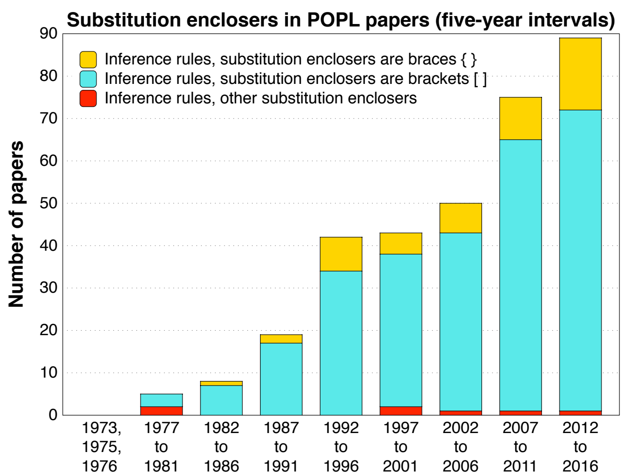

Figure 2. Substitution enclosers in POPL papers, 1973–2016

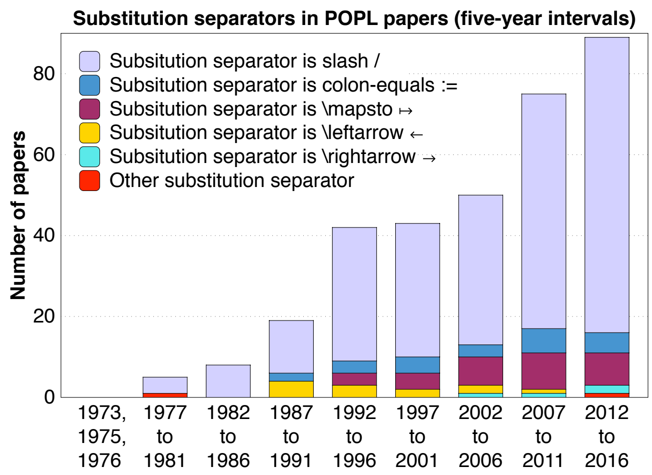

Figure 3. Substitution separators in POPL papers, 1973–2016

for the last lustrum were used in 80% of papers that used substitution notation, but plain braces { } may be making a comeback: their percentage of use was 19% in 1992–1996, then dipped to 13% for the next three lustra, but for the last lustrum is back to 19%.

Figure 3 shows the breakdown of separators within the same 331 data points. The slash character "/" has always dominated, and its percentage for the most recent lustrum is over 80% (73 of 89 papers). But ":=" came into use after 1986, and likewise "↦" after 1991; over the last ten years, roughly one paper per year has used ":=" and almost two papers per year have used "↦". Out of 327 papers, 12 have used "←" and 4 have used "→"; just 1 paper [P14T] has used the backslash "\", and of course one paper [P77A] used the idiosyncratic form "$e\frac{v}{x}$".

Of the 327 papers using substitution notation, 100 used it to express multiple simultaneous substitution of two or more values for two more corresponding variables. Of those, 16 used braces { } or double braces {{ }} as enclosers; the other 84 used brackets [ ].

In many (but not all) papers that use multiple simultaneous substitution, repetition notation is used as a part of substitution notation; examples are $S[y_1/x_1, \ldots, y_n/x_n]$ [P85E, §4.3], $p[\vec{q}/\vec{x}]$ [P90D, §4.1], $[\overline{\beta_n/\alpha_n}]\tau$ [P93P, Fig. 6], $P\{\vec{z}/\vec{x}\}$ [P93Q, §2], $[\overline{x} \mapsto \overline{t'}]t$ [P94M, §1.4], $E[\overline{z/y}]$ [P96L, §1], $[\tilde{y}/\tilde{z}]P$ [P96N, §4.2], $\Gamma\{\vec{x} \mapsto \vec{y}\}$ [P02B, §2.2], $\{\overline{\alpha \mapsto \phi'}\}(\phi_1)$ [P16Q, Fig. 3].

For comparison, we examined several well-known books and monographs. Church used $\mathrm{S}^x_N M|$ to "stand for the formula which results by substitution of N for x throughout M" [3, p. 9]. Barendregt used $M[x := N]$ for the same purpose [1, §2.1], and Bruce used $[N/x]M$ [2, p. 128]. Gunter (reversing the use of M and N) used $\{M/x\}N$ to mean a substitution of M for x within N that does *not* avoid variable capture, and then immediately used it to define $[M/x]N$ to mean a substitution of M for x in N that *does* avoid variable capture (by using α-conversion as appropriate) [5, pp. 35–37]. Winskel used $A[a/i]$ to represent the result of substituting a for i within A [13, pp. 82–83]. Reynolds wrote, "$p/v \to e$ denotes the result of substituting e for v in p (after replacing these metavariables by particular phrases)" [9, p. 18]; while we otherwise admire his book, we feel that this particular choice of notation is ill-advised because in the book, as here, the extra space around the arrow visually makes p and v bind tightly to the slash, making it seems as if p/v, not just v, somehow maps to or becomes e.

2.2 Notations for Repetition

Out of 609 papers selected for careful examination, 184 used ellipses to indicate repetition and 174 used some form of overline to indicate repetition. There was overlap: 57 papers used both ellipses and overlines. We observed exactly three forms of overline: $\overline{\text{overbar}}$ (used in 94 papers), $\overrightarrow{\text{overarrow}}$ (used in 60 papers), and $\widetilde{\text{tilde}}$ (used in 20 papers). Our impression was that almost every paper that used tilde to indicate repetition addressed either the π-calculus or bisimulation (π-calculus notation uses overbars for another purpose).

Table 3 contains an entry for each of the 609 papers; each paragraph is one year's worth of data.

Figure 4 shows the usage of inference rules and substitution notation in POPL papers within each lustrum, and furthermore breaks down usage of overline and ellipsis notations. Its vertical axis matches that of Figure 1 and so those two figures may be compared directly. Observations: (1) Over the last 15 years, of papers that present inference rules, over half also use some repetition notation. (2) Repetition notations were rarely used before 1992, but after that their use increased sharply. (3) From 1992 to 2006, ellipses were used much more frequently than overline notations, but in the last ten years overline notations have come to dominate slightly.

Figure 5 shows the breakdown of kinds of overline (tilde, overarrow, or overbar) within just the 174 papers that used overline notation for repetition. Over the last 20 years, an average of one paper per year has used tilde. During 1997–2001 and 2007–2011, the numbers of papers using overarrow and overbar were roughly the same; during 2002–2006 and 2012–2016, the number of papers using overbar was about twice the number using overarrow.

Of the 174 papers that used overline notation, 27 (15.5%) used overline notation over syntax fragments rather than just single symbols. Of these, 4 used overarrows [P06Q, P11S, P12Y, P16I] and 23 used overbars [P05A, P05S, P06H, P08M, P10J, P10O, P11I, P12D, P12M, P12Q, P12Π, P13P, P13T, P13Y, P14B, P14E, P14Ξ, P14Ψ, P15G, P15I, P15N, P16B, P16Y].

Of the 174 papers that used overline notation, just 13 (7%) used explicit index variables in conjunction with the overline notation [P05U, P09H, P10J, P10O, P11I, P12Z, P13V, P15G, P15I, P15M, P16B, P16R, P16Y]. While the yearly number of papers using this combination is small, it has been increasing. Of these 13 papers, 3 also used an explicit iterator notation as part of the overline notation; we quote an example or two from each: $\overline{K_i \Rightarrow e_i}^{\,i}$ [P11I], $\overline{C_i \parallel D_i : T_i}^{\,i}$ and $\overline{C_i}^{\,i\in\{1,\ldots,n\}}$ [P15I], $\overline{C_i\,x \to e_i}^{\,i\in m}$ [P16Y].

This table contains bibliographic citations to 609 POPL papers, each decorated to indicate whether the paper contains inference rules and whether it uses a repetition metanotation. Brackets [ ] around a citation indicate the presence of at least one inference rule; parentheses ( ) indicate no inference rule was seen. If a paper uses some sort of overline notation, then such a notation appears above the citation. Only three distinct forms of overline were observed: overbar, right-pointing overarrow, and tilde. (While collecting data, we made no distinction between the tiny overarrow produced by `\vec` and the larger overarrow produced by `\overrightarrow`, nor between the tiny tilde `\tilde` and the larger tilde `\widetilde`.) If a paper uses overlines only over monograms, then an overbar or `\vec` or `\tilde` is shown over the first character "P" of the citation, and if it uses dual overlines then two overlines are shown; on the other hand, if a paper uses overlines over larger syntax fragments, then an overbar or `\overrightarrow` or `\widetilde` is shown over the entire citation, and if it uses nested overlines, then a second overline appears over just the middle two characters. If a paper uses explicit subscripted index variables in conjunction with the overline notation, then a subscripted i appears. If a paper uses an explicit iterator notation in conjunction with the overline notation, then a superscripted i appears. Independently of whether a paper uses an overline notation, if it uses an ellipsis, then three dots appear beneath the citation, and if it uses an iterator notation, then a superscripted $\forall$ appears (but the paper itself may or may not use the "$\forall$" symbol in its iterator notation).

(P75A) (P75B)
(P76A)
[P77A] (P77B)
[P78A] [P78B] [P78C]
(P79A) [P79B] [P79C] [P79D] [P79E]
[P80A]
(P81A) (P81B) [P81C]
[P82A] [P82B]
[P83A] [P83B] [P83C$^\forall$]
[P84A] [P84B] [P84C]
[P85A] [P85B] [P85C] [P85D] [P85E]
[P86A] [P86B] (P86C) [P86D] [P86E]
[P87A] [P87B] [P87C] [P87D] [P87E] [P87F] [P87G]
[P88A] [P88B] [P88C] [P88D] (P88E) [P88F] [P88G]
[P89A] [P89B] [P89C] [P89D] [P89E] [P89F]
[P90A] [P90B] [P90C] [P90D] [P90E] [P90F] [P90G] [P90H] [P90I] [P90J] [P90K] [P90L] [P90M]
[P91A] [P91B] [P91C] [P91D] [P91E] [P91F] [P91G] [P91H] [P91I] [P91J] [P91K] [P91L]
[P92A] [P92B] [P92C] [P92D] [P92E] [P92F] [P92G] [P92H] [P92I] [P92J] [P92K]
[P93A] [P93B] [P93C] [P93D] [P93E] [P93F] [P93G] [P93H] [P93I] [P93J] [P93K] [P93L] [P93M] [P93N] [P93O] [P93P] [P93Q] [P93R]
[P94A] [P94B] [P94C] [P94D] [P94E] [P94F] [P94G] [P94H] [P94I] [P94J] [P94K] [P94L] [P94M] (P94N$_i$) [P94O] [P94P] [P94Q] [P94R]
[P95A] [P95B] [P95C] [P95D] [P95E] [P95F] [P95G] [P95H] [P95I] [P95J] [P95K] [P95L] [P95M]
[P96A] [P96B] [P96C] [P96D] [P96E] [P96F] [P96G] [P96H] [P96I] [P96J] [P96K] [P96L] [P96M] [P96N] [P96O] [P96P$^\forall$] [P96Q]
[P97A] [P97B$^\forall$] [P97C] [P97D] [P97E] [P97F] [P97G] [P97H] [P97I] [P97J] [P97K] [P97L] [P97M] [P97N] [P97O] [P97P] [P97Q] [P97R] [P97S$^\forall$] [P97T]
[P98A] [P98B] [P98C] [P98D] [P98E] [P98F] [P98G] [P98H$^\forall$] [P98I] [P98J] [P98K] [P98L] [P98M] [P98N] [P98O] [P98P] [P98Q] [P98R] [P98S]
[P99A] [P99B] [P99C] [P99D] [P99E] [P99F] [P99G] [P99H] [P99I] [P99J] [P99K] [P99L] [P99M] [P99N] [P99O$^\forall$] [P99P] [P99Q]
[P00A] [P00B] [P00C] [P00D] [P00E] [P00F] [P00G] [P00H] [P00I] [P00J] [P00K] [P00L] [P00M] [P00N] [P00O] [P00P] [P00Q] [P00R] [P00S]
[P01A] [P01B] [P01C] [P01D] [P01E] [P01F] [P01G] [P01H] [P01I] [P01J] [P01K] [P01L] [P01M] [P01N] [P01O] [P01P] [P01Q]
[P02A] [P02B] [P02C] [P02D] [P02E] [P02F] [P02G] [P02H] [P02I] [P02J] [P02K] [P02L] [P02M]
[P03A] [P03B] [P03C] [P03D] [P03E] [P03F] [P03G] [P03H] [P03I] [P03J] [P03K] [P03L] [P03M]
[P04A] [P04B] [P04C] [P04D] [P04E] [P04F] [P04G] [P04H] [P04I] [P04J] [P04K] [P04L] [P04M] [P04N] [P04O] [P04P] [P04Q] [P04R] [P04S]
[P05A] [P05B] [P05C] [P05D$^\forall$] [P05E] [P05F] [P05G] [P05H] [P05I] [P05J] [P05K] [P05L] [P05M] [P05N$^\forall$] [P05O] [P05P] [P05Q] [P05R] [P05S] [P05T] [P05U$_i$]
[P06A] [P06B] [P06C] [P06D] [P06E] [P06F] [P06G] [P06H] [P06I] [P06J] [P06K] [P06L] [P06M] [P06N] [P06O$^\forall$] [P06P] [P06Q] [P06R] [P06S] [P06T] [P06U] [P06V$^\forall$] [P06W] [P06X]
[P07A] [P07B] [P07C] [P07D] [P07E] [P07F] [P07G] [P07H] [P07I] [P07J] [P07K] [P07L] [P07M$^\forall$] [P07N] [P07O] [P07P] [P07Q] [P07R]
[P08A] [P08B] [P08C] [P08D] [P08E] [P08F] [P08G$^\forall$] [P08H] [P08I] [P08J] [P08K] [P08L] [P08M] [P08N] [P08O] [P08P] [P08Q] [P08R] [P08S] [P08T] [P08U] [P08V] [P08W] [P08X] [P08Y]
[P09A] [P09B] [P09C] [P09D] [P09E] [P09F] [P09G] [P09H$_i$] [P09I] [P09J] [P09K] [P09L] [P09M] [P09N] [P09O] [P09P] [P09Q] [P09R] [P09S]
[P10A] [P10B] [P10C] [P10D] [P10E] [P10F] [P10G] [P10H] [P10I] [P10J$_i$] [P10K] [P10L] [P10M] [P10N] [P10O$_i$] [P10P] [P10Q] [P10R] [P10S] [P10T] [P10U] [P10V] [P10W] [P10X] [P10Y] [P10Z]
[P11A] [P11B] [P11C] [P11D] [P11E] [P11F] [P11G] [P11H] [P11I$_i^i$] [P11J] [P11K] [P11L] [P11M] [P11N] [P11O] [P11P] [P11Q] [P11R] [P11S] [P11T$^\forall$] [P11U] [P11V] [P11W] [P11X] [P11Y] [P11Z]
[P12A] [P12B] [P12C] [P12D] [P12E] [P12F] [P12G] [P12H] [P12I] [P12J] [P12K] [P12L] [P12M$^\forall$] [P12N] [P12O] [P12P] [P12Q] [P12R] [P12S] [P12T] [P12U] [P12V] [P12W] [P12X] [P12Y] [P12Z$_i$] [P12Γ] [P12Δ$^\forall$] [P12Θ] [P12Λ] [P12Ξ] [P12Π$^\forall$] [P12Σ] [P12Υ]
[P13A] [P13B] [P13C] [P13D] [P13E] [P13F$^\forall$] [P13G] [P13H] [P13I] [P13J] [P13K] [P13L] [P13M] [P13N] [P13O] [P13P] [P13Q] [P13R] [P13S] [P13T] [P13U] [P13V$_i$] [P13W] [P13X] (P13Y) [P13Z] [P13Γ] [P13Δ]
[P14A$^\forall$] [P14B] [P14C] [P14D] [P14E] [P14F] [P14G] [P14H] [P14I] [P14J] [P14K] [P14L] [P14M] [P14N] [P14O] [P14P$^\forall$] [P14Q] [P14R] [P14S] [P14T] [P14U] [P14V] [P14W] [P14X] [P14Y] [P14Z$^\forall$] [P14Γ] [P14Δ] [P14Θ] [P14Λ] [P14Ξ] [P14Π] [P14Σ] [P14Υ] [P14Φ] [P14Ψ]
[P15A] [P15B] [P15C] [P15D] [P15E] [P15F] [P15G$_i$] [P15H$^\forall$] [P15I$_i^i$] [P15J] [P15K] [P15L$^\forall$] [P15M$_i$] [P15N] [P15O] [P15P] (P15Q) [P15R] [P15S] [P15T$^\forall$] [P15U] [P15V] [P15W] [P15X] [P15Y] [P15Z] [P15Γ] [P15Δ] [P15Θ] [P15Λ] [P15Ξ] [P15Π] [P15Σ]
[P16A] [P16B$_i$] [P16C] [P16D] [P16E] [P16F] [P16G] [P16H] [P16I$^\forall$] [P16J] [P16K] [P16L] [P16M] [P16N] [P16O] [P16P] [P16Q] [P16R$_i$] [P16S] [P16T$^\forall$] [P16U] [P16V] [P16W] [P16X$^\forall$] [P16Y$_i^i$]

Table 3. Repetition notations used in POPL papers (mostly considering only papers that also use inference rules; see text), 1977–2016

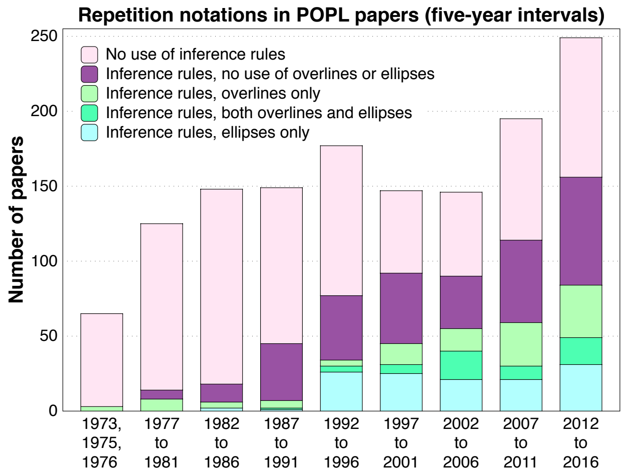

Figure 4. Repetition notations in POPL papers, 1973–2016

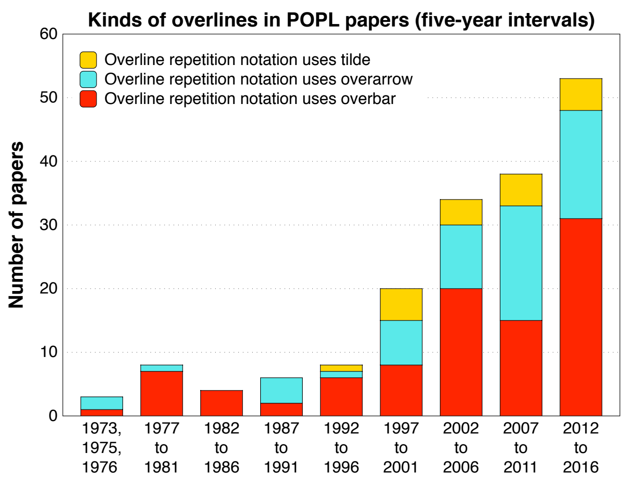

Figure 5. Kinds of overlines in POPL papers, 1973–2016

We observed 9 papers that used nested overline notation; of these, 8 used nested overbars [P94N, P99B, P05A, P13Y, P14E, P14Ψ, P16R, P16Y] and 1 used nested overarrows [P15W].

For comparison, we examined several well-known books. Barendregt [1, §2.1.3] wrote "Let $\vec{N} \equiv N_1, \ldots, N_n$. Then $MN_1 \cdots N_n \equiv M\vec{N} \equiv (\cdots((MN_1)N_2)\cdots N_n)$"; note that he clearly does not intend that $\vec{N}$ necessarily literally include separating commas in its expansion. Milner *et al.* [7, p. 44] used ellipses not only within an assertion but also to indicate a sequence of premisses in an inference rule:

$$\frac{E(\mathit{longstrid}_1) = E_1 \qquad \cdots \qquad E(\mathit{longstrid}_n) = E_n}{E \vdash \texttt{open}\ \mathit{longstrid}_1 \cdots \mathit{longstrid}_n \Rightarrow E_1 + \cdots + E_n}$$

and Gunter [5, p. 292] did the same:

$$\frac{H \vdash M_1 : t_1 \qquad \cdots \qquad H \vdash M_n : t_n}{H \vdash \{l_1 = M_1, \ldots, l_n = M_n\} : \{l_1 : t_1, \ldots, l_n : t_n\}}$$

and so did Reynolds [9, p. 227] (using zero-origin indexing):

$$\frac{e_0 \Rightarrow z_0 \qquad \cdots \qquad e_{n-1} \Rightarrow z_{n-1}}{\langle e_0, \ldots, e_{n-1}\rangle \Rightarrow \langle z_0, \ldots, z_{n-1}\rangle}$$

Bruce [2, p. 164] used both ellipses and iterator notation:

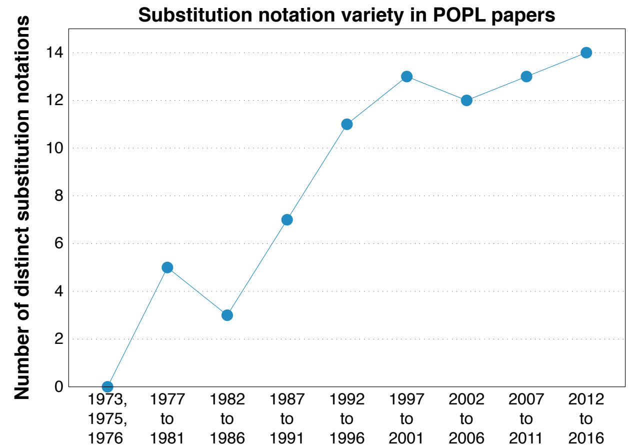

Figure 6. Substitution variety in POPL papers, 1973–2016

$$\frac{\mathcal{E} \vdash M_i : T_i,\ \ \mathit{for}\ 1 \leq i \leq n}{\mathcal{E} \vdash \langle M_1, \ldots, M_n\rangle : T_1 \times \ldots \times T_n}$$

3. Analysis

Based on the data reported in Section 2 plus other observations about the papers, we infer this story about metanotation in POPL papers: Before 1987, there was relatively little use of metanotation for formal specification of the behaviors and type systems of languages. In the late 1980s, there was a shift from studying small languages (such as various forms of the λ-calculus) in which functions took only one argument to explaining larger, more realistic languages in which sequences of items (parameters, arguments, declarations, statements, . . .) played a role, and so there was a sharply increased need for metanotation to describe such sequences. Moreover, the use of inference-rule metanotation became increasingly popular, to the point that over 60% of POPL papers now use inference rules for some descriptive purpose. Different subjects have had slightly different descriptive needs, and so there has been a natural experimentation with alternative extensions to the metanotation, leading to an ever-increasing diversity of notations.

This is how languages grow—but we believe that it has reached the point where it is causing problems, and it is time to apply the tools of our trade (including formalism and critical analysis), which we normally apply to programming languages, to the metanotation.

In this section we point out specific problems we have observed with three aspects of the metanotation: substitution notations, overline notations, and ellipses. We cite specific examples from specific papers for concreteness, to document the existence of these problems, but it is not our intent to single out individual papers as "bad examples"; rather, we believe the problems exist as a natural consequence of language evolution, as cutting-edge pioneers try out different (and experimental) modes of expression, and the nature of the problems observed is a global inconsistency across a large body of work rather than defects in particular papers.

3.1 Substitution Notations

We don't believe that there is an inherent problem with any one of the 27 specific notations for substitution listed in Table 1. The difficulties we see are social rather than technical, and twofold.

The first problem is that authors use a wide variety of substitution notations, but some take them for granted; many, many papers use substitution notation without explaining what it means. For each notation, enough other POPL papers use the same notation (or a very similar notation) for substitution that any individual author might well feel justified in assuming that readers will recognize it. But the diversity of notations has been increasing over time (see Figure 6); during the last five years alone, 14 different notations have been used, making it less likely that a reader will instantly recognize any one of them as intended to denote substitution.

Some general comments about the notations observed: (1) Both $e[v/x]$ and $e[x/v]$ are in use, identical except as to whether v precedes or follows x. Therefore a reader who sees $a[b/c]$ cannot be sure which is meant. (2) Focusing only on separators, 4 papers use $x \to v$ and 12 papers use $x \leftarrow v$. A reader who had already seen one (say, $x \to v$) might well think that the rule is that the arrow points from the variable to the replacement value, and on encountering the other, say in the form $a[b \leftarrow c]$, might well think that c is the variable and b the replacement value, when in fact the opposite was intended. (3) The very fact that most notations that use "/" as a separator (including the two most popular notations, $e[v/x]$ and $[v/x]e$) have v to the left of x, whereas notations that use ":=" or some form of arrow as a separator have v to the right of x, is itself a potential source of confusion. (4) We speculated that the increasing use of braces might reflect a desire to emphasize that a substitution conceptually includes a set, not an ordered list, of variable-value pairs when multiple substitution is involved. However, our observations provide little statistical support for this conjecture: of the 327 papers that use substitution notation, 100 papers (30.5%) use multiple simultaneous substitution, and of the 50 papers out of 327 that uses braces as enclosers, just 16 (32%, almost exactly the same percentage) use multiple simultaneous substitution.

The second problem, which exacerbates the first, is that many of these notations are also used for completely different purposes in other POPL papers. As a result, the reader cannot even be certain, on seeing a notation such as $e[v/x]$ or $e[x := v]$, whether it is intended to denote substitution or some other operation. For example, paper [P86C, §5] uses the notation one $\mathbf{e}[\hat{\mathbf{x}}/\mathbf{x}]$ to mean environment extension, not substitution; the notation is not explicitly explained, but from its use we infer that if $\mathbf{e}$ is an environment then

$$\big(\mathbf{e}[\hat{\mathbf{x}}/\mathbf{x}]\big)\,[\![id]\!] = \begin{cases} \hat{\mathbf{x}} & \text{if } id \text{ is } \mathbf{x} \\ \mathbf{e}[\![id]\!] & \text{otherwise} \end{cases}$$

Another paper [P88E, §4.3.1] gives the explicit definition

$$f[a \to b] \stackrel{\text{def}}{=} \lambda x.\text{if } x = a \text{ then } b \text{ else } f(x)$$

and so $f[a \to b]$ denotes function update, not substitution. In a third paper [P89E], the authors write "To denote the extension of a type environment TE by a binding of $\mathtt{x}$ to $\mathtt{T}$, we write $TE[\mathtt{x} \leftarrow \mathtt{T}]$." A fourth paper [P12Π] uses $\rho\overline{\{x_i \mapsto x_i'\}}$ to indicate "repeated extension of the environment ρ with variable mappings"; thus $e\{x \mapsto v\}$ would indicate a single environment extension, not substitution. These are but a handful of the many papers we observed that use substitution-like notations for other purposes.

3.2 Overline Notations

The earliest uses of overline notation were over single symbols, and the interpretation was straightforward: one may regard $\overline{A}$ as standing for an empty sequence, or a singleton sequence "A_1" or a length-2 sequence "A_1, A_2" or a length-3 sequence "A_1, A_2, A_3" and so on for any finite length of sequence desired. Typically such expansions are described using ellipsis notation: we say that "$\overline{A}$" stands for "$A_1, \ldots, A_n$". Often this notation is used in the context of abstract syntax, where the need for parentheses is often fudged away or glossed over, and the need for commas to separate the copies of the symbol can be similarly fudged away or glossed over. It is completely clear what to replicate and where to attach the indices: the unit of replication is the single symbol under the overline, and an integer index is attached to each copy of that symbol. It is assumed that separate occurrences of the same overlined symbol will expand into sequences of the same length.

Over time the notation was extended in three significant ways, which we will refer to as *pointwise clustering*, *expression replication*, and *nesting*. Each of these extensions solved a problem at the expense of introducing a different problem.

The first extension, pointwise clustering, uses multiple overline repetitions within a specific syntactic context to indicate that the unit of replication should be not just individual symbols, but the entire context. At first this convention was described explicitly, but then came to be taken for granted and extended to situations not previously documented, creating the potential for confusion. We quote a typical explanation of the convention [P97E, Appendix B]:

> We use vector notation $\overline{A}$ to indicate a sequence $A_1, \ldots, A_n$. If $\overline{A} = A_1, \ldots, A_n$ and $\overline{B} = B_1, \ldots, B_n$ and $\oplus$ is a binary operator then $\overline{A} \oplus \overline{B}$ stands for $A_1 \oplus B_1, \ldots, A_n \oplus B_n$. If $\overline{A}$ if $\overline{B}$ have different lengths then $\overline{A} \oplus \overline{B}$ is not defined. Each predicate p is promoted to a predicate over vectors: $p(A_1, \ldots, A_n)$ is interpreted as $p(A_1) \wedge \ldots \wedge p(A_n)$.

So far, so good. However, the same paper goes on to use the same convention for declarations of function parameters: $Ax(\overline{B}\ \overline{y})\{S\}$ is intended to be interpreted as $Ax(B_1\ y_1, \ldots, B_n\ y_n)\{S\}$. Perhaps the whitespace that separates the type B from the parameter name y is to be construed as a "binary operator"?

Many papers have used this convention; by 2006 at least one set of authors regarded this convention as widespread, but still worth explaining: "Following common convention, $\overline{\mathsf{T}}\ \overline{\mathsf{f}}$ represents a list of pairs $\mathsf{T}_1\ \mathsf{f} \cdots \mathsf{T}_n\ \mathsf{f}_n$ rather than a pair of lists" [P06P, §3.1].

Pointwise clustering may induce transitive constraints: if, within some context, we have $\overline{T\ f}$ and elsewhere $\overline{T : \kappa}$, f and κ must have the same length even though they nowhere appear together. (Following Einstein, we call this "spooky action at a distance.")

Pointwise clustering solves the problem of wanting to replicate syntactic units larger than single symbols, with the effect of being able to "zip together multiple lists of the same length" in constructing the copies. The downside is that the unit of replication is not indicated explicitly, and so must be explained separately.

But consider now this example [P04J, Ex. 4.7], one of the clearest illustrations of the difficulty that arises when the contexts to be replicated are not explicitly marked or explained:

> Take any $\overline{w} = [\overline{v}/\overline{x}]\overline{e}$ and $\overline{w}' = [\overline{v}'/\overline{x}]\overline{e}$ with $(\overline{v}, \overline{v}') \in R$

There are five different vectors involved: $\overline{w}$, $\overline{w}'$, $\overline{v}$, $\overline{v}'$, and $\overline{e}$. Which pairs of vectors must be of the same length? All of them? We soon realize that to answer this, we need to understand what are the intended units of syntactic replication. For the first two equations, a plausible answer is "the entire equation," leading to the interpretation

> Take any $w_1 = [v_1/x_1]e_1, \ldots, w_n = [v_n/x_n]e_n$
> and $w_1' = [v_1'/x_1]e_1, \ldots, w_n' = [v_n'/x_n]e_n$ with ...

However, careful consideration of the rest of the paper leads one to conclude that the intended interpretation is:

> Take any $w_1 = [v_1/x_1, \ldots, v_m/x_m]e_1, \ldots,$
> $w_n = [v_1/x_1, \ldots, v_m/x_m]e_n$
> and $w_1' = [v_1'/x_1, \ldots, v_m'/x_m]e_1, \ldots,$
> $w_n' = [v_1'/x_1, \ldots, v_m'/x_m]e_n$ with ...

That last is quite a mouthful, so we can understand why the authors chose to use abbreviate it using a repetition notation. But this particular form of overline notation, which fails to mark the units of replication, is not quite up to the task. The same problem shows up elsewhere in the example: how is $(\overline{v}, \overline{v}') \in R$ to be interpreted?

Here is a clear example of the sort of special-purpose explanation that is required when using pointwise clustering, particularly with substitution notation [P11I, §3.3]:

> The notation $\overline{a} : \overline{\kappa}$ zips together a list of type variables and a list of kinds to create a type variable context Δ. These two lists must have the same length for the notation to be well-defined. The notation $\varphi[\overline{a} \mapsto \overline{\psi}]$ applies a multi-substitution of the types $\overline{\psi}$ for each of the corresponding variables in the list $\overline{a}$.

The second extension, which we call expression replication, solves these problems by writing the overline over an entire expression, not just individual symbols, thereby indicating quite precisely what is the unit of replication. This works very well for multiple substitution notation such as $\{\overline{\alpha \mapsto \phi'}\}(\phi_1)$ [P16Q, Fig. 3], but in the general case it comes at a cost: the precise points at which to attach the integer subscripts are no longer explicitly marked. For the previous example, we could try writing $\overline{(v, v') \in R}$; the unit of replication is clearly "$(v, v') \in R$" but it is not at all clear that v and v' should receive subscripts in each copy but R should not—for that matter, we might even ask whether $\in$ should also receive subscripts. Some authors have solved this problem by explicitly marking the subscript attachment points, typically using either i or j or k, intended to represent a "typical index value" (example: $\overline{(\mathcal{G}_i, \psi_i)}$ [P05U, Fig. 3]) or m or m, intended to do double duty by also indicating the intended length of the sequence (example: $\theta = \{\overline{X_n \mapsto t_n}\}$ [P94N, §3]). In the former case, authors may also provide a binding of the index variable that indicates the attachment points, for example $\overline{K_i \Rightarrow e_i}^{\,i}$ [P11I, Fig. 2] or $\overline{C_i\, x \to e_i}^{\,i \in m}$ [P16Y, Fig. 8]. This makes the attachment points quite clear, but with less conciseness than a pure overline notation.

We did find this clear description of a pure overline notation with implicit subscript attachment points [P06H, §2]:

> We will also use an overbar as a syntactic meta-operator to denote a comma-separated sequence of syntax fragments: $\overline{\sigma} = \sigma_1, \ldots, \sigma_n$ where σ is a syntax fragment and σ_i is the same fragment with i-subscripts on all meta-identifiers it contains. For example we will write $[\overline{u/x}]e$ instead of $[u_1/x_1, \ldots, u_n/x_n]e$, and $\overline{(v, v')} \in R$ instead of $(v_1, v'_1), \ldots, (v_n, v'_n) \in R$.

The third extension, nesting, effectively provides "nested loops." The earliest example we found, $\sigma' = \{F \mapsto \lambda\overline{y_n}.a(\overline{H_m(\overline{y_n})})\}$ [P94N, Fig. 1], explicitly marks the index attachment points but provides no bindings, thereby removing all ambiguity as to where subscripts are to be attached but not entirely eliminating all confusion as to which subscript attachment points correspond to which overline (it can easily be figured out, but it requires a deduction step). The same is true of $\mathtt{data}\ T\ \overline{\alpha_k} = \overline{C_i\, \overline{\tau_{ij}}}$ [P99B, Fig. 1], which uses three distinct index variables i, j, and k to correspond to the three overbars. A more complex example is $\overline{y_i : (\forall\overline{\alpha_k}.\tau_i)^{u_i} \mapsto \Lambda\overline{\alpha_k}.e_i[S]}$ [P99B, Fig. 1]; note the occurrence of the index variable i within the superscripted expression u_i. Such an example would be difficult to understand without explicit index variables or some specific conventions about index attachment.

An interesting example of apparently mixed conventions is $\Omega \vdash \overline{[\overline{\tau'}/\overline{\alpha}]\tau} \rightsquigarrow v$ [P05A, Fig. 4]. Elsewhere the authors generally use expression replication in such situations as $\mathbf{data}\ \overline{D\ \alpha} \Rightarrow S\ \alpha\ \overline{\beta} = C\ \overline{\tau}$ [P05A, Fig. 1] and $\Delta \to \overline{d : \theta}$ [P05A, Fig. 3], so we assume that the authors believed that $[\overline{\tau'}/\overline{\alpha}]\tau$ rather than $[\overline{\tau'/\alpha}]\tau$ was the appropriate notation for multiple substitution.

A different example of mixed conventions within a single assertion is $\beta = \{\!|(\mathcal{G}_0, \mathsf{null}), \overline{(\mathcal{G}_i, o_i.f_1.\cdots.f_n)}|\!\}$ [P05U, Fig. 5]. Here there is nested repetition: a use of an ellipsis within an overline cluster that uses the index variable i to indicate attachment points. Nothing wrong with that; this may be the clearest way to express this situation. It is in fact clearly the intent of the authors that the sequence of field accesses $.f_1.\cdots.f_n$ be the same for every value of i. This example does suggest that in some other scenario one might wish to have a different set of field accessors for each value of i; this could be expressed as $\overline{(\mathcal{G}_i, o_i.f_{i\,1}.\cdots.f_{i\,n_i})}$. Can this example be expressed more concisely using nested overlines? We answer this question in §6. (This example also illustrates a conventional treatment of the overline notation as concrete syntax or "macro expansion" rather than abstract syntax: the intent is that the copies produced by the overline cluster be comma-separated but not surrounded by enclosers, so that β has $n+1$ pairs as its elements.)

3.3 Ellipses

We wish to provide a formal specification of the meaning of such notations as "$x_1, \ldots, x_n$" and "$x_1, x_2, \ldots$" and "$x_1 \oplus \cdots \oplus x_n$" and "$[v_1/x_1, \ldots, v_n/x_n]e$". Sometimes two ellipses are used together, for example when nested function calls are involved: $f_1(f_2(\cdots f_n(x)\cdots))$. Ellipsis notation is sometimes used in tricky ways, for example $C = [p_1, s_1, \ldots, p_n, s_n]$ [P06X, Fig. 6]. In any case, it won't do simply to assume that the repeated pattern is comma-separated and contains no commas.

4. A Specific Proposal, with Three Novelties

We propose these design principles as desirable for future POPL metanotation: (1) Be compatible with past usage. (2) Obey the principles of abstract syntax (that is, a correct concrete syntax can be obtained purely by inserting enclosers and commas, but only as needed to maintain the integrity of the parse tree). (3) Provide a range of notational choices, allowing authors to make the choices about conciseness and readability. (4) Avoid context-dependent behavior. (5) Be as agnostic as possible about the structure and meaning of the assertion language. (6) Be as agnostic as possible about the structure and meaning of the object language. (7) Provide a purely formal explanation of the interpretation of the metanotation (in particular, the interpretation should not depend on types or semantics of either the assertion language or the object language).

In this section we present an informal description of a few aspects of our proposed metanotation, including some novel features. In §5.2 we present a definition and formal interpretation.

> In general, an *inference rule* consists of zero or more *premisses* and a single *conclusion*. . . . The premisses and conclusion are each a scheme for an assertion, that is, a pattern containing metavariables that each range over some type of phrase, such that one obtains an assertion by replacing each metavariable by any phrase in its range. . . . An instance of an inference rule is obtained by replacing all occurrences of each metavariable by a phrase in its range. (Sometimes, there will be side conditions on the rule that must be satisfied by the replacement. Also, there may be syntactic operations, such as substitution, that must be carried out after the replacement.) A *proof*—more precisely, a *formal proof*—is a sequence of assertions, each of which is the conclusion of some instance of an inference rule whose premisses all occur earlier in the sequence. —Reynolds [9, §1.3]

Each of the assertions (sometimes called *judgments*) is written in an *assertion* language, which is typically the standard language of mathematics and logic, possibly augmented with substitution and/or repetition notations, and also with the possibility of mentioning tokens of some *object language* (Reynold's term) and/or nonterminals of a context-free grammar defined in some BNF notation (of which there are many variations); such nonterminals are one kind of metavariable. Examples of possible object languages are Java [4] and Featherweight Java [6].

We will use the term *monogram* to refer to a single letter that, rather than being used for decorative purposes, is itself possibly "decorated" with one or more prime marks and/or a sequence of one or more integer subscripts. We will define this term formally in §5; for now, consider as examples the monograms x, β, e', α_2, and $\tau'_{15\,27}$. The decorations are part of the monogram; thus x and x' and x_3 are distinct monograms. Multicharacter identifiers such as *expr* and *type* and `if` are not monograms; neither are nonalphabetic symbols such as $=$ and $+$ and `&`.

Our basic approach to making overline notation unambiguous: (1) Implicit pointwise clustering is never used. The material beneath (that is, within) an overline is the unit of replication. Thus" $\overline{T}\ \overline{f}$" expands to "$T_1, \ldots, T_n\ f_1, \ldots, f_n$". If "$T_1\ f_1, \ldots, T_n\ f_n$" is the desired expansion, the correct way to write that is "$\overline{T\ f}$".
(2) If explicit index variables are to be used, then the overline notation must include an explicit binding of that variable, so that the correspondence of index variables to overlines will be unambiguous. For example, we cannot write $\overline{v_i/x_i}$; instead we must write at least $\overline{v_i/x_i}^{\,i}$, and perhaps more explicitly $\overline{v_i/x_i}^{\,i\leq n}$ or $\overline{v_i/x_i}^{\,p\leq i\leq q}$. Index variables are the way to go if it is necessary to attach subscripts to symbols or multicharacter identifiers.
(3) If no explicit index variables are to be used, then integer indices will be implicitly attached to *all* monograms (but see §4.1 below), and *only* to monograms. Thus $\overline{f(x)}$ will mean $f_1(x_1), \ldots, f_n(x_n)$, not $f(x_1), \ldots, f(x_n)$, because f, like x, is a monogram.

We allow any combination of three optional notations at the upper right of an overline: an *explicit separator*, a *multiplicity marker* (which may be $+$ or ?), and a *variable binding*. The explicit separator is used instead of a comma to separate copies; if s represents statements, then a notation such as $\overline{s}^{;}$ could be useful when describing an object language that uses semicolons as separators rather than terminators. The multiplicity marker $+$ means that the expanded sequence must not be empty; ? means that the expanded sequence must contain either zero or one copy (and therefore in this case also specifying an explicit separator would be pointless). A variable binding may provide bounds, or may consist of just the variable name, in which case bounds will be inferred. Examples are $\overline{x}^{;+}$, $\overline{x}^{+\,i}$, $\overline{x}^{?}$, $\overline{x}^{\,i}$, $\overline{x}^{\,j<i}$, $\overline{x}^{\boxed{\texttt{max}}}$, $\overline{x}^{\,1\leq i\leq n}$, $\overline{x}^{\otimes+\,i\leq n}$, and $\overline{x}^{\,i\in 1..n}$.

4.1 Underlining

What if we want an expansion such as $f(x_1), \ldots, f(x_n)$? We can always use explicit indices, as in $\overline{f(x_i)}^{\,i}$, but what if we want further conciseness? We take inspiration from the α notation for parallel computation in Connection Machine Lisp [12], which was in turn inspired by the backquote notation of Common Lisp [11]. In each of these notations, an expression is marked for special treatment (making a copy; executing many copies in parallel), but there is also a way to mark subexpressions as *exceptional* (use the value of the subexpression instead of making a copy; use corresponding elements of a vector rather than replicating a single value). The overline notation attaches subscripts to *every* monogram; we need a way to say "except here," and for this we use *underlines*. Just as every comma must correspond to a governing backquote in Common Lisp, just as every bullet must correspond to a governing α, so in our metanotation every underline must fall beneath a corresponding overline. As simple examples, for $f(x_1), \ldots, f(x_n)$ we can write $\overline{\underline{f}(x)}$, and for $f(x_1 + z, \ldots, x_n + z)$ we can write $f(\overline{x + \underline{z}})$.

Consider again this example [P04J, Ex. 4.7] from §3.2:

$$\text{Take any } \overline{w} = [\overline{v}/\overline{x}]\overline{e} \text{ and } \overline{w}' = [\overline{v}'/\overline{x}]\overline{e} \text{ with } (\overline{v}, \overline{v}') \in R$$

With our proposed notation, we use nested overlines to indicate the (nested) units of replication and underlines to indicate where the uppermost overline should *not* attach indices:

$$\text{Take any } \overline{w = [\underline{\overline{v/x}}]e} \text{ and } \overline{w' = [\underline{\overline{v'/x}}]e} \text{ with } \overline{(v, v') \in \underline{R}}$$

Thus, in each of the two equalities, v and v' and x receive indices only from the inner overline. Note also the underline under R to indicate that there is just one R, not a subscripted sequence of R's.

4.2 Harpoons and Boxes

Harpoons used as overlines do not provide separators between the copies. (We use harpoons so as to avoid conflict with past usage of overarrows; besides, harpoons take less vertical space.) The direction of the harpoon indicates whether copies are numbered in forward or reverse order. If a harpoon overline cluster immediately contains a single boxed cluster, then the material in the boxed cluster is used as is. and the material to its left and right is replicated; this provides a concise way to notation certain expressions that would otherwise require two ellipses. We illustrate with examples:

$$
\begin{array}{rcl}
\overrightharpoon{x} & \equiv & x_1 x_2 x_3 \ldots x_{n-1} x_n \\
\overleftharpoon{x} & \equiv & x_n x_{n-1} x_{n-2} \ldots x_2 x_1 \\
\overrightharpoon{\mathsf{let}\ x = v\ \mathsf{in}\ \boxed{e}} & \equiv & \mathsf{let}\ x_1 = v_1\ \mathsf{in} \ldots \mathsf{let}\ x_n = v_n\ \mathsf{in}\ e \\
\overleftharpoon{\mathsf{let}\ x = v\ \mathsf{in}\ \boxed{e}} & \equiv & \mathsf{let}\ x_n = v_n\ \mathsf{in} \ldots \mathsf{let}\ x_1 = v_1\ \mathsf{in}\ e \\
\overleftharpoon{\boxed{e}.f} & \equiv & e.f_1.f_2.f_3 \cdots .f_{n-1}.f_n \\
\overrightharpoon{\boxed{e}.f} & \equiv & e.f_n.f_{n-1}.f_{n-2} \cdots .f_2.f_1 \\
\overrightharpoon{\underline{h}(x, \boxed{e}, z)} & \equiv & h_1(x_1, h_2(x_2, \ldots h_n(x_n, e, z_n) \ldots, z_2), z_1) \\
\overleftharpoon{\underline{h}(x, \boxed{e}, z)} & \equiv & h_n(x_n, \ldots h_2(x_2, h_1(x_1, e, z_1), z_2) \ldots, z_n)
\end{array}
$$

4.3 Explaining Ellipses Rigorously

We provide the overline notation because it is concise, mentioning the repeated material just once. But we also wish to provide ellipsis notation, despite the fact that it typically mentions (variations of) the repeated material two or more times, because it is often more readable. In §5.2 we will explain the formal interpretation of several ellipsis idioms by transforming them into instances of overline notation. Here are some illustrative examples:

$$
\begin{array}{rll@{\qquad}rll}
x_1, x_2, \ldots & \text{becomes} & \overline{x}\boxed{,} & x_1, \ldots, x_n & \text{becomes} & \overline{x_i}\boxed{,}^{\,1\leq i\leq n} \\
x_1 x_2 \ldots & \text{becomes} & \overrightharpoon{x} & x_1; \ldots; x_n & \text{becomes} & \overline{x_i}\boxed{;}^{\,1\leq i\leq n}
\end{array}
$$

$$
\begin{array}{rll}
f_1(x_1, y), \ldots, f_n(x_n, y) & \text{becomes} & \overline{f_i(x_i, y)}\boxed{,}^{\,1\leq i\leq n} \\
x_3 \oplus \cdots \oplus x_7 & \text{becomes} & \overline{x_i}\boxed{\oplus}^{\,3\leq i\leq 7} \\
x_1 \ldots x_n & \text{becomes} & \overrightharpoon{x_i}^{\,1\leq i\leq n} \\
f(x_1, \ldots f(x_n, e) \ldots) & \text{becomes} & \overrightharpoon{\underline{f}(x, \boxed{e})}
\end{array}
$$

5. A Careful Specification

Our description of metanotation builds on that of Reynolds [9, §1].

5.1 Syntax

Assertions, inference rules, and BNF are all built from tokens.

5.1.1 Monograms and Other Tokens

A *letter* is a single letter (Latin, Greek, or perhaps from some other alphabet), for example x, A, Z, β, and Γ. An *accented letter* is either a letter, or an accented letter to which a depending or surmounting or superscripted symbol (other than an overbar or harpoon) has been added; examples are x, $ç$, $é$, $\hat{x}$, A', $\widetilde{Z}'$, $\beta^{\dagger}$, and Γ''. A *monogram* is either an accented letter, or a monogram to which an integer subscript has been attached; examples of monograms are x, $\hat{x}$, x_1, $A_{2\,2}$, and $\psi''_{1\,3\,2}$. Here we use whitespace between subscripts, so that the monogram $\tau_{4\,12}$ (having subscripts 4 and 12) is clearly different from $\tau_{41\,2}$ (which has subscripts 41 and 2). An *indexed monogram* is either a monogram to which a subscript other than an integer has been attached, or an indexed monogram to which a subscript has been attached; examples are x_i, $x_{3\,j}$, and $x'_{k\,3\,k}$.

Monograms may be used as names of metanotation index variables and as BNF nonterminals. They may also be used within the assertion language and the object language for other purposes.

We assume that the syntax of the assertion language (which may include part or all of the syntax of the object language) may be regarded abstractly as a linear sequence of tokens, some of which may have a compound structure that may include tokens or sequences of tokens. We also assume that such tokens may be divided into five classes: *monograms*, *commas*, *left enclosers*, *right enclosers*, and all other tokens, which we will call *common tokens*. The assertion template language has six additional classes of token: *ellipses* such as "$\ldots$" or "$\cdots$", *overline clusters*, *underline clusters*, *boxed clusters*, the *don't-care symbol* "$\_$", and the *empty-sequence symbol* "$\bullet$". An overline cluster consists of a nonempty sequence of assertion template tokens surmounted by an overline, where an overline is either an overbar, a left-pointing harpoon, or a right-pointing harpoon; such an overline cluster may have additional information attached at the upper right, as described below. An underline cluster consists of a nonempty sequence of assertion template tokens with an underbar beneath. A boxed cluster consists of a nonempty sequence of assertion template tokens within a rectangular box. In each case, a cluster is said to *contain* the token sequence, which is sometimes referred to as the *material* within the cluster, and to *immediately contain* each of the tokens in the material. A cluster also contains any token contained by any token in the material; thus (non-immediate) containment is recursive.

Some tokens in the assertion language may belong to the object language. The assertion language comma token is ","; there may also be a separate object language comma token such as "`,`". Possible examples of left enclosers are "$($" and "`(`" and "$[$" and "`[`" and "$\langle$" and "`begin`" and "`if`"; examples of right enclosers are "$)$" and "`)`" and "$]$" and "`]`" and "$\rangle$" and "`end`" and "`fi`".

As an example, the assertion "$\Gamma \vdash f(\overline{y + \underline{z}}) : \tau$" might be regarded as a sequence of eight tokens: "Γ" and "f" and "τ" are monograms, "$\vdash$" and ":" are common tokens, "$($" is a left encloser, "$)$" is a right encloser, and "$\overline{y + \underline{z}}$" is an overline cluster that happens to contain a sequence of three other tokens, namely the monogram "y", the common token "$+$", and an underline cluster "$\underline{z}$"that contains the monogram "z".

We say that a sequence of tokens is an *entire* sequence if it constitutes the whole of an assertion or a BNF alternative or a formula mentioned within text, or if it constitutes all of the material within an overline cluster or underline cluster.

We say that a token is *left-delimited* if either (a) it is the leftmost of an entire sequence of tokens, or (b) the token immediately to its left is either a comma or a left encloser. Similarly, we say that a token is *right-delimited* if either (a) it is the rightmost of an entire sequence of tokens, or (b) the token immediately to its right is either a comma or a right encloser.

We say that a sequence of tokens is *left-balanced* if (a) the material in every overline cluster and every underline cluster immediately contained in the sequence is balanced, and (b) there is no prefix of the sequence (including the sequence itself) that immediately contains more right enclosers than left enclosers. Similarly, a sequence of tokens is *right-balanced* if (a) the material in every overline cluster and every underline cluster immediately contained in the sequence is balanced, and (b) there is no suffix of the sequence (including the sequence itself) that immediately contains more left enclosers than right enclosers. A sequence of tokens is *balanced* if it is both left-balanced and right-balanced, *left-enclosing* if it is left-balanced but not right-balanced, *right-enclosing* if it is right-balanced but not left-balanced. A token z is *unenclosed* if either the maximal subsequence to the left of z in the immediately containing token sequence is not left-enclosing or the maximal subsequence to the right of z in that token sequence is not right-enclosing.

5.1.2 Assertions and Inference Rules

A *assertion* is a sentence of the assertion language that may be determined to be valid or invalid. We rely critically on only one characteristic of assertion language syntax: that it have a comma token. The assertion language may use a notation such as $e[v/x]$ or $e[v_1/x_1, \ldots, v_n/x_n]$ to indicate substitution, where e and v and every v_i represent phrases of the object language, x and every x_i represent single-token identifiers of the object language, and "$[$"and "$]$"and "$/$" are tokens of the assertion language but not of the object language. The assertion language may include a function $\#$ that can take any number of arguments and returns a nonnegative integer indicating now many arguments it was given; for example, $\#(a, z_i, x > y) = 3$.[1]

An *inference rule* consists of a set of assertions called the *premisses* and a second, nonempty set of assertions called the *conclusions*[2]; it is customary to notate an inference rule as a horizontal line with the premisses above the line and the conclusion(s) below the line. The premisses (and conclusions) may be stacked vertically and/or separated by commas [P78A, §1.5ff.], but more recent custom is to put multiple premisses on a line, separated only by wide whitespace (2 ems or more), while trying to minimize the number of lines required. If an inference rule has no premisses, one may either (1) leave whitespace above the horizontal line, (2) write "$\bullet$" above the horizontal line, or (3) omit the horizontal line.

A *possibly repeated inference rule* is either an inference rule or an overline cluster whose overline is an overbar and whose material is a possibly repeated inference rule (rather than a set of tokens).

5.1.3 BNF

A BNF (Backus-Naur Form) description of an object language consists of one or more *productions*. Each production has a *nonterminal* on its left-hand side[3] and a set of *alternatives* on its right-hand side. Each nonterminal is typically an identifier, and may be a monogram. Each alternative is a *BNF token sequence* annotated by a set of *constraints*. Each token in a BNF token sequence must be either an object-language token, a nonterminal appearing in the left-hand side of some BNF production, a monogram that *matches* a nonterminal appearing in the left-hand side of some BNF production, an ellipsis, or an overline cluster, underline cluster, or boxed cluster whose material is a BNF token sequence. (Thus, all repetition notations may eb used in BNF alternatives.) Each constraint is an assertion. Typically each production is written as the symbol $::=$ with the nonterminals written to its left, separated by commas, and the alternatives written to its right, separated by vertical bars "$|$".

A monogram is said to *match* a nonterminal if the nonterminal is also a monogram and a sequence of *decorations* of the nonterminal can produce the original monogram, where a decoration operation consists of either adding an accent or attaching an integer subscript. (For example, x' matches x, $\hat{x}_3$ matches x, and $\hat{x}'_{3\,8}$ matches any of $\hat{x}'_3$, $\hat{x}'$, $\hat{x}_{3\,8}$, $\hat{x}_3$, $\hat{x}$, $x'_{3\,8}$, x'_3, x', $x_{3\,8}$, x_3, and x.) If a monogram matches more than one nonterminal, the *best match* is the one

[1] We dislike the use of $|\overline{x}|$ to denote the length of the sequence $\overline{x}$ because if we happen to choose $\overline{x} = x_1$, then it is not clear whether $|\overline{x}| = 1$ (using abstract syntax) or $|\overline{x}| = |x_1| = (\mathsf{if}\ x_1 \geq 0\ \mathsf{then}\ x_i\ \mathsf{else}\ -x_i)$ (using concrete syntax). A similar convention with big operators is less dangerous, because when $\#(\overline{x}) = 1$, $\bigwedge \overline{x} = x_1$ under either interpretation.

[2] In most cases an inference rule will have only a single conclusion, but in some cases it is useful to allow several, particularly when they may be generated by a repetition notation. It is as if there were multiple inference rules, each with one conclusion and each having all the same premisses.

[3] As an abbreviation, one may write a comma-separated sequence of nonterminals on the left-hand side of a BNF production; it is as if several copies of the production were written, one for each of the nonterminals listed, with just that nonterminal on its left-hand side.

(if any) that matches all the others. For example, if τ and τ' are nonterminals, then τ' is the best match for the monogram τ'_3, but τ is the best match for τ_3.

5.2 Interpretation

Metanotation is interpreted by expanding a template into a specific instance, which may have constraints attached; the instance is relevant only if the constraints are satisfied. Interpretation proceeds in three steps: macro-expanding repetition notations, replacing BNF nonterminals, and performing substitutions. Each of the first two steps may involve making free choices. In effect, a template is regarded as representing all possible expansions.

5.2.1 Expand Repetition Tokens

Repetition expansion can be performed within any of the following contexts: an inference rule template, one alternative of a BNF production, or a sentence or paragraph of text. Repetition expansion proceeds in eight steps:

Initialize bookkeeping. Create two data structures, each initially empty: C (Constraints) is a set of equalities, and P (Pairs) is a set of pairs (m, v) where m is a monogram and v is a variable.

Transform ellipses to overbars. In this section we use greek-letter monograms to denote sequences of tokens and p and q to represent integer-valued expressions of the assertion language. We use "..." to denote an ellipsis, though it might actually have another appearance such as "$\cdots$" or "$\vdots$".

Repeat the following sentence until execution of the sentence results in no changes to the context: For every ellipsis in the context (visiting them as if in some sequential order), attempt a left-and-right single-ellipsis replacement (see below); if it does not succeed, attempt a left-only single-ellipsis replacement (see below). if it does not succeed, attempt a double-ellipsis replacement (see below).[4]

After the preceding repetition has completed, then every remaining ellipsis that is both left-delimited and right-delimited is replaced with "$\overline{\_}$" (an overline cluster containing the don't-care symbol). It is an error if this process does not eliminate all remaining ellipses in the context.

To attempt a left-and-right single-ellipsis replacement with respect to a given ellipsis: Let x be a fresh variable, and examine the tokens to the left and right of the ellipsis to identify material having the pattern $\alpha'\kappa \ldots \kappa\alpha''$ such that (a) the "..." in the pattern corresponds to the given ellipsis; (b) each of $\alpha'\kappa$ and $\kappa\alpha''$ is maximal (as long as possible), is balanced, and contains no ellipsis; (c) all the material lies within the span of any overline or underline that is above or below the given ellipsis; and (d) there exist α and p and q such that x occurs at least once in α, and $p \neq q$, and the result of replacing every occurrence of x with p in α is α', and the result of replacing every occurrence of x with q in α is α'', and p and q are minimal balanced substrings (or subexpressions) of the assertion language. (Note that κ may be empty.) If such material is identified, replace it with $\overrightarrow{\alpha}^{\,p\leq x\leq q}$ if κ is empty or with $\overline{\alpha}^{\boxed{\kappa}\,p\leq x\leq q}$ if κ is not empty, and the attempt succeeds.

To attempt a left-only single-ellipsis replacement with respect to a given ellipsis: Examine the tokens to the left of the ellipsis to identify material having the pattern $\alpha'\kappa\alpha''\kappa\ldots$ such that (a) the "..." in the pattern corresponds to the given ellipsis; (b) each of $\alpha'\kappa\alpha''\kappa$ is maximal, is balanced, and contains no ellipsis; (c) all the material lies within the span of any overline or underline that is above or below the given ellipsis; and (d) there exists α such that x occurs at least once in α, and the result of replacing every occurrence of x with 1 in α is α', and the result of replacing every occurrence of x with 2 in α is α''. (Note that κ may be empty.) If such material is identified, replace it with $\overrightarrow{\alpha}$ if κ is empty or with $\overline{\alpha}^{\boxed{\kappa}}$ if κ is not empty, and the attempt succeeds.

To attempt a double-ellipsis replacement with respect to a given ellipsis: Let x and y be fresh variables, and examine the tokens to the left and right of the ellipsis to identify material having the pattern $\alpha' \ldots \alpha''\omega\gamma'' \ldots \gamma'$ (note that γ'' precedes γ') such that (a) the material is balanced, the first "..." in the pattern corresponds to the given ellipsis and the second "..." in the pattern corresponds to another ellipsis of the same kind; (b) each of α' and α'' is maximal, is left-enclosing, and contains no ellipsis, and each of γ' and γ'' is maximal, is right-enclosing, and contains no ellipsis, and ω is balanced and contains no ellipsis; (c) all the material lies within the span of any overline or underline that is above or below the given ellipsis; and (d) there exist α and γ and p and q such that x occurs at least once in either α or γ, and $p \neq q$, and the result of replacing every occurrence of x with p in α is α', and the result of replacing every occurrence of x with q in α is α'', and the result of replacing every occurrence of x with p in γ is γ', and the result of replacing every occurrence of x with q in γ is γ'', and p and q are minimal balanced substrings (or subexpressions) of the assertion language. If such material is identified, replace it with $\overrightarrow{\alpha\boxed{\omega}\gamma}^{\,p\leq x\leq q}$, and the attempt succeeds.

Make implicit indices explicit. Repeat the following sentence until the context contains no plain overline (an overbar or overharpoon with no explicit variable binding): For some plain overline cluster in the context that has no plain overline above it, do five things: (1) choose a fresh variable i; (2) for every monogram m that is under that plain overline and has no underline below it, attach i to m as an additional subscript; (3) within the span of the plain overline, identify every underline cluster that has no other underline below it; (4) replace each underline cluster identified in step (3) with the material it contains; (5) add the variable binding i to the plain overline (thereby making it no longer plain).

It is an error if the context now contains any underlines.

Normalize separators. For every overline cluster in the context, if it has an explicit separator that is a common token, replace it with a boxed cluster containing that common token.

For every overline cluster in the context, if the overline is an overbar and the overline cluster has no explicit separator, then add an explicit separator that is a boxed cluster containing a single comma token. If there is an object-language comma that is different from the assertion-language comma token, and either (1) there is an object-language comma or left encloser immediately to the left of the overline cluster, or (2) there is an object-language comma or right encloser immediately to the right of the overline cluster, or (3) the overline cluster appears in a BNF alternative, then the single comma token shall be an object-language comma, and otherwise an assertion-language comma.[5]

At this point, every overline cluster for which the overline is an overbar has an explicit separator that is a boxed cluster.

Normalize variable bindings. For every overline in the context, if its variable binding is of the form $i < n$, replace it with $1 \leq i < n$; if of the form $i \leq n$, replace it with $1 \leq i \leq n$; if of the form $n > i$, replace it with $n > i \geq 1$; if of the form $n \geq i$, replace it with $n \geq i \geq 1$; if of the form $i \in p..q$, replace it with $p \leq i \leq q$.

At this point, every overline cluster has a variable binding that has one of the nine forms i, $p \leq i \leq q$, $p \leq i < q$, $p < i \leq q$,

[4] Other patterns of ellipsis usage, such as "$x_1, \ldots$" and "$x_1, x_2, \ldots, x_n$" and "$x_1, x_2, \ldots, x_{n-1}, x_n$" may easily be interpreted in a similar manner. We omit the handling of such additional patterns for lack of space, but may include them in the final paper.

[5] This rule is context-sensitive. We explored context-free rules for deciding which comma to use and found that none covered all cases of interest.

$p < i < q, p \geq i \geq q, p \geq i > q, p > i \geq q, p > i > q$. Bindings of the first five forms are called *upward* bindings; bindings of the last four forms are called *downward* bindings. In all nine cases we say that i is the bound variable.

Create repetitions. Let $\boxed{e}$ denote (the decimal representation of) the integer value of the expression e. Repeat the following sentence until the context contains no overline: For some overline cluster in the context that has an overbar for its overline and has no overline above it, do five things: (1) freely choose an integer b; (2) if the overline cluster has no multiplicity marker, freely choose a nonnegative integer ℓ, or if the overline cluster has a multiplicity marker $+$, freely choose a positive integer ℓ, or if the overline cluster has a multiplicity marker ?, freely choose ℓ to be either 0 or 1;[6] (3) if the variable binding of the overline is of the form i, then do three things: (a) choose a fresh variable n; (b) for every occurrence of i within the material below the overline, if it occurs as a subscript within an indexed monogram m and the result of removing that occurrence and all following subscripts from m results in a monogram m', then add[7] the pair (m, n) to P; (c) replace the variable binding i with $1 \leq i \leq n$; (4) replace the overline cluster with ℓ consecutive copies of the material within the overline, where within the first copy (if any) i is everywhere replaced by $\boxed{b}$, within the second copy (if any) i is everywhere replaced by $\boxed{b+1}$ (if the binding is an upward binding) or $\boxed{b-1}$ (if the binding is a downward binding), and in general within the jth copy (if any) i is everywhere replaced by $\boxed{b-1+j}$ (if the binding is an upward binding) or $\boxed{b+1-j}$ (if the binding is a downward binding), so that within the last copy (if any) i is everywhere replaced by $\boxed{b-1+\ell}$ (if the binding is an upward binding) or $\boxed{b+1-\ell}$ (if the binding is a downward binding), and where if $\ell > 1$ then adjacent copies of the material within the overline are separated by copies of the material within the boxed cluster that is the explicit separator of the overline cluster; (5) add to C two constraints, depending on the form of the variable binding, according to the following table:

binding form	the two constraints to be added to C	
$p < i < q$	p equals $\boxed{b-1}$	q equals $\boxed{b+\ell}$
$p < i \leq q$	p equals $\boxed{b-1}$	q equals $\boxed{b+\ell-1}$
$p \leq i < q$	p equals $\boxed{b}$	q equals $\boxed{b+\ell}$
$p \leq i \leq q$	p equals $\boxed{b}$	q equals $\boxed{b+\ell-1}$
$p > i > q$	p equals $\boxed{b+1}$	q equals $\boxed{b-\ell}$
$p > i \geq q$	p equals $\boxed{b+1}$	q equals $\boxed{b-\ell+1}$
$p \geq i > q$	p equals $\boxed{b}$	q equals $\boxed{b-\ell}$
$p \geq i \geq q$	p equals $\boxed{b}$	q equals $\boxed{b-\ell+1}$

Overline clusters are handled in a similar manner (we omit the details for lack of space, but may include them in the final paper).

Create length constraints. For all (m, n) in P and all (m', n') in P, if m and m' are the same monogram and the associated variables n and n' are not the same variable, then add to C a constraint[8] that the value of n must equal the value of n'. (Once this is done, P is no longer needed.)

Add constraints to context. The set of equality constraints C is implicitly added to the expansion of the entire context. If the context is an inference rule template, then the constraints may be considered to be additional premisses; because an inference rule has no effect if any of its premisses is not valid, it is as if expansions for which the constraints are not satisfied are never produced at all. If the context is an alternative of a BNF production, then when expanding a nonterminal, an expansion of that alternative may be chosen only if the constraints are satisfied; it is as if expansions for which the constraints are not satisfied are not present at all. If the context is a declarative sentence or paragraph of text, then the claims of that sentence or paragraph should be regarded as true only for expansions for which the constraints are satisfied; it is as if expansions for which the constraints are not satisfied do not occur in the text.

5.2.2 Expand BNF Nonterminals and Other Symbols

If the context is not a BNF alternative, then consider the set of all distinct tokens in the context such that each such token is either a BNF nonterminal or a monogram (possibly both). For each one: (1) if it is a BNF nonterminal, then freely choose some phrase of the object language generated by that nonterminal; (2) if it is not a BNF nonterminal but is a monogram that matches at least one BNF nonterminal, then freely choose some phrase of the object language generated by the nonterminal that is the best match for the monogram (it is an error if there is no best match); (3) if it is a monogram that does not match any BNF nonterminal, then choose (a copy of) the monogram itself as its "freely chosen phrase."

A phrase of the object language is generated by a BNF nonterminal (call it α), if it can be produced by an instance of the following process that terminates after a finite number of steps: (1) first freely choose a production that has the nonterminal on its left-hand side and has at least one alternative whose constraints are satisfied; (2) then freely choose some alternative from that production whose constraints are satisfied; (3) then replace each nonterminal in the token sequence of the alternative with a freely chosen phrase generated by that nonterminal (thus this is a recursive process)—the result is a phrase freely chosen for α.

5.2.3 Perform Substitutions

For every occurrence in the context of a substitution notation (say $e[v_1/x_1, \ldots, v_n/v_n]$), replace each of e and each v_i and each x_i with the result of replacing each BNF nonterminal or monogram within it by the phrase freely chosen for it in the previous step. Then perform the indicated capture-avoiding substitution using the corresponding results, and finally use the result to replace that occurrence of the substitution notation within the context.

For every occurrence of the don't-care symbol "_" in the context, replace it with a freely chosen balanced sequence of object-language tokens that contains no unenclosed comma tokens.

Delete the token "•" everywhere it occurs within the context.

The result is a completely expanded instance of the context.

6. Examples

We believe that our metanotation gives the intended interpretation to the following example [P07D, Fig. 3], which uses ellipses both between assertions and within an assertion:

$$\frac{\delta = \mathsf{C}_1 \text{ of } \tau_1 \mid \cdots \mid \mathsf{C}_n \text{ of } \tau_n \qquad \Gamma \vdash v : \delta \qquad \Gamma, x_1 : \tau_1 \vdash e_1 : \tau \qquad \cdots \qquad \Gamma, x_n : \tau_n \vdash e_n : \tau}{\Gamma \vdash \mathsf{match}\ v\ \mathsf{with}\ (\mathsf{C}_1\, x_1 \to e_1 \mid \cdots \mid \mathsf{C}_n\, x_n \to e_n) : \tau}$$

but also allows further abbreviation using overline notation (and we choose to rename τ to τ' for clarity):

$$\frac{\Gamma \vdash v : \delta \qquad \delta = \overline{\mathsf{C} \text{ of } \tau}^{\,|} \qquad \overline{\underline{\Gamma}, x : \tau \vdash e : \underline{\tau'}}}{\Gamma \vdash \mathsf{match}\ v\ \mathsf{with}\ (\overline{\mathsf{C}\, x \to e}) : \tau'}$$

---

[6] The number of copies ℓ is freely chosen, and then constraints are added to ensure that this choice ℓ satisfies any stated bounds. In this way, the process of expanding repetitions does not rely on the semantics of the assertion-language expressions that denote the bounds; instead, interpretation of such expressions is deferred until after the expansion process is complete.

[7] Adding (m, n) to P may eventually result in creating contraints equating n to other variables, each of which also represents the number of copies produced by expanding an overline cluster. These constraints implement the "spooky action at a distance" aspect of the overline notation.

[8] Spooky!

and if we don't like the look of that explicit separator "|" then we can write it this way, using a left-only ellipsis:

$$\frac{\Gamma \vdash v : \delta \qquad \delta = \mathsf{C}_1 \text{ of } \tau_1 \mid \mathsf{C}_2 \text{ of } \tau_2 \mid \cdots \qquad \overline{\underline{\Gamma}, x : \tau \vdash e : \underline{\tau'}}}{\Gamma \vdash \mathsf{match}\ v\ \mathsf{with}\ (\overline{\mathsf{C}\,x \to e}) : \tau'}$$

One paper [P09E, §4.1] explains "We use the notation $\overline{a}$ for both the list $a_1, \ldots, a_n$ and the set $\{a_1, \ldots, a_n\}$, for $n \geq 0$. We abbreviate terms with list subterms in the obvious way, e.g., $\overline{T}\ \overline{x}$ stands for $T_1\ \ x_1, \ldots, T_n\ \ x_n$, $T\backslash\overline{M}$ stands for $T\backslash M_1\backslash \ldots \backslash M_n$, and $p.\overline{S}$ stands for $p.S_1, \ldots, p.S_n$." But it is not obvious on syntactic grounds alone why $T\backslash\overline{M}$ should not stand for $T\backslash M_1, \ldots, T\backslash M_n$, or $p.\overline{S}$ for $p.S_1. \cdots .S_n$. The metanotation we propose allows one to write $\overleftharpoon{\boxed{T}\backslash M}$ for $T\backslash M_1\backslash \ldots \backslash M_n$ and $\overline{\underline{p}.S}$ for $p.S_1, \ldots, p.S_n$.

Another paper [P16L, §3.3] states "We write [for $\vec{z} < \vec{e} \to E$] short for [for $z_1 < e_1 \to \ldots$ [for $z_n < e_n \to E] \ldots$], and $E[\vec{z}]$ for $E[z_1]...[z_n]$." There are fine *ad hoc* notations when well explained (as here), but such patterns are also easily captured by our harpoon overline notation as $\overrightharpoon{[\mathsf{for}\ z < e \to \boxed{E}]}$ and $\overleftharpoon{\boxed{E}[z]}$ (or perhaps $E\overrightharpoon{[z]}$), and moreover the syntactic process we present in §5.2 provides appropriate interpretations for the ellipsis forms [for $z_1 < e_1 \to \ldots$ [for $z_n < e_n \to E] \ldots$] and $E[z_1]...[z_n]$.

A good example of double ellipses: "We write $\delta(q, \overline{\mathcal{D}(t_i, p_i)})$ for $\delta(\ldots(\delta(q, \mathcal{D}(t_1, p_1)), \ldots), \mathcal{D}(_t n, p_n))$." [P02H, §6.1] (Note the use of the index variable i with the overline, but with no explicit indication that i is bound.) Using harpoons, we write $\overleftharpoon{\delta(\boxed{q}, \mathcal{D}(t_i, p_i))}$ for $\delta(\ldots(\delta(q, \mathcal{D}(t_1, p_1)), \ldots), \mathcal{D}(_t n, p_n))$.

To answer a question we pose near the end of §3.2: using nested overline notation, we write $\beta = \{\!|(\mathcal{G}_0, \mathsf{null}), \overline{(\mathcal{G}_i, o_i.f_1. \cdots .f_n)}|\!\}$ [P05U, Fig. 5] as $\beta = \{\!|(\mathcal{G}_0, \mathsf{null}), \overline{(\mathcal{G}, \overleftharpoon{\boxed{o}.\underline{f}})}|\!\}$, and we write our alternative example $\overline{(\mathcal{G}_i, o_i.f_{i\,1}. \cdots .f_{i\,n_i})}$ as $\overline{(\mathcal{G}, \overleftharpoon{\boxed{o}.f})}$.

Here is a complex example of nested repetition notation [P14Ψ, Fig. 4]:

$$\frac{\begin{array}{c} C{:}\Psi \in \Sigma \qquad \Psi = \overline{[\overline{\alpha{:}\kappa}].\ F(\overline{\rho}) \sim \upsilon} \\ \overline{\Sigma; \Delta \vdash_{\mathsf{ty}} \tau : \kappa_i} \qquad \vdash_{\mathsf{ctx}} \Sigma; \Delta \\ \forall j < i, \mathsf{no\_conflict}(\Psi, i, \overline{\tau}, j) \end{array}}{\Sigma; \Delta \vdash_{\mathsf{co}} C[i]\,\overline{\tau} : F(\overline{\rho_i[\overline{\tau/\alpha_i}]}) \sim \upsilon_i[\overline{\tau/\alpha_i}]}\ \textsc{Co\_Axiom}$$

As the authors explain in the text [P14Ψ, §4.3], the free variable i indicates which of the equations in Ψ (the list of equations of axiom C) is to be instantiated. Elsewhere in the text [P14Ψ, §4.1] we see:

> Although our notation for lists does not make it apparent, we restrict the form of the equations to require that F refers to only one type family—that is, there are no independent F_i. We use subscripts on metavariables to denote which equation they refer to, and we refer to the types $\overline{\rho_i}$ as the type patterns of the ith equation. We assume that the variables $\overline{\alpha}$ bound in each equation are distinct from the variables bound in other equations.

A little thought then shows that we must regard α, κ, and ρ as two-dimensional (that is, doubly indexed) aggregates, but F is not to be regarded as a vector or sequence, despite the fact that in one place it occurs beneath an overline, and τ must be regarded as one-dimensional, despite the fact that in one place it occurs beneath two overlines. The inference rule is perfectly consistent, but cannot be interpreted on a purely syntactic basis; some understanding of the dimensions or types of the metavariables is required. We believe that the following version, expressed in the metanotation presented in this paper, accurately conveys the intention of that paper's authors, but can be interpreted (that is, correctly expanded) using the purely formal, syntactic process we present in §5.2:

$$\overline{\frac{\begin{array}{c} C{:}\Psi \in \Sigma \qquad \Psi = \langle\overline{[\overline{\alpha{:}\kappa}].\ \underline{F}(\overline{\rho}) \sim \upsilon}\rangle \\ \overline{\underline{\Sigma}; \underline{\Delta} \vdash_{\mathsf{ty}} \tau : \kappa_i} \qquad \vdash_{\mathsf{ctx}} \Sigma; \Delta \\ \overline{\mathsf{no\_conflict}(\underline{\Psi}, \underline{i}, \langle\underline{\overline{\tau}}\rangle, j)}^{\,j<i} \end{array}}{\Sigma; \Delta \vdash_{\mathsf{co}} C[i]\langle\overline{\tau}\rangle : F\big(\overline{\rho_i[\underline{\overline{\tau/\alpha_i}}]}\big) \sim \upsilon_i[\overline{\tau/\alpha_i}]}}^{\,i}$$

$\textsc{Co\_Axiom}_i$

We have added enclosers $\langle\,\rangle$ where otherwise a concrete-syntax interpretation might differ from an abstract-syntax interpretation. Underlines make explicit where an overline should not attach subscripts (for example, F and Σ and Δ). An overline cluster replaces the notation $\forall j < i$. Finally, an overline over the entire inference rule provides a binding point for the variable i, making clear that this is really a set of inference rules and that they differ according to the value of i (therefore we added a subscript i to the rule label). Because i is used to index α and κ and ρ and υ, they are all constrained to have the same length n, and the number of inference rules is also constrained to be n, exactly as desired.

7. Recommendations and Future Work

We make no recommendation to prefer one form of metanotation over another, although we do hope that the one we have outlined will prove attractive. We do, however, strongly recommend to future authors that, whatever notations you use for repetition and especially substitution, do not take them for granted, but explain them carefully to readers. By way of example, here are sentences that we plan to use in future papers; they happen to illustrate our notational preferences, but you can easily use them as models for describing any notation you prefer:

> We use $e[v/x]$ to denote the standard capture-avoiding substitution of (a copy of) v for every free occurrence of x within e.
>
> We use $e[\overline{v/x}]$ to denote the standard capture-avoiding simultaneous substitution of (copies of) v_i for every free occurrence of x_i within e, for all $1 \leq i \leq \#(\overline{x})$.
>
> We use $\sigma[x \to v]$ to denote a substitution σ' that is identical to σ except that σ' substitutes v for free occurrences of x.
>
> We use $f[x \mapsto v]$ to denote a function f' that is identical to function f except that $f'(x) = v$.
>
> We use $M[x := v]$ to denote a memory M' that is identical to memory M except that $M'[x] = v$.
>
> We use $\overline{x}$ to mean the sequence $x_1, \ldots, x_n$; more generally, we use $\overline{\textit{token-sequence}}$ to denote a (possibly empty) comma-separated sequence of copies of the *token-sequence*, where within the ith copy, i is attached as a subscript to every monogram that is not underlined. (A monogram is any single letter, possibly with accents and/or subscripts.)

As for future work: It remains to survey and systematically analyze the use of metanotation in other conference series; we believe the most illuminating would be OOPSLA, ICFP, and PLDI. It would be also interesting to trace the development of changes in metanotation notation by examining the connections among papers, including common co-authors, institutional influence, and bibliographic citation; such considerations were beyond the scope of this paper. There may be other useful ways to extend the metanotation presented here, including the accommodation of other patterns for using ellipses. In this paper we primarily used English as the metametanotation, but we should explore using the metanotation itself to define its own syntax and semantics, in the style of one of Reynolds' metacircular definitional interpreters [8]. Finally, there are great opportunities for mechanizing the metanotation, one possible goal being to produce appropriate input for theorem provers such as Coq; one line of research [10, P12Ξ, P15V] has already taken steps in this direction.

Note to reviewers: As required by the 2017 POPL call for papers, our paper is no more than 12 pages in length, excluding the bibliography, in (at least) 9pt format. We have done our best to format the bibliography so as to minimize the total number of pages while maximizing its usefulness.

We are aware that many of the bibliographic entries are in bad shape. That's because we took the BIBTEX entries for all the POPL papers straight off the ACM Digital Library website. We are working to clean them up to acceptable standards by comparing them to the original paper proceedings, and expect to complete the process well before the author response period. (We regard this as the normal process that any author must go through to clean up third-party BIBTEX data; it's just that this time we need to clean up over 600 of them.)

References

[1] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, revised edition. Elsevier Science Publishers, Amsterdam, The Netherlands, 1984. ISBN 0-444-87508-5.

[2] Kim B. Bruce. *Foundations of Object-Oriented Languages: Types and Semantics*. MIT Press, Cambridge, Massachusetts, 2002. ISBN 0-262-02523-X.

[3] Alonzo Church. *The Calculi of Lambda Conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, New Jersey, 1941. Reprinted by Klaus Reprint Corp., New York, 1965.

[4] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley Longman Publishing Co., Inc., Reading, Massachusetts, 1996. ISBN 0-201-63451-1.

[5] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, Cambridge, Massachusetts, 1992. ISBN 0-262-07143-6.

[6] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: A minimal core calculus for Java and GJ. *ACM Trans. Programming Languages and Systems* (*TOPLAS*), 23(3):396–450, May 2001. Association for Computing Machinery, New York. http://doi.acm.org/10.1145/503502.503505

[7] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, Cambridge, Massachusetts, 1997. ISBN 0-262-63181-4.

[8] John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of the ACM Annual Conference: Volume 2*, Boston, Massachusetts, USA, ACM '72, 717–740, New York, 1972. Association for Computing Machinery. http://doi.acm.org/10.1145/800194.805852

[9] John C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, Cambridge, United Kingdom, 1998. ISBN 978-0-521-59414-1.

[10] Grigore Roşu and Traian Florin Şerbănută. An overview of the K semantic framework. *The Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010. http://www.sciencedirect.com/science/article/pii/S1567832610000160

[11] Guy L. Steele Jr., Scott E. Fahlman, Richard P. Gabriel, David A. Moon, Daniel L. Weinreb, Daniel G. Bobrow, Linda G. DeMichiel, Sonya E. Keene, Gregor Kiczales, Crispin Perdue, Kent M. Pitman, Richard C. Waters, and Jon L White. *Common Lisp: The Language (Second Edition)*. Digital Press, Bedford, MA, 1990. ISBN 1-55558-041-6.

[12] Guy L. Steele Jr. and W. Daniel Hillis. Connection Machine Lisp: Fine-grained parallel symbolic processing. In *LFP '86: Proc. 1986 ACM Conference on LISP and Functional Programming*, Cambridge, Massachusetts, 279–297. Association for Computing Machinery SIGPLAN and SIGACT and SIGART, August 1986. ISBN 0-89791-200-4. http://doi.acm.org/10.1145/319838.319870

[13] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, Cambridge, Massachusetts, 1993. ISBN 0-262-73103-7-6.

[P75] **POPL '75**: *Proc. 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1975. Association for Computing Machinery. http://dl.acm.org/citation.cfm?id=512976

[P75A] Neil D. Jones and Steven S. Muchnick. Even simple programs are hard to analyze. In *POPL '75* [P75], 106–118. http://doi.acm.org/10.1145/512976.512988

[P75B] Marvin Solomon. Modes, values and expressions. In *POPL '75* [P75], 149–159. http://doi.acm.org/10.1145/512976.512992

[P76] **POPL '76**: *Proc. 3rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1976. Association for Computing Machinery.

[P76A] Neil D. Jones and Steven S. Muchnick. Binding time optimization in programming languages: Some thoughts toward the design of an ideal language. In *POPL '76* [P76], 77–94. http://doi.acm.org/10.1145/800168.811542

[P77] **POPL '77**: *Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1977. Association for Computing Machinery.

[P77A] Edmund Melson Clarke, Jr. Programming language constructs for which it is impossible to obtain good hoare-like axiom systems. In *POPL '77* [P77], 10–20. http://doi.acm.org/10.1145/512950.512952

[P77B] Bernard Lang. Threshold evaluation and the semantics of call by value, assignment and generic procedures. In *POPL '77* [P77], 227–237. http://doi.acm.org/10.1145/512950.512972

[P78] **POPL '78**: *Proc. 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, USA, 1978. Association for Computing Machinery.

[P78A] Steven M. German. Automating proofs of the absence of common runtime errors. In *POPL '78* [P78], 105–118. http://doi.acm.org/10.1145/512760.512772

[P78B] Robert Cartwright and Derek Oppen. Unrestricted procedure calls in hoare's logic. In *POPL '78* [P78], 131–140. http://doi.acm.org/10.1145/512760.512774

[P78C] David Harel and Vaughan R. Pratt. Nondeterminism in logics of programs. In *POPL '78* [P78], 203–213. http://doi.acm.org/10.1145/512760.512782

[P79] **POPL '79**: *Proc. 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1979. Association for Computing Machinery.

[P79A] Robert Cartwright and John McCarthy. First order programming logic. In *POPL '79* [P79], 68–80. http://doi.acm.org/10.1145/567752.567759

[P79B] David Harel. Recursion in logics of programs. In *POPL '79* [P79], 81–92. http://doi.acm.org/10.1145/567752.567760

[P79C] V. R. Pratt. Process logic: Preliminary report. In *POPL '79* [P79], 93–100. http://doi.acm.org/10.1145/567752.567761

[P79D] Marco A. Casanova and Philip A. Bernstein. The logic of a relational data manipulation language. In *POPL '79* [P79], 101–109. http://doi.acm.org/10.1145/567752.567762

[P79E] Richard P. Reitman and Gregory R. Andrews. Certifying information flow properties of programs: An axiomatic approach. In *POPL '79* [P79], 283–290. http://doi.acm.org/10.1145/567752.567779

[P80] **POPL '80**: *Proc. 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1980. Association for Computing Machinery. ISBN 0-89791-011-7.

[P80A] Rohit Parikh. Propositional logics of programs: Systems, models, and complexity. In *POPL '80* [P80], 186–192. http://doi.acm.org/10.1145/567446.567464

[P81] **POPL '81**: *Proc. 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1981. Association for Computing Machinery. ISBN 0-89791-029-X.

[P81A] William L. Scherlis. Program improvement by internal specialization. In *POPL '81* [P81], 41–49. http://doi.acm.org/10.1145/567532.567536

[P81B] Ashok K. Chandra. Programming primitives for database languages. In *POPL '81* [P81], 50–62. http://doi.acm.org/10.1145/567532.567537

[P81C] D. J. Kuck, R. H. Kuhn, D. A. Padua, B. Leasure, and M. Wolfe. Dependence graphs and compiler optimizations. In *POPL '81* [P81], 207–218. http://doi.acm.org/10.1145/567532.567555

[P82] **POPL '82**: *Proc. 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1982. Association for Computing Machinery. ISBN 0-89791-065-6.

[P82A] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *POPL '82* [P82], 207–212. http://doi.acm.org/10.1145/582153.582176

[P82B] Shaula Yemini. An axiomatic treatment of exception handling. In *POPL '82* [P82], 281–288. http://doi.acm.org/10.1145/582153.582183

[P83] **POPL '83**: *Proc. 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1983. Association for Computing Machinery. ISBN 0-89791-090-7.

[P83A] Leslie Lamport. Reasoning about nonatomic operations. In *POPL '83* [P83], 28–37. http://doi.acm.org/10.1145/567067.567072

[P83B] Alan Demers and James Donahue. Making variables abstract: An equational theory for russell. In *POPL '83* [P83], 59–72. http://doi.acm.org/10.1145/567067.567075

[P83C] Daniel Leivant. Polymorphic type inference. In *POPL '83* [P83], 88–98. http://doi.acm.org/10.1145/567067.567077

[P84] **POPL '84**: *Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1984. Association for Computing Machinery. ISBN 0-89791-125-3. 549840.

[P84A] Thomas Reps and Bowen Alpern. Interactive proof checking. In *POPL '84* [P84], 36–45. http://doi.acm.org/10.1145/800017.800514 549840.

[P84B] David MacQueen, Gordon Plotkin, and Ravi Sethi. An ideal model for recursive polymorphic types. In *POPL '84* [P84], 165–174. http://doi.acm.org/10.1145/800017.800528 549840.

[P84C] Joseph Y. Halpern. A good hoare axiom system for an algol-like language. In *POPL '84* [P84], 262–271. http://doi.acm.org/10.1145/800017.800538 549840.

[P85] **POPL '85**: *Proc. 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1985. Association for Computing Machinery. ISBN 0-89791-147-4. 549850.

[P85A] Prateek Mishra and Uday S. Reddy. Declaration-free type checking. In *POPL '85* [P85], 7–21. http://doi.acm.org/10.1145/318593.318603 549850.

[P85B] John C. Mitchell and Gordon D. Plotkin. Abstract types have existential types. In *POPL '85* [P85], 37–51. http://doi.acm.org/10.1145/318593.318606 549850.

[P85C] Van Nguyen, David Gries, and Susan Owicki. A model and temporal proof system for networks of processes. In *POPL '85* [P85], 121–131. http://doi.acm.org/10.1145/318593.318624 549850.

[P85D] Daniel Leivant. Logical and mathematical reasoning about imperative programs: Preliminary report. In *POPL '85* [P85], 132–140. http://doi.acm.org/10.1145/318593.318625 549850.

[P85E] Leslie Lamport and Fred B. Schneider. Constraints: A uniform approach to aliasing and typing. In *POPL '85* [P85], 205–216. http://doi.acm.org/10.1145/318593.318640 549850.

[P86] **POPL '86**: *Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1986. Association for Computing Machinery.

[P86A] Howard Barringer, Ruurd Kuiper, and Amir Pnueli. A really abstract concurrent model and its temporal logic. In *POPL '86* [P86], 173–183. http://doi.acm.org/10.1145/512644.512660

[P86B] Pierre America, Jaco de Bakker, Joost N. Kok, and Jan J. M. M. Rutten. Operational semantics of a parallel object-oriented language. In *POPL '86* [P86], 194–208. http://doi.acm.org/10.1145/512644.512662

[P86C] Paul Hudak and Lauren Smith. Para-functional programming: A paradigm for programming multiprocessor systems. In *POPL '86* [P86], 243–254. http://doi.acm.org/10.1145/512644.512667

[P86D] John C. Mitchell. Representation independence and data abstraction. In *POPL '86* [P86], 263–276. http://doi.acm.org/10.1145/512644.512669

[P86E] Albert R. Meyer and Mark B. Reinhold. "type" is not a type. In *POPL '86* [P86], 287–295. http://doi.acm.org/10.1145/512644.512671

[P87] **POPL '87**: *Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, 1987. Association for Computing Machinery. ISBN 0-89791-215-2.

[P87A] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In *POPL '87* [P87], 1–2. http://doi.acm.org/10.1145/41625.41626

[P87B] J. Widom, D. Gries, and F. B. Schneider. Completeness and incompleteness of trace-based network proof systems. In *POPL '87* [P87], 27–38. http://doi.acm.org/10.1145/41625.41628

[P87C] V. A. Saraswat. The concurrent logic programming language cp: Definition and operational semantics. In *POPL '87* [P87], 49–62. http://doi.acm.org/10.1145/41625.41630

[P87D] T.-M. Kuo and P. Mishra. On strictness and its analysis. In *POPL '87* [P87], 144–155. http://doi.acm.org/10.1145/41625.41638

[P87E] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for real-time programming. In *POPL '87* [P87], 178–188. http://doi.acm.org/10.1145/41625.41641

[P87F] C. Huizing, R. Gerth, and W. P. deRoever. Full abstraction of a real-time denotational semantics for an occam-like language. In *POPL '87* [P87], 223–236. http://doi.acm.org/10.1145/41625.41645

[P87G] A. R. Meyer, J. C. Mitchell, E. Moggi, and R. Statman. Empty types in polymorphic lambda calculus. In *POPL '87* [P87], 253–262. http://doi.acm.org/10.1145/41625.41648

[P88] **POPL '88**: *Proc. 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1988. Association for Computing Machinery. ISBN 0-89791-252-7.

[P88A] J. C. Mitchell and R. Harper. The essence of ml. In *POPL '88* [P88], 28–46. http://doi.acm.org/10.1145/73560.73563

[P88B] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *POPL '88* [P88], 47–57. http://doi.acm.org/10.1145/73560.73564

[P88C] A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. A proper extension of ml with an effective type-assignment. In *POPL '88* [P88], 58–69. http://doi.acm.org/10.1145/73560.73565

[P88D] L. Cardelli. Structural subtyping and the notion of power type. In *POPL '88* [P88], 70–79. http://doi.acm.org/10.1145/73560.73566

[P88E] S. Kamin. Inheritance in smalltalk-80: A denotational definition. In *POPL '88* [P88], 80–87. http://doi.acm.org/10.1145/73560.73567

[P88F] R. Stansifer. Type inference with subtypes. In *POPL '88* [P88], 88–97. http://doi.acm.org/10.1145/73560.73568

[P88G] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. In *POPL '88* [P88], 229–239. http://doi.acm.org/10.1145/73560.73580

[P89] **POPL '89**: *Proc. 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1989. Association for Computing Machinery. ISBN 0-89791-294-2.

[P89A] P. Wadler and S. Blott. How to make ad-hoc polymorphism less ad hoc. In *POPL '89* [P89], 60–76. http://doi.acm.org/10.1145/75277.75283

[P89B] D. Rémy. Typechecking records and variants in a natural extension of ml. In *POPL '89* [P89], 77–88. http://doi.acm.org/10.1145/75277.75284

[P89C] C. Paulin-Mohring. Extracting F_ω's programs from proofs in the calculus of constructions. In *POPL '89* [P89], 89–104. http://doi.acm.org/10.1145/75277.75285

[P89D] B. Thomsen. A calculus of higher order communicating systems. In *POPL '89* [P89], 143–154. http://doi.acm.org/10.1145/75277.75290

[P89E] M. Abadi, L. Cardelli, B. Pierce, and G. Plotkin. Dynamic typing in a statically-typed language. In *POPL '89* [P89], 213–227. http://doi.acm.org/10.1145/75277.75296

[P89F] J. Meseguer. Relating models of polymorphism. In *POPL '89* [P89], 228–241. http://doi.acm.org/10.1145/75277.75297

[P90] **POPL '90**: *Proc. 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1990. Association for Computing Machinery. ISBN 0-89791-343-4.

[P90A] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy. Explicit substitutions. In *POPL '90* [P90], 31–46. http://doi.acm.org/10.1145/96709.96712

[P90B] Timothy G. Griffin. A formulae-as-type notion of control. In *POPL '90* [P90], 47–58. http://doi.acm.org/10.1145/96709.96714

[P90C] Andrea Asperti, Gian Luigi Ferrari, and Roberto Gorrieri. Implicative formulae in the "proofs of computations" analogy. In *POPL '90* [P90], 59–71. http://doi.acm.org/10.1145/96709.96715

[P90D] Gerard Berry and Gerard Boudol. The chemical abstract machine. In *POPL '90* [P90], 81–94. http://doi.acm.org/10.1145/96709.96717

[P90E] Yves Lafont. Interaction nets. In *POPL '90* [P90], 95–108. http://doi.acm.org/10.1145/96709.96718

[P90F] John C. Mitchell. Toward a typed foundation for method specialization and inheritance. In *POPL '90* [P90], 109–124. http://doi.acm.org/10.1145/96709.96719

[P90G] William R. Cook, Walter Hill, and Peter S. Canning. Inheritance is not subtyping. In *POPL '90* [P90], 125–135. http://doi.acm.org/10.1145/96709.96721

[P90H] Justin O. Graver and Ralph E. Johnson. A type system for smalltalk. In *POPL '90* [P90], 136–150. http://doi.acm.org/10.1145/96709.96722

[P90I] Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *POPL '90* [P90], 232–245. http://doi.acm.org/10.1145/96709.96733

[P90J] Carl A. Gunter. Relating total and partial correctness interpretations of non-deterministic programs. In *POPL '90* [P90], 306–319. http://doi.acm.org/10.1145/96709.96741

[P90K] Robert Harper, John C. Mitchell, and Eugenio Moggi. Higher-order modules and the phase distinction. In *POPL '90* [P90], 341–354. http://doi.acm.org/10.1145/96709.96744

[P90L] Francois Rouaix. Safe run-time overloading. In *POPL '90* [P90], 355–366. http://doi.acm.org/10.1145/96709.96746

[P90M] Satish Thatte. Quasi-static typing. In *POPL '90* [P90], 367–381. http://doi.acm.org/10.1145/96709.96747

[P91] **POPL '91**: *Proc. 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1991. Association for Computing Machinery. ISBN 0-89791-419-8.

[P91A] Keshav Pingali, Micah Beck, Richard Johnson, Mayan Moudgill, and Paul Stodghill. Dependence flow graphs: An algebraic approach to program dependencies. In *POPL '91* [P91], 67–78. http://doi.acm.org/10.1145/99583.99595

[P91B] Robert Harper and Benjamin Pierce. A record calculus based on symmetric concatenation. In *POPL '91* [P91], 131–142. http://doi.acm.org/10.1145/99583.99603

[P91C] William Clinger and Jonathan Rees. Macros that work. In *POPL '91* [P91], 155–162. http://doi.acm.org/10.1145/99583.99607

[P91D] Bruce Duba, Robert Harper, and David MacQueen. Typing first-class continuations in ml. In *POPL '91* [P91], 163–173. http://doi.acm.org/10.1145/99583.99608

[P91E] Alon Kleinman, Yael Moscowitz, Amir Pnueli, and Ehud Sharpio. Communication with directed logic variables. In *POPL '91* [P91], 221–232. http://doi.acm.org/10.1145/99583.99615

[P91F] Jon G. Riecke. Fully abstract translations between functional languages. In *POPL '91* [P91], 245–254. http://doi.acm.org/10.1145/99583.99617

[P91G] Luc Maranget. Optimal derivations in weak lambda-calculi and in orthogonal term rewriting systems. In *POPL '91* [P91], 255–269. http://doi.acm.org/10.1145/99583.99618

[P91H] Alex Aiken and Brian Murphy. Static type inference in a dynamically typed language. In *POPL '91* [P91], 279–290. http://doi.acm.org/10.1145/99583.99621

[P91I] Xavier Leroy and Pierre Weis. Polymorphic type inference and assignment. In *POPL '91* [P91], 291–302. http://doi.acm.org/10.1145/99583.99622

[P91J] Pierre Jouvelot and David Gifford. Algebraic reconstruction of types and effects. In *POPL '91* [P91], 303–310. http://doi.acm.org/10.1145/99583.99623

[P91K] Martin Abadi and Gordon Plotkin. A logical view of composition and refinement. In *POPL '91* [P91], 323–332. http://doi.acm.org/10.1145/99583.99626

[P91L] Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91* [P91], 333–352. http://doi.acm.org/10.1145/99583.99627

[P92] **POPL '92**: *Proc. 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1992. Association for Computing Machinery. ISBN 0-89791-453-8. 549920.

[P92A] Andrzej Filinski. Linear continuations. In *POPL '92* [P92], 27–38. http://doi.acm.org/10.1145/143165.143174 549920.

[P92B] Patrick Cousot and Radhia Cousot. Inductive definitions, semantics and abstract interpretations. In *POPL '92* [P92], 83–94. http://doi.acm.org/10.1145/143165.143184 549920.

[P92C] Martin C. Rinard and Monica S. Lam. Semantic foundations of jade. In *POPL '92* [P92], 105–118. http://doi.acm.org/10.1145/143165.143189 549920.

[P92D] Dave Berry, Robin Milner, and David N. Turner. A semantics for ml concurrency primitives. In *POPL '92* [P92], 119–129. http://doi.acm.org/10.1145/143165.143191 549920.

[P92E] Atsushi Ohori. A compilation method for ml-style polymorphic record calculi. In *POPL '92* [P92], 154–165. http://doi.acm.org/10.1145/143165.143200 549920.

[P92F] Xavier Leroy. Unboxed objects and polymorphic typing. In *POPL '92* [P92], 177–188. http://doi.acm.org/10.1145/143165.143205 549920.

[P92G] Mads Tofte. Principal signatures for higher-order program modules. In *POPL '92* [P92], 189–199. http://doi.acm.org/10.1145/143165.143206 549920.

[P92H] Robero Di Cosmo. Type isomorphisms in a type-assignment framework. In *POPL '92* [P92], 200–210. http://doi.acm.org/10.1145/143165.143208 549920.

[P92I] QingMing Ma. Parametricity as subtyping. In *POPL '92* [P92], 281–292. http://doi.acm.org/10.1145/143165.143225 549920.

[P92J] Patrick Lincoln and John C. Mitchell. Algorithmic aspects of type inference with subtypes. In *POPL '92* [P92], 293–304. http://doi.acm.org/10.1145/143165.143227 549920.

[P92K] Benjamin C. Pierce. Bounded quantification is undecidable. In *POPL '92* [P92], 305–315. http://doi.acm.org/10.1145/143165.143228 549920.

[P93] **POPL '93**: *Proc. 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1993. Association for Computing Machinery. ISBN 0-89791-560-7. 549930.

[P93A] Stephen Weeks and Matthias Felleisen. On the orthogonality of assignments and procedures in algol. In *POPL '93* [P93], 57–70. http://doi.acm.org/10.1145/158511.158523 549930.

[P93B] G. Berry, S. Ramesh, and R. K. Shyamasundar. Communicating reactive processes. In *POPL '93* [P93], 85–98. http://doi.acm.org/10.1145/158511.158526 549930.

[P93C] Atsushi Ohori and Kazuhiko Kato. Semantics for communication primitives in a polymorphic language. In *POPL '93* [P93], 99–112. http://doi.acm.org/10.1145/158511.158529 549930.

[P93D] Julia L. Lawall and Olivier Danvy. Separating stages in the continuation-passing style transformation. In *POPL '93* [P93], 124–136. http://doi.acm.org/10.1145/158511.158613 549930.

[P93E] Mitchell Wand. Specifying the correctness of binding-time analysis. In *POPL '93* [P93], 137–143. http://doi.acm.org/10.1145/158511.158614 549930.

[P93F] John Launchbury. A natural semantics for lazy evaluation. In *POPL '93* [P93], 144–154. http://doi.acm.org/10.1145/158511.158618 549930.

[P93G] Martín Abadi, Luca Cardelli, and Pierre-Louis Curien. Formal parametric polymorphism. In *POPL '93* [P93], 157–170. http://doi.acm.org/10.1145/158511.158622 549930.

[P93H] Jon G. Riecke and Ramesh Subrahmanyam. Algebraic reasoning and completeness in typed languages. In *POPL '93* [P93], 185–195. http://doi.acm.org/10.1145/158511.158627 549930.

[P93I] Robert Harper and Mark Lillibridge. Explicit polymorphism and cps conversion. In *POPL '93* [P93], 206–219. http://doi.acm.org/10.1145/158511.158630 549930.

[P93J] Xavier Leroy. Polymorphism by name for references and continuations. In *POPL '93* [P93], 220–231. http://doi.acm.org/10.1145/158511.158632 549930.

[P93K] Kim B. Bruce. Safe type checking in a statically-typed object-oriented programming language. In *POPL '93* [P93], 285–298. http://doi.acm.org/10.1145/158511.158650 549930.

[P93L] Benjamin C. Pierce and David N. Turner. Object-oriented programming without recursive types. In *POPL '93* [P93], 299–312. http://doi.acm.org/10.1145/158511.158653 549930.

[P93M] Harry G. Mairson. A constructive logic of multiple subtyping. In *POPL '93* [P93], 313–324. http://doi.acm.org/10.1145/158511.158657 549930.

[P93N] E. Villemonte de la Clergerie. Layer sharing: An improved structure-sharing framework. In *POPL '93* [P93], 345–358. http://doi.acm.org/10.1145/158511.158687 549930.

[P93O] A. Bossi, M. Bugliesi, M. Gabbrielli, G. Levi, and M. C. Meo. Differential logic programming. In *POPL '93* [P93], 359–370. http://doi.acm.org/10.1145/158511.158689 549930.

[P93P] Tobias Nipkow and Christian Prehofer. Type checking type classes. In *POPL '93* [P93], 409–418. http://doi.acm.org/10.1145/158511.158698 549930.

[P93Q] Simon J. Gay. A sort inference algorithm for the polyadic &pgr;-calculus. In *POPL '93* [P93], 429–438. http://doi.acm.org/10.1145/158511.158701 549930.

[P93R] Maria Virginia Aponte. Extending record typing to type parametric modules with sharing. In *POPL '93* [P93], 465–478. http://doi.acm.org/10.1145/158511.158704 549930.

[P94] **POPL '94**: *Proc. 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1994. Association for Computing Machinery. ISBN 0-89791-636-0. 549940.

[P94A] Jacques Garrigue and Hassan Aït-Kaci. The typed polymorphic label-selective λ-calculus. In *POPL '94* [P94], 35–47. http://doi.acm.org/10.1145/174675.174434 549940.

[P94B] Martin Odersky. A functional theory of local names. In *POPL '94* [P94], 48–59. http://doi.acm.org/10.1145/174675.175187 549940.

[P94C] Pierre Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$: A journey through calculi of explicit substitutions. In *POPL '94* [P94], 60–69. http://doi.acm.org/10.1145/174675.174707 549940.

[P94D] Hanne Riis Nielson and Flemming Nielson. Higher-order concurrent programs with finite communication topology (extended abstract). In *POPL '94* [P94], 84–97. http://doi.acm.org/10.1145/174675.174538 549940.

[P94E] Frank S. de Boer, Maurizio Gabbrielli, Elena Marchiori, and Catuscia Palamidessi. Proving concurrent constraint programs correct. In *POPL '94* [P94], 98–108. http://doi.acm.org/10.1145/174675.176925 549940.

[P94F] Xavier Leroy. Manifest types, modules, and separate compilation. In *POPL '94* [P94], 109–122. http://doi.acm.org/10.1145/174675.176926 549940.

[P94G] Robert Harper and Mark Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *POPL '94* [P94], 123–137. http://doi.acm.org/10.1145/174675.176927 549940.

[P94H] Giuseppe Castagna and Benjamin C. Pierce. Decidable bounded quantification. In *POPL '94* [P94], 151–162. http://doi.acm.org/10.1145/174675.177844 549940.

[P94I] Alexander Aiken, Edward L. Wimmers, and T. K. Lakshman. Soft typing with conditional types. In *POPL '94* [P94], 163–173. http://doi.acm.org/10.1145/174675.177847 549940.

[P94J] Satish R. Thatté. Automated synthesis of interface adapters for reusable classes. In *POPL '94* [P94], 174–187. http://doi.acm.org/10.1145/174675.177850 549940.

[P94K] Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value λ-calculus using a stack of regions. In *POPL '94* [P94], 188–201. http://doi.acm.org/10.1145/174675.177855 549940.

[P94L] Chris Hankin and Daniel Le Métayer. Deriving algorithms from type inference systems: Application to strictness analysis. In *POPL '94* [P94], 202–212. http://doi.acm.org/10.1145/174675.177858 549940.

[P94M] Fritz Henglein and Jesper Jørgensen. Formally optimal boxing. In *POPL '94* [P94], 213–226. http://doi.acm.org/10.1145/174675.177874 549940.

[P94N] Zhenyu Qian. Higher-order equational logic programming. In *POPL '94* [P94], 254–267. http://doi.acm.org/10.1145/174675.177889 549940.

[P94O] Bard Bloom. Chocolate: Calculi of higher order communication and lambda terms (preliminary report). In *POPL '94* [P94], 339–347. http://doi.acm.org/10.1145/174675.177948 549940.

[P94P] Mitchell Wand and Paul Steckler. Selective and lightweight closure conversion. In *POPL '94* [P94], 435–445. http://doi.acm.org/10.1145/174675.178044 549940.

[P94Q] Andrzej Filinski. Representing monads. In *POPL '94* [P94], 446–457. http://doi.acm.org/10.1145/174675.178047 549940.

[P94R] John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In *POPL '94* [P94], 458–471. http://doi.acm.org/10.1145/174675.178053 549940.

[P95] **POPL '95**: *Proc. 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1995. Association for Computing Machinery. ISBN 0-89791-692-1. 549950.

[P95A] Jon G. Riecke and Ramesh Viswanathan. Isolating side effects in sequential languages. In *POPL '95* [P95], 1–12. http://doi.acm.org/10.1145/199448.199450 549950.

[P95B] Catherine Dubois, François Rouaix, and Pierre Weis. Extensional polymorphism. In *POPL '95* [P95], 118–129. http://doi.acm.org/10.1145/199448.199473 549950.

[P95C] Robert Harper and Greg Morrisett. Compiling polymorphism using intensional type analysis. In *POPL '95* [P95], 130–141. http://doi.acm.org/10.1145/199448.199475 549950.

[P95D] Xavier Leroy. Applicative functors and fully transparent higher-order modules. In *POPL '95* [P95], 142–153. http://doi.acm.org/10.1145/199448.199476 549950.

[P95E] Sandip K. Biswas. Higher-order functors with transparent signatures. In *POPL '95* [P95], 154–163. http://doi.acm.org/10.1145/199448.199478 549950.

[P95F] Sergei G. Vorobyov. Structural decidable extensions of bounded quantification. In *POPL '95* [P95], 164–175. http://doi.acm.org/10.1145/199448.199479 549950.

[P95G] My Hoang and John C. Mitchell. Lower bounds on type inference with subtypes. In *POPL '95* [P95], 176–185. http://doi.acm.org/10.1145/199448.199481 549950.

[P95H] Martin Hofmann and Benjamin Pierce. Positive subtyping. In *POPL '95* [P95], 186–197. http://doi.acm.org/10.1145/199448.199482 549950.

[P95I] Cormac Flanagan and Matthias Felleisen. The semantics of future and its use in program optimization. In *POPL '95* [P95], 209–220. http://doi.acm.org/10.1145/199448.199484 549950.

[P95J] Vijay A. Saraswat, Radha Jagadeesan, and Vineet Gupta. Default timed concurrent constraint programming. In *POPL '95* [P95], 272–285. http://doi.acm.org/10.1145/199448.199513 549950.

[P95K] Patrick M. Sansom and Simon L. Peyton Jones. Time and space profiling for non-strict, higher-order functional languages. In *POPL '95* [P95], 355–366. http://doi.acm.org/10.1145/199448.199531 549950.

[P95L] Jens Palsberg and Patrick O'Keefe. A type system equivalent to flow analysis. In *POPL '95* [P95], 367–378. http://doi.acm.org/10.1145/199448.199533 549950.

[P95M] John Field, G. Ramalingam, and Frank Tip. Parametric program slicing. In *POPL '95* [P95], 379–392. http://doi.acm.org/10.1145/199448.199534 549950.

[P96] **POPL '96**: *Proc. 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1996. Association for Computing Machinery. ISBN 0-89791-769-3.

[P96A] Bjarne Steensgaard. Points-to analysis in almost linear time. In *POPL '96* [P96], 32–41. http://doi.acm.org/10.1145/237721.237727

[P96B] Trevor Jim. What are principal typings and what are they good for? In *POPL '96* [P96], 42–53. http://doi.acm.org/10.1145/237721.237728

[P96C] Martin Odersky and Konstantin Läufer. Putting type annotations to work. In *POPL '96* [P96], 54–67. http://doi.acm.org/10.1145/237721.237729

[P96D] Mark P. Jones. Using parameterized signatures to express modular structure. In *POPL '96* [P96], 68–78. http://doi.acm.org/10.1145/237721.237731

[P96E] Daniel Jackson, Somesh Jha, and Craig A. Damon. Faster checking of software specifications by eliminating isomorphs. In *POPL '96* [P96], 79–90. http://doi.acm.org/10.1145/237721.237733

[P96F] Lars Birkedal, Mads Tofte, and Magnus Vejlstrup. From region inference to von neumann machines via region representation inference. In *POPL '96* [P96], 171–183. http://doi.acm.org/10.1145/237721.237771

[P96G] Christopher Colby and Peter Lee. Trace-based program analysis. In *POPL '96* [P96], 195–207. http://doi.acm.org/10.1145/237721.237776

[P96H] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In *POPL '96* [P96], 258–270. http://doi.acm.org/10.1145/237721.237788

[P96I] Yasuhiko Minamide, Greg Morrisett, and Robert Harper. Typed closure conversion. In *POPL '96* [P96], 271–283. http://doi.acm.org/10.1145/237721.237791

[P96J] Leonidas Fegaras and Tim Sheard. Revisiting catamorphisms over datatypes with embedded functions (or, programs from outer space). In *POPL '96* [P96], 284–294. http://doi.acm.org/10.1145/237721.237792

[P96K] John Greiner and Guy E. Blelloch. A provably time-efficient parallel implementation of full speculation. In *POPL '96* [P96], 309–321. http://doi.acm.org/10.1145/237721.237797

[P96L] Joachim Niehren. Functional computation as concurrent computation. In *POPL '96* [P96], 333–343. http://doi.acm.org/10.1145/237721.237801

[P96M] Kohei Honda. Composing processes. In *POPL '96* [P96], 344–357. http://doi.acm.org/10.1145/237721.237802

[P96N] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. In *POPL '96* [P96], 358–371. http://doi.acm.org/10.1145/237721.237804

[P96O] Andrew D. Gordon and Gareth D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *POPL '96* [P96], 386–395. http://doi.acm.org/10.1145/237721.237807

[P96P] Martín Abadi, Luca Cardelli, and Ramesh Viswanathan. An interpretation of objects and object types. In *POPL '96* [P96], 396–409. http://doi.acm.org/10.1145/237721.237809

[P96Q] John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *POPL '96* [P96], 410–423. http://doi.acm.org/10.1145/237721.240882

[P97] **POPL '97**: *Proc. 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1997. Association for Computing Machinery. ISBN 0-89791-853-3.

[P97A] Pascal Fradet and Daniel Le Métayer. Shape types. In *POPL '97* [P97], 27–39. http://doi.acm.org/10.1145/263699.263706

[P97B] Didier Rémy and Jérôme Vouillon. Objective ml: A simple object-oriented extension of ml. In *POPL '97* [P97], 40–53. http://doi.acm.org/10.1145/263699.263707

[P97C] Chih-Ping Chen and Paul Hudak. Rolling your own mutable adt—a connection between linear types and monads. In *POPL '97* [P97], 54–66. http://doi.acm.org/10.1145/263699.263708

[P97D] George C. Necula. Proof-carrying code. In *POPL '97* [P97], 106–119. http://doi.acm.org/10.1145/263699.263712

[P97E] Martin Odersky and Philip Wadler. Pizza into java: Translating theory into practice. In *POPL '97* [P97], 146–159. http://doi.acm.org/10.1145/263699.263715

[P97F] C.-H. L. Ong and C. A. Stewart. A curry-howard foundation for functional computation with control. In *POPL '97* [P97], 215–227. http://doi.acm.org/10.1145/263699.263722

[P97G] Gérard Boudol. The π-calculus in direct style. In *POPL '97* [P97], 228–242. http://doi.acm.org/10.1145/263699.263726

[P97H] Benjamin C. Pierce and Davide Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *POPL '97* [P97], 242–255. http://doi.acm.org/10.1145/263699.263729

[P97I] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous &pgr;-calculus. In *POPL '97* [P97], 256–265. http://doi.acm.org/10.1145/263699.263731

[P97J] Luca Cardelli. Program fragments, linking, and modularization. In *POPL '97* [P97], 266–277. http://doi.acm.org/10.1145/263699.263735

[P97K] Jakob Rehof. Minimal typings in atomic subtyping. In *POPL '97* [P97], 278–291. http://doi.acm.org/10.1145/263699.263738

[P97L] Amokrane Saïbi. Typing algorithm in type theory with inheritance. In *POPL '97* [P97], 292–301. http://doi.acm.org/10.1145/263699.263742

[P97M] François Bourdoncle and Stephan Merz. Type checking higher-order polymorphic multi-methods. In *POPL '97* [P97], 302–315. http://doi.acm.org/10.1145/263699.263743

[P97N] Patrick Cousot. Types as abstract interpretations. In *POPL '97* [P97], 316–331. http://doi.acm.org/10.1145/263699.263744

[P97O] Flemming Nielson and Hanne Riis Nielson. Infinitary control flow analysis: A collecting semantics for closure analysis. In *POPL '97* [P97], 332–345. http://doi.acm.org/10.1145/263699.263745

[P97P] Sandip K. Biswas. A demand-driven set-based analysis. In *POPL '97* [P97], 372–385. http://doi.acm.org/10.1145/263699.263753

[P97Q] Mitchell Wand and Gregory T. Sullivan. Denotational semantics using an operationally-based term model. In *POPL '97* [P97], 386–399. http://doi.acm.org/10.1145/263699.263755

[P97R] David Sands. From sos rules to proof principles: An operational metatheory for functional languages. In *POPL '97* [P97], 428–441. http://doi.acm.org/10.1145/263699.263760

[P97S] Patrik Jansson and Johan Jeuring. Polyp—a polytypic programming language extension. In *POPL '97* [P97], 470–482. http://doi.acm.org/10.1145/263699.263763

[P97T] Mark P. Jones. First-class polymorphism with type inference. In *POPL '97* [P97], 483–496. http://doi.acm.org/10.1145/263699.263765

[P98] **POPL '98**: *Proc. 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1998. Association for Computing Machinery. ISBN 0-89791-979-3.

[P98A] John Hannan and Patrick Hicks. Higher-order uncurrying. In *POPL '98* [P98], 1–11. http://doi.acm.org/10.1145/268946.268947

[P98B] Simon Peyton Jones, Mark Shields, John Launchbury, and Andrew Tolmach. Bridging the gulf: A common intermediate language for ml and haskell. In *POPL '98* [P98], 49–61. http://doi.acm.org/10.1145/268946.268951

[P98C] Zena M. Ariola and Amr Sabry. Correctness of monadic state: An imperative call-by-need calculus. In *POPL '98* [P98], 62–74. http://doi.acm.org/10.1145/268946.268952

[P98D] Yasuhiko Minamide. A functional representation of data structures with a hole. In *POPL '98* [P98], 75–84. http://doi.acm.org/10.1145/268946.268953

[P98E] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system f to typed assembly language. In *POPL '98* [P98], 85–97. http://doi.acm.org/10.1145/268946.268954

[P98F] Raymie Stata and Martín Abadi. A type system for java bytecode subroutines. In *POPL '98* [P98], 149–160. http://doi.acm.org/10.1145/268946.268959

[P98G] Tobias Nipkow and David von Oheimb. Javalight is type-safe—definitely. In *POPL '98* [P98], 161–170. http://doi.acm.org/10.1145/268946.268960

[P98H] Matthew Flatt, Shriram Krishnamurthi, and Matthias Felleisen. Classes and mixins. In *POPL '98* [P98], 171–183. http://doi.acm.org/10.1145/268946.268961

[P98I] Jens Palsberg and Christina Pavlopoulou. From polyvariant flow information to intersection and union types. In *POPL '98* [P98], 197–208. http://doi.acm.org/10.1145/268946.268963

[P98J] Thomas P. Jensen. Inference of polymorphic and conditional strictness properties. In *POPL '98* [P98], 209–221. http://doi.acm.org/10.1145/268946.268964

[P98K] Benjamin C. Pierce and David N. Turner. Local type inference. In *POPL '98* [P98], 252–265. http://doi.acm.org/10.1145/268946.268967

[P98L] Susumu Nishimura. Static typing for dynamic messages. In *POPL '98* [P98], 266–278. http://doi.acm.org/10.1145/268946.268968

[P98M] Mark Shields, Tim Sheard, and Simon Peyton Jones. Dynamic typing as staged type inference. In *POPL '98* [P98], 289–302. http://doi.acm.org/10.1145/268946.268970

[P98N] Zhenjiang Hu, Masato Takeichi, and Wei-Ngan Chin. Parallelization in calculational forms. In *POPL '98* [P98], 316–328. http://doi.acm.org/10.1145/268946.268972

[P98O] Alexander Aiken and David Gay. Barrier inference. In *POPL '98* [P98], 342–354. http://doi.acm.org/10.1145/268946.268974

[P98P] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *POPL '98* [P98], 355–364. http://doi.acm.org/10.1145/268946.268975

[P98Q] Nevin Heintze and Jon G. Riecke. The slam calculus: Programming with secrecy and integrity. In *POPL '98* [P98], 365–377. http://doi.acm.org/10.1145/268946.268976

[P98R] James Riely and Matthew Hennessy. A typed language for distributed mobile processes (extended abstract). In *POPL '98* [P98], 378–390. http://doi.acm.org/10.1145/268946.268978

[P98S] Xavier Leroy and François Rouaix. Security properties of typed applets. In *POPL '98* [P98], 391–403. http://doi.acm.org/10.1145/268946.268979

[P99] **POPL '99**: *Proc. 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1999. Association for Computing Machinery. ISBN 1-58113-095-3.

[P99A] Peter Harry Eidorff, Fritz Henglein, Christian Mossin, Henning Niss, Morten Heine Sørensen, and Mads Tofte. Annodomini: From type theory to year 2000 conversion tool. In *POPL '99* [P99], 1–14. http://doi.acm.org/10.1145/292540.292543

[P99B] Keith Wansbrough and Simon Peyton Jones. Once upon a polymorphic type. In *POPL '99* [P99], 15–28. http://doi.acm.org/10.1145/292540.292545

[P99C] Naoki Kobayashi. Quasi-linear types. In *POPL '99* [P99], 29–42. http://doi.acm.org/10.1145/292540.292546

[P99D] Andrew Moran and David Sands. Improvement in a lazy context: An operational theory for call-by-need. In *POPL '99* [P99], 43–56. http://doi.acm.org/10.1145/292540.292547

[P99E] Robert O'Callahan. A simple, comprehensive type system for java bytecode subroutines. In *POPL '99* [P99], 70–78. http://doi.acm.org/10.1145/292540.292549

[P99F] Luca Cardelli and Andrew D. Gordon. Types for mobile ambients. In *POPL '99* [P99], 79–92. http://doi.acm.org/10.1145/292540.292550

[P99G] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *POPL '99* [P99], 93–104. http://doi.acm.org/10.1145/292540.292551

[P99H] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *POPL '99* [P99], 147–160. http://doi.acm.org/10.1145/292540.292555

[P99I] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *POPL '99* [P99], 161–174. http://doi.acm.org/10.1145/292540.292556

[P99J] Andrzej Filinski. Representing layered monads. In *POPL '99* [P99], 175–188. http://doi.acm.org/10.1145/292540.292557

[P99K] Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Stochastic processes as concurrent constraint programs. In *POPL '99* [P99], 189–202. http://doi.acm.org/10.1145/292540.292558

[P99L] Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *POPL '99* [P99], 214–227. http://doi.acm.org/10.1145/292540.292560

[P99M] Andrew C. Myers. Jflow: Practical mostly-static information flow control. In *POPL '99* [P99], 228–241. http://doi.acm.org/10.1145/292540.292561

[P99N] Neal Glew and Greg Morrisett. Type-safe linking and modular assembly language. In *POPL '99* [P99], 250–261. http://doi.acm.org/10.1145/292540.292563

[P99O] Karl Crary, David Walker, and Greg Morrisett. Typed memory management in a calculus of capabilities. In *POPL '99* [P99], 262–275. http://doi.acm.org/10.1145/292540.292564

[P99P] François Pessaux and Xavier Leroy. Type-based analysis of uncaught exceptions. In *POPL '99* [P99], 276–290. http://doi.acm.org/10.1145/292540.292565

[P99Q] Mitchell Wand and Igor Siveroni. Constraint systems for useless variable elimination. In *POPL '99* [P99], 291–302. http://doi.acm.org/10.1145/292540.292567

[P00] **POPL '00**: *Proc. 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2000. Association for Computing Machinery. ISBN 1-58113-125-9. 549201.

[P00A] Javier Esparza and Andreas Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL '00* [P00], 1–11. http://doi.acm.org/10.1145/325694.325697 549201.

[P00B] Nevin Heintze, Joxan Jaffar, and Răzvan Voicu. A framework for combining analysis and verification. In *POPL '00* [P00], 26–39. http://doi.acm.org/10.1145/325694.325700 549201.

[P00C] Johan Agat. Transforming out timing leaks. In *POPL '00* [P00], 40–53. http://doi.acm.org/10.1145/325694.325702 549201.

[P00D] Thomas Colcombet and Pascal Fradet. Enforcing trace properties by program transformation. In *POPL '00* [P00], 54–66. http://doi.acm.org/10.1145/325694.325703 549201.

[P00E] Zhendong Su, Manuel Fähndrich, and Alexander Aiken. Projection merging: Reducing redundancies in inclusion constraint graphs. In *POPL '00* [P00], 81–95. http://doi.acm.org/10.1145/325694.325706 549201.

[P00F] Andrea Asperti, Paolo Coppola, and Simone Martini. (optimal) duplication is not elementary recursive. In *POPL '00* [P00], 96–107. http://doi.acm.org/10.1145/325694.325707 549201.

[P00G] Jeffrey R. Lewis, John Launchbury, Erik Meijer, and Mark B. Shields. Implicit parameters: Dynamic scoping with static types. In *POPL '00* [P00], 108–118. http://doi.acm.org/10.1145/325694.325708 549201.

[P00H] Ralf Hinze. A new approach to generic functional programming. In *POPL '00* [P00], 119–132. http://doi.acm.org/10.1145/325694.325709 549201.

[P00I] Karl Crary and Stephanie Weirich. Resource bound certification. In *POPL '00* [P00], 184–198. http://doi.acm.org/10.1145/325694.325716 549201.

[P00J] Ben Liblit and Alexander Aiken. Type systems for distributed data structures. In *POPL '00* [P00], 199–213. http://doi.acm.org/10.1145/325694.325717 549201.

[P00K] Christopher A. Stone and Robert Harper. Deciding type equivalence in a language with singleton kinds. In *POPL '00* [P00], 214–227. http://doi.acm.org/10.1145/325694.325724 549201.

[P00L] Todd B. Knoblock and Jakob Rehof. Type elaboration and subtype completion for java bytecode. In *POPL '00* [P00], 228–242. http://doi.acm.org/10.1145/325694.325725 549201.

[P00M] Andrew W. Appel and Amy P. Felty. A semantic model of types and machine instructions for proof-carrying code. In *POPL '00* [P00], 243–253. http://doi.acm.org/10.1145/325694.325727 549201.

[P00N] David Walker. A type system for expressive security policies. In *POPL '00* [P00], 254–267. http://doi.acm.org/10.1145/325694.325728 549201.

[P00O] Dennis Volpano and Geoffrey Smith. Verifying secrets and relative secrecy. In *POPL '00* [P00], 268–276. http://doi.acm.org/10.1145/325694.325729 549201.

[P00P] Anders Sandholm and Michael I. Schwartzbach. A type system for dynamic web documents. In *POPL '00* [P00], 290–301. http://doi.acm.org/10.1145/325694.325733 549201.

[P00Q] Martín Abadi, Cédric Fournet, and Georges Gonthier. Authentication primitives and their compilation. In *POPL '00* [P00], 302–315. http://doi.acm.org/10.1145/325694.325734 549201.

[P00R] Carl A. Gunter and Trevor Jim. Generalized certificate revocation. In *POPL '00* [P00], 316–329. http://doi.acm.org/10.1145/325694.325736 549201.

[P00S] Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *POPL '00* [P00], 352–364. http://doi.acm.org/10.1145/325694.325741 549201.

[P01] **POPL '01**: *Proc. 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2001. Association for Computing Machinery. ISBN 1-58113-336-7.

[P01A] Davide Sangiorgi. Extensionality and intensionality of the ambient logics. In *POPL '01* [P01], 4–13. http://doi.acm.org/10.1145/360204.375707

[P01B] Samin S. Ishtiaq and Peter W. O'Hearn. Bi as an assertion language for mutable data structures. In *POPL '01* [P01], 14–26. http://doi.acm.org/10.1145/360204.375719

[P01C] Martin Odersky, Christoph Zenger, and Matthias Zenger. Colored local type inference. In *POPL '01* [P01], 41–53. http://doi.acm.org/10.1145/360204.360207

[P01D] Jakob Rehof and Manuel Fähndrich. Type-base flow analysis: From polymorphic subtyping to cfl-reachability. In *POPL '01* [P01], 54–66. http://doi.acm.org/10.1145/360204.360208

[P01E] Haruo Hosoya and Benjamin Pierce. Regular expression pattern matching for xml. In *POPL '01* [P01], 67–80. http://doi.acm.org/10.1145/360204.360209

[P01F] Harald Ganzinger. Efficient deductive methods for program analysis. In *POPL '01* [P01], 102–103. http://doi.acm.org/10.1145/360204.360212

[P01G] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL '01* [P01], 104–115. http://doi.acm.org/10.1145/360204.360213

[P01H] Asis Unyapoth and Peter Sewell. Nomadic pict: Correct communication infrastructure for mobile computation. In *POPL '01* [P01], 116–127. http://doi.acm.org/10.1145/360204.360214

[P01I] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. In *POPL '01* [P01], 128–141. http://doi.acm.org/10.1145/360204.360215

[P01J] George C. Necula and S. P. Rahul. Oracle-based checking of untrusted software. In *POPL '01* [P01], 142–154. http://doi.acm.org/10.1145/360204.360216

[P01K] Cristiano Calcagno. Stratified operational semantics for safety and correctness of the region calculus. In *POPL '01* [P01], 155–165. http://doi.acm.org/10.1145/360204.360217

[P01L] Michele Bugliesi and Giuseppe Castagna. Secure safe ambients. In *POPL '01* [P01], 222–235. http://doi.acm.org/10.1145/360204.360223

[P01M] Peter Sewell. Modules, abstract types, and distributed versioning. In *POPL '01* [P01], 236–247. http://doi.acm.org/10.1145/360204.360225

[P01N] Andrew D. Gordon and Don Syme. Typing a multi-language intermediate code. In *POPL '01* [P01], 248–260. http://doi.acm.org/10.1145/360204.360228

[P01O] Mark Shields and Erik Meijer. Type-indexed rows. In *POPL '01* [P01], 261–275. http://doi.acm.org/10.1145/360204.360230

[P01P] Joseph (Yossi) Gil. Subtyping arithmetical types. In *POPL '01* [P01], 276–289. http://doi.acm.org/10.1145/360204.360232

[P01Q] Jérôme Vouillon. Combining subsumption and binary methods: An object calculus with views. In *POPL '01* [P01], 290–303. http://doi.acm.org/10.1145/360204.360233

[P02] **POPL '02**: *Proc. 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2002. Association for Computing Machinery. ISBN 1-58113-450-9. 549021.

[P02A] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *POPL '02* [P02], 33–44. http://doi.acm.org/10.1145/503272.503277 549021.

[P02B] Sagar Chaki, Sriram K. Rajamani, and Jakob Rehof. Types as models: Model checking message-passing programs. In *POPL '02* [P02], 45–57. http://doi.acm.org/10.1145/503272.503278 549021.

[P02C] Massimo Merro and Matthew Hennessy. Bisimulation congruences in safe ambients. In *POPL '02* [P02], 71–80. http://doi.acm.org/10.1145/503272.503280 549021.

[P02D] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *POPL '02* [P02], 81–92. http://doi.acm.org/10.1145/503272.503281 549021.

[P02E] George C. Necula, Scott McPeak, and Westley Weimer. Ccured: Type-safe retrofitting of legacy code. In *POPL '02* [P02], 128–139. http://doi.acm.org/10.1145/503272.503286 549021.

[P02F] Anindya Banerjee and David A. Naumann. Representation independence, confinement and access control [extended abstract]. In *POPL '02* [P02], 166–177. http://doi.acm.org/10.1145/503272.503289 549021.

[P02G] Zhong Shao, Bratin Saha, Valery Trifonov, and Nikolaos Papaspyrou. A type system for certified binaries. In *POPL '02* [P02], 217–232. http://doi.acm.org/10.1145/503272.503293 549021.

[P02H] Matthias Neubauer, Peter Thiemann, Martin Gasbichler, and Michael Sperber. Functional logic overloading. In *POPL '02*

[P02], 233–244.
http://doi.acm.org/10.1145/503272.503294 549021.

[P02I] Umut A. Acar, Guy E. Blelloch, and Robert Harper. Adaptive functional programming. In *POPL '02* [P02], 247–259.
http://doi.acm.org/10.1145/503272.503296 549021.

[P02J] Martin Hofmann. The strength of non-size increasing computation. In *POPL '02* [P02], 260–269.
http://doi.acm.org/10.1145/503272.503297 549021.

[P02K] Cédric Fournet and Andrew D. Gordon. Stack inspection: Theory and variants. In *POPL '02* [P02], 307–318.
http://doi.acm.org/10.1145/503272.503301 549021.

[P02L] François Pottier and Vincent Simonet. Information flow inference for ml. In *POPL '02* [P02], 319–330.
http://doi.acm.org/10.1145/503272.503302 549021.

[P02M] Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. In *POPL '02* [P02], 331–342.
http://doi.acm.org/10.1145/503272.503303 549021.

[P03] **POPL '03**: *Proc. 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2003. Association for Computing Machinery. ISBN 1-58113-628-5. 549031.

[P03A] Jérôme Siméon and Philip Wadler. The essence of xml. In *POPL '03* [P03], 1–13.
http://doi.acm.org/10.1145/604131.604132 549031.

[P03B] Umut A. Acar, Guy E. Blelloch, and Robert Harper. Selective memoization. In *POPL '03* [P03], 14–25.
http://doi.acm.org/10.1145/604131.604133 549031.

[P03C] Walid Taha and Michael Florentin Nielsen. Environment classifiers. In *POPL '03* [P03], 26–37.
http://doi.acm.org/10.1145/604131.604134 549031.

[P03D] Alan Schmitt and Jean-Bernard Stefani. The m-calculus: A higher-order distributed process calculus. In *POPL '03* [P03], 50–61. http://doi.acm.org/10.1145/604131.604136 549031.

[P03E] Hayo Thielecke. From control effects to typed continuation passing. In *POPL '03* [P03], 139–149.
http://doi.acm.org/10.1145/604131.604144 549031.

[P03F] Gang Chen. Coercive subtyping for the calculus of constructions. In *POPL '03* [P03], 150–159.
http://doi.acm.org/10.1145/604131.604145 549031.

[P03G] Leaf Petersen, Robert Harper, Karl Crary, and Frank Pfenning. A type theory for memory allocation and data layout. In *POPL '03* [P03], 172–184.
http://doi.acm.org/10.1145/604131.604147 549031.

[P03H] Martin Hofmann and Steffen Jost. Static prediction of heap space usage for first-order functional programs. In *POPL '03* [P03], 185–197. http://doi.acm.org/10.1145/604131.604148 549031.

[P03I] Karl Crary. Toward a foundational typed assembly language. In *POPL '03* [P03], 198–212.
http://doi.acm.org/10.1145/604131.604149 549031.

[P03J] Chandrasekhar Boyapati, Barbara Liskov, and Liuba Shrira. Ownership types for object encapsulation. In *POPL '03* [P03], 213–223. http://doi.acm.org/10.1145/604131.604156 549031.

[P03K] Hongwei Xi, Chiyan Chen, and Gang Chen. Guarded recursive datatype constructors. In *POPL '03* [P03], 224–235.
http://doi.acm.org/10.1145/604131.604150 549031.

[P03L] Derek Dreyer, Karl Crary, and Robert Harper. A type system for higher-order modules. In *POPL '03* [P03], 236–249.
http://doi.acm.org/10.1145/604131.604151 549031.

[P03M] Gilles Barthe, Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Pure patterns type systems. In *POPL '03* [P03], 250–261.
http://doi.acm.org/10.1145/604131.604152 549031.

[P04] **POPL '04**: *Proc. 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 39, New York, 2004. Association for Computing Machinery. ISBN 1-58113-729-X. 549041.

[P04A] Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In *POPL '04* [P04], 14–25.
http://doi.acm.org/10.1145/964001.964003 549041.

[P04B] Dachuan Yu, Andrew Kennedy, and Don Syme. Formalization of generics for the .net common language runtime. In *POPL '04* [P04], 39–51.
http://doi.acm.org/10.1145/964001.964005 549041.

[P04C] Jerome Vouillon and Paul-André Melliès. Semantic types: A fresh look at the ideal model for types. In *POPL '04* [P04], 52–63.
http://doi.acm.org/10.1145/964001.964006 549041.

[P04D] Vincent Balat, Roberto Di Cosmo, and Marcelo Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In *POPL '04* [P04], 64–76.
http://doi.acm.org/10.1145/964001.964007 549041.

[P04E] Marcelo Fiore. Isomorphisms of generic recursive polynomial types. In *POPL '04* [P04], 77–88.
http://doi.acm.org/10.1145/964001.964008 549041.

[P04F] François Pottier and Nadji Gauthier. Polymorphic typed defunctionalization. In *POPL '04* [P04], 89–98.
http://doi.acm.org/10.1145/964001.964009 549041.

[P04G] Denis Caromel, Ludovic Henrio, and Bernard Paul Serpette. Asynchronous and deterministic objects. In *POPL '04* [P04], 123–134. http://doi.acm.org/10.1145/964001.964012 549041.

[P04H] Silvano Dal Zilio, Denis Lugiez, and Charles Meyssonnier. A logic you can count on. In *POPL '04* [P04], 135–146.
http://doi.acm.org/10.1145/964001.964013 549041.

[P04I] Nobuko Yoshida. Channel dependent types for higher-order mobile processes. In *POPL '04* [P04], 147–160.
http://doi.acm.org/10.1145/964001.964014 549041.

[P04J] Eijiro Sumii and Benjamin C. Pierce. A bisimulation for dynamic sealing. In *POPL '04* [P04], 161–172.
http://doi.acm.org/10.1145/964001.964015 549041.

[P04K] Roberto Giacobazzi and Isabella Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *POPL '04* [P04], 186–197.
http://doi.acm.org/10.1145/964001.964017 549041.

[P04L] Lars Birkedal, Noah Torp-Smith, and John C. Reynolds. Local reasoning about a copying garbage collector. In *POPL '04* [P04], 220–231. http://doi.acm.org/10.1145/964001.964020 549041.

[P04M] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In *POPL '04* [P04], 232–244. http://doi.acm.org/10.1145/964001.964021 549041.

[P04N] Shaz Qadeer, Sriram K. Rajamani, and Jakob Rehof. Summarizing procedures in concurrent programs. In *POPL '04* [P04], 245–255.
http://doi.acm.org/10.1145/964001.964022 549041.

[P04O] Cormac Flanagan and Stephen N Freund. Atomizer: A dynamic atomicity checker for multithreaded programs. In *POPL '04* [P04], 256–267.
http://doi.acm.org/10.1145/964001.964023 549041.

[P04P] Peter W. O'Hearn, Hongseok Yang, and John C. Reynolds. Separation and information hiding. In *POPL '04* [P04], 268–280.
http://doi.acm.org/10.1145/964001.964024 549041.

[P04Q] Joshua Dunfield and Frank Pfenning. Tridirectional typechecking. In *POPL '04* [P04], 281–292.
http://doi.acm.org/10.1145/964001.964025 549041.

[P04R] Derek Dreyer. A type system for well-founded recursion. In *POPL '04* [P04], 293–305.
http://doi.acm.org/10.1145/964001.964026 549041.

[P04S] Davide Ancona and Elena Zucca. Principal typings for java-like languages. In *POPL '04* [P04], 306–317.
http://doi.acm.org/10.1145/964001.964027 549041.

[P05] **POPL '05**: *Proc. 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2005. Association for Computing Machinery. ISBN 1-58113-830-X. 549051.

[P05A] Manuel M. T. Chakravarty, Gabriele Keller, Simon Peyton Jones, and Simon Marlow. Associated types with class. In *POPL '05*

[P05], 1–13.
http://doi.acm.org/10.1145/1040305.1040306 549051.

[P05B] Richard Cobbe and Matthias Felleisen. Environmental acquisition revisited. In *POPL '05* [P05], 14–25.
http://doi.acm.org/10.1145/1040305.1040307 549051.

[P05C] Davide Ancona, Ferruccio Damiani, Sophia Drossopoulou, and Elena Zucca. Polymorphic bytecode: Compositional compilation for java-like languages. In *POPL '05* [P05], 26–37.
http://doi.acm.org/10.1145/1040305.1040308 549051.

[P05D] Juan Chen and David Tarditi. A simple typed intermediate language for object-oriented languages. In *POPL '05* [P05], 38–49.
http://doi.acm.org/10.1145/1040305.1040309 549051.

[P05E] Haruo Hosoya, Alain Frisch, and Giuseppe Castagna. Parametric polymorphism for xml. In *POPL '05* [P05], 50–62.
http://doi.acm.org/10.1145/1040305.1040310 549051.

[P05F] Eijiro Sumii and Benjamin C. Pierce. A bisimulation for type abstraction and recursion. In *POPL '05* [P05], 63–74.
http://doi.acm.org/10.1145/1040305.1040311 549051.

[P05G] Healfdene Goguen. A syntactic approach to eta equality in type theory. In *POPL '05* [P05], 75–84.
http://doi.acm.org/10.1145/1040305.1040312 549051.

[P05H] Simon J. Gay and Rajagopal Nagarajan. Communicating quantum processes. In *POPL '05* [P05], 145–157.
http://doi.acm.org/10.1145/1040305.1040318 549051.

[P05I] Peng Li and Steve Zdancewic. Downgrading policies and relaxed noninterference. In *POPL '05* [P05], 158–170.
http://doi.acm.org/10.1145/1040305.1040319 549051.

[P05J] Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic language based upon sampling functions. In *POPL '05* [P05], 171–182.
http://doi.acm.org/10.1145/1040305.1040320 549051.

[P05K] Gareth Stoyle, Michael Hicks, Gavin Bierman, Peter Sewell, and Iulian Neamtiu. Mutatis mutandis: Safe and predictable dynamic software updating. In *POPL '05* [P05], 183–194.
http://doi.acm.org/10.1145/1040305.1040321 549051.

[P05L] John Field and Carlos A. Varela. Transactors: A programming model for maintaining globally consistent distributed state in unreliable environments. In *POPL '05* [P05], 195–208.
http://doi.acm.org/10.1145/1040305.1040322 549051.

[P05M] Roberto Bruni, Hernán Melgratti, and Ugo Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL '05* [P05], 209–220.
http://doi.acm.org/10.1145/1040305.1040323 549051.

[P05N] Matthias Neubauer and Peter Thiemann. From sequential programs to multi-tier applications by program transformation. In *POPL '05* [P05], 221–232.
http://doi.acm.org/10.1145/1040305.1040324 549051.

[P05O] Matthew Parkinson and Gavin Bierman. Separation logic and abstraction. In *POPL '05* [P05], 247–258.
http://doi.acm.org/10.1145/1040305.1040326 549051.

[P05P] Richard Bornat, Cristiano Calcagno, Peter O'Hearn, and Matthew Parkinson. Permission accounting in separation logic. In *POPL '05* [P05], 259–270.
http://doi.acm.org/10.1145/1040305.1040327 549051.

[P05Q] Cristiano Calcagno, Philippa Gardner, and Uri Zarfaty. Context logic and tree update. In *POPL '05* [P05], 271–282.
http://doi.acm.org/10.1145/1040305.1040328 549051.

[P05R] John Tang Boyland and William Retert. Connecting effects and uniqueness with adoption. In *POPL '05* [P05], 283–295.
http://doi.acm.org/10.1145/1040305.1040329 549051.

[P05S] Noam Rinetzky, Jörg Bauer, Thomas Reps, Mooly Sagiv, and Reinhard Wilhelm. A semantics for procedure local heaps and its abstractions. In *POPL '05* [P05], 296–309.
http://doi.acm.org/10.1145/1040305.1040330 549051.

[P05T] Brian Hackett and Radu Rugina. Region-based shape analysis with tracked locations. In *POPL '05* [P05], 310–323.
http://doi.acm.org/10.1145/1040305.1040331 549051.

[P05U] Yichen Xie and Alex Aiken. Scalable error detection using boolean satisfiability. In *POPL '05* [P05], 351–363.
http://doi.acm.org/10.1145/1040305.1040334 549051.

[P06] **POPL '06**: *Proc. 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2006. Association for Computing Machinery. ISBN 1-59593-027-2. 549061.

[P06A] Kathleen Fisher, Yitzhak Mandelbaum, and David Walker. The next 700 data description languages. In *POPL '06* [P06], 2–15.
http://doi.acm.org/10.1145/1111037.1111039 549061.

[P06B] Xavier Leroy. Formal certification of a compiler back-end or: Programming a compiler with a proof assistant. In *POPL '06* [P06], 42–54.
http://doi.acm.org/10.1145/1111037.1111042 549061.

[P06C] Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Engineering with logic: Hol specification and symbolic-evaluation testing for tcp implementations. In *POPL '06* [P06], 55–66.
http://doi.acm.org/10.1145/1111037.1111043 549061.

[P06D] Mads Dam. Decidability and proof systems for language-based noninterference relations. In *POPL '06* [P06], 67–78.
http://doi.acm.org/10.1145/1111037.1111044 549061.

[P06E] Sebastian Hunt and David Sands. On flow-sensitive security types. In *POPL '06* [P06], 79–90.
http://doi.acm.org/10.1145/1111037.1111045 549061.

[P06F] Torben Amtoft, Sruthi Bandhakavi, and Anindya Banerjee. A logic for information flow in object-oriented programs. In *POPL '06* [P06], 91–102.
http://doi.acm.org/10.1145/1111037.1111046 549061.

[P06G] Jérôme Vouillon. Polymorphic regular tree types and patterns. In *POPL '06* [P06], 103–114.
http://doi.acm.org/10.1145/1111037.1111047 549061.

[P06H] Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL '06* [P06], 141–152.
http://doi.acm.org/10.1145/1111037.1111050 549061.

[P06I] Norman Danner and James S. Royer. Adventures in time and space. In *POPL '06* [P06], 168–179.
http://doi.acm.org/10.1145/1111037.1111053 549061.

[P06J] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. N-synchronous kahn networks: A relaxed model of synchrony for real-time systems. In *POPL '06* [P06], 180–193.
http://doi.acm.org/10.1145/1111037.1111054 549061.

[P06K] Nils Anders Danielsson, John Hughes, Patrik Jansson, and Jeremy Gibbons. Fast and loose reasoning is morally correct. In *POPL '06* [P06], 206–217.
http://doi.acm.org/10.1145/1111037.1111056 549061.

[P06L] Philippe Meunier, Robert Bruce Findler, and Matthias Felleisen. Modular set-based analysis from contracts. In *POPL '06* [P06], 218–231. http://doi.acm.org/10.1145/1111037.1111057 549061.

[P06M] François Pottier and Yann Régis-Gianas. Stratified type inference for generalized algebraic data types. In *POPL '06* [P06], 232–244.
http://doi.acm.org/10.1145/1111037.1111058 549061.

[P06N] Cormac Flanagan. Hybrid type checking. In *POPL '06* [P06], 245–256. http://doi.acm.org/10.1145/1111037.1111059 549061.

[P06O] Ik-Soon Kim, Kwangkeun Yi, and Cristiano Calcagno. A polymorphic modal type system for lisp-like multi-staged languages. In *POPL '06* [P06], 257–268.
http://doi.acm.org/10.1145/1111037.1111060 549061.

[P06P] Erik Ernst, Klaus Ostermann, and William R. Cook. A virtual class calculus. In *POPL '06* [P06], 270–282.
http://doi.acm.org/10.1145/1111037.1111062 549061.

[P06Q] Jed Liu, Aaron Kimball, and Andrew C. Myers. Interruptible iterators. In *POPL '06* [P06], 283–294.
http://doi.acm.org/10.1145/1111037.1111063 549061.

[P06R] Hayo Thielecke. Frame rules from answer types for code pointers. In *POPL '06* [P06], 309–319. http://doi.acm.org/10.1145/1111037.1111065 549061.

[P06S] Zhaozhong Ni and Zhong Shao. Certified assembly programming with embedded code pointers. In *POPL '06* [P06], 320–333. http://doi.acm.org/10.1145/1111037.1111066 549061.

[P06T] Bill McCloskey, Feng Zhou, David Gay, and Eric Brewer. Autolocker: Synchronization inference for atomic sections. In *POPL '06* [P06], 346–358. http://doi.acm.org/10.1145/1111037.1111068 549061.

[P06U] Yi Lu and John Potter. Protecting representation with effect encapsulation. In *POPL '06* [P06], 359–371. http://doi.acm.org/10.1145/1111037.1111069 549061.

[P06V] Daniel S. Dantas and David Walker. Harmless advice. In *POPL '06* [P06], 383–396. http://doi.acm.org/10.1145/1111037.1111071 549061.

[P06W] Vijay S. Menon, Neal Glew, Brian R. Murphy, Andrew McCreight, Tatiana Shpeisman, Ali-Reza Adl-Tabatabai, and Leaf Petersen. A verifiable ssa program representation for aggressive compiler optimization. In *POPL '06* [P06], 397–408. http://doi.acm.org/10.1145/1111037.1111072 549061.

[P06X] Reuben Olinsky, Christian Lindig, and Norman Ramsey. Staged allocation: A compositional technique for specifying and implementing procedure calling conventions. In *POPL '06* [P06], 409–421. http://doi.acm.org/10.1145/1111037.1111073 549061.

[P07] **POPL '07**: *Proc. 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2007. Association for Computing Machinery. ISBN 1-59593-575-4. 549071.

[P07A] Jacob Matthews and Robert Bruce Findler. Operational semantics for multi-language programs. In *POPL '07* [P07], 3–10. http://doi.acm.org/10.1145/1190216.1190220 549071.

[P07B] Juan Chen. A typed intermediate language for compiling multiple inheritance. In *POPL '07* [P07], 25–30. http://doi.acm.org/10.1145/1190216.1190222 549071.

[P07C] Derek Dreyer, Robert Harper, Manuel M. T. Chakravarty, and Gabriele Keller. Modular type classes. In *POPL '07* [P07], 63–70. http://doi.acm.org/10.1145/1190216.1190229 549071.

[P07D] Andrew M. Pitts and Mark R. Shinwell. Generative unbinding of names. In *POPL '07* [P07], 85–95. http://doi.acm.org/10.1145/1190216.1190232 549071.

[P07E] Harvey Tuch, Gerwin Klein, and Michael Norrish. Types, bytes, and separation logic. In *POPL '07* [P07], 97–108. http://doi.acm.org/10.1145/1190216.1190234 549071.

[P07F] Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A very modal model of a modern, major, general type system. In *POPL '07* [P07], 109–122. http://doi.acm.org/10.1145/1190216.1190235 549071.

[P07G] Cristiano Calcagno, Philippa Gardner, and Uri Zarfaty. Context logic as modal logic: Completeness and parametric inexpressivity. In *POPL '07* [P07], 123–134. http://doi.acm.org/10.1145/1190216.1190236 549071.

[P07H] Atsushi Ohori and Isao Sasano. Lightweight fusion by fixed point promotion. In *POPL '07* [P07], 143–154. http://doi.acm.org/10.1145/1190216.1190241 549071.

[P07I] Kristian Støvring and Soren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *POPL '07* [P07], 161–172. http://doi.acm.org/10.1145/1190216.1190244 549071.

[P07J] Daniel K. Lee, Karl Crary, and Robert Harper. Towards a mechanized metatheory of standard ml. In *POPL '07* [P07], 173–184. http://doi.acm.org/10.1145/1190216.1190245 549071.

[P07K] Matthew Might. Logic-flow analysis of higher-order programs. In *POPL '07* [P07], 185–198. http://doi.acm.org/10.1145/1190216.1190247 549071.

[P07L] Ben Wiedermann and William R. Cook. Extracting queries by static analysis of transparent persistence. In *POPL '07* [P07], 199–210. http://doi.acm.org/10.1145/1190216.1190248 549071.

[P07M] Dachuan Yu, Ajay Chander, Nayeem Islam, and Igor Serikov. Javascript instrumentation for browser security. In *POPL '07* [P07], 237–249. http://doi.acm.org/10.1145/1190216.1190252 549071.

[P07N] Michele Bugliesi and Marco Giunti. Secure implementations of typed channel abstractions. In *POPL '07* [P07], 251–262. http://doi.acm.org/10.1145/1190216.1190253 549071.

[P07O] Matthew Parkinson, Richard Bornat, and Peter O'Hearn. Modular verification of a non-blocking stack. In *POPL '07* [P07], 297–302. http://doi.acm.org/10.1145/1190216.1190261 549071.

[P07P] John Reppy and Yingqi Xiao. Specialization of cml message-passing primitives. In *POPL '07* [P07], 315–326. http://doi.acm.org/10.1145/1190216.1190264 549071.

[P07Q] Mayur Naik and Alex Aiken. Conditional must not aliasing for static race detection. In *POPL '07* [P07], 327–338. http://doi.acm.org/10.1145/1190216.1190265 549071.

[P07R] Dan R. Ghica. Geometry of synthesis: A structured approach to vlsi design. In *POPL '07* [P07], 363–375. http://doi.acm.org/10.1145/1190216.1190269 549071.

[P08] **POPL '08**: *Proc. 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 43, New York, 2008. Association for Computing Machinery. ISBN 978-1-59593-689-9. 549081.

[P08A] Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *POPL '08* [P08], 3–15. http://doi.acm.org/10.1145/1328438.1328443 549081.

[P08B] Jean-Baptiste Tristan and Xavier Leroy. Formal verification of translation validators: A case study on instruction scheduling optimizations. In *POPL '08* [P08], 17–27. http://doi.acm.org/10.1145/1328438.1328444 549081.

[P08C] Iulian Neamtiu, Michael Hicks, Jeffrey S. Foster, and Polyvios Pratikakis. Contextual effects for version-consistent dynamic software updating and safe concurrent programming. In *POPL '08* [P08], 37–49. http://doi.acm.org/10.1145/1328438.1328447 549081.

[P08D] Katherine F. Moore and Dan Grossman. High-level small-step operational semantics for transactions. In *POPL '08* [P08], 51–62. http://doi.acm.org/10.1145/1328438.1328448 549081.

[P08E] Martín Abadi, Andrew Birrell, Tim Harris, and Michael Isard. Semantics of transactional memory and automatic mutual exclusion. In *POPL '08* [P08], 63–74. http://doi.acm.org/10.1145/1328438.1328449 549081.

[P08F] Matthew J. Parkinson and Gavin M. Bierman. Separation logic, abstraction and inheritance. In *POPL '08* [P08], 75–86. http://doi.acm.org/10.1145/1328438.1328451 549081.

[P08G] Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. Enhancing modular oo verification with separation logic. In *POPL '08* [P08], 87–99. http://doi.acm.org/10.1145/1328438.1328452 549081.

[P08H] James Brotherston, Richard Bornat, and Cristiano Calcagno. Cyclic proofs of program termination in separation logic. In *POPL '08* [P08], 101–112. http://doi.acm.org/10.1145/1328438.1328453 549081.

[P08I] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of pspace. In *POPL '08* [P08], 121–131. http://doi.acm.org/10.1145/1328438.1328456 549081.

[P08J] Nils Anders Danielsson. Lightweight semiformal time complexity analysis for purely functional data structures. In *POPL '08* [P08], 133–144. http://doi.acm.org/10.1145/1328438.1328457 549081.

[P08K] Shuvendu Lahiri and Shaz Qadeer. Back to the future: Revisiting precise program verification using smt solvers. In *POPL '08* [P08], 171–182. http://doi.acm.org/10.1145/1328438.1328461 549081.

[P08L] Christopher Unkel and Monica S. Lam. Automatic inference of stationary fields: A generalization of java's final fields. In *POPL*

'08 [P08], 183–195.
http://doi.acm.org/10.1145/1328438.1328463 549081.

[P08M] Marius Nita, Dan Grossman, and Craig Chambers. A theory of platform-dependent low-level software. In *POPL '08* [P08], 209–220. http://doi.acm.org/10.1145/1328438.1328465 549081.

[P08N] Sumit Gulwani, Bill McCloskey, and Ashish Tiwari. Lifting abstract interpreters to quantified logical domains. In *POPL '08* [P08], 235–246.
http://doi.acm.org/10.1145/1328438.1328468 549081.

[P08O] Bor-Yuh Evan Chang and Xavier Rival. Relational inductive shape analysis. In *POPL '08* [P08], 247–260.
http://doi.acm.org/10.1145/1328438.1328469 549081.

[P08P] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. In *POPL '08* [P08], 261–272.
http://doi.acm.org/10.1145/1328438.1328471 549081.

[P08Q] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL '08* [P08], 273–284.
http://doi.acm.org/10.1145/1328438.1328472 549081.

[P08R] Umut A. Acar, Amal Ahmed, and Matthias Blume. Imperative self-adjusting computation. In *POPL '08* [P08], 309–322.
http://doi.acm.org/10.1145/1328438.1328476 549081.

[P08S] Cédric Fournet and Tamara Rezk. Cryptographically sound implementations for typed information-flow security. In *POPL '08* [P08], 323–335.
http://doi.acm.org/10.1145/1328438.1328478 549081.

[P08T] Peeter Laud. On the computational soundness of cryptographically masked flows. In *POPL '08* [P08], 337–348.
http://doi.acm.org/10.1145/1328438.1328479 549081.

[P08U] Noam Zeilberger. Focusing and higher-order abstract syntax. In *POPL '08* [P08], 359–369.
http://doi.acm.org/10.1145/1328438.1328482 549081.

[P08V] Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *POPL '08* [P08], 371–382.
http://doi.acm.org/10.1145/1328438.1328483 549081.

[P08W] Hugo Herbelin and Silvia Ghilezan. An approach to call-by-name delimited continuations. In *POPL '08* [P08], 383–394.
http://doi.acm.org/10.1145/1328438.1328484 549081.

[P08X] Sam Tobin-Hochstadt and Matthias Felleisen. The design and implementation of typed scheme. In *POPL '08* [P08], 395–406.
http://doi.acm.org/10.1145/1328438.1328486 549081.

[P08Y] Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang: Resourceful lenses for string data. In *POPL '08* [P08], 407–419.
http://doi.acm.org/10.1145/1328438.1328487 549081.

[P09] **POPL '09**: *Proc. 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 44, New York, 2009. Association for Computing Machinery. ISBN 978-1-60558-379-2. 549091.

[P09A] Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran. A calculus of atomic actions. In *POPL '09* [P09], 2–15.
http://doi.acm.org/10.1145/1480881.1480885 549091.

[P09B] Alexey Gotsman, Byron Cook, Matthew Parkinson, and Viktor Vafeiadis. Proving that non-blocking algorithms don't block. In *POPL '09* [P09], 16–28.
http://doi.acm.org/10.1145/1480881.1480886 549091.

[P09C] Martin Abadi and Gordon Plotkin. A model of cooperative threads. In *POPL '09* [P09], 29–40.
http://doi.acm.org/10.1145/1480881.1480887 549091.

[P09D] Dana N. Xu, Simon Peyton Jones, and Koen Claessen. Static contract checking for haskell. In *POPL '09* [P09], 41–52.
http://doi.acm.org/10.1145/1480881.1480889 549091.

[P09E] Xin Qi and Andrew C. Myers. Masked types for sound object initialization. In *POPL '09* [P09], 53–65.
http://doi.acm.org/10.1145/1480881.1480890 549091.

[P09F] Daan Leijen. Flexible types: Robust type inference for first-class polymorphism. In *POPL '09* [P09], 66–77.
http://doi.acm.org/10.1145/1480881.1480891 549091.

[P09G] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL '09* [P09], 90–101.
http://doi.acm.org/10.1145/1480881.1480894 549091.

[P09H] Ronald Garcia, Andrew Lumsdaine, and Amr Sabry. Lazy evaluation and delimited control. In *POPL '09* [P09], 153–164.
http://doi.acm.org/10.1145/1480881.1480903 549091.

[P09I] Ruy Ley-Wild, Umut A. Acar, and Matthew Fluet. A cost semantics for self-adjusting computation. In *POPL '09* [P09], 186–199.
http://doi.acm.org/10.1145/1480881.1480907 549091.

[P09J] Akihiko Tozawa, Michiaki Tatsubori, Tamiya Onodera, and Yasuhiko Minamide. Copy-on-write in the php language. In *POPL '09* [P09], 200–212.
http://doi.acm.org/10.1145/1480881.1480908 549091.

[P09K] Peter A. Jonsson and Johan Nordlander. Positive supercompilation for a higher order call-by-value language. In *POPL '09* [P09], 277–288. http://doi.acm.org/10.1145/1480881.1480916 549091.

[P09L] Cristiano Calcagno, Dino Distefano, Peter O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. In *POPL '09* [P09], 289–300.
http://doi.acm.org/10.1145/1480881.1480917 549091.

[P09M] Xinyu Feng. Local rely-guarantee reasoning. In *POPL '09* [P09], 315–327. http://doi.acm.org/10.1145/1480881.1480922 549091.

[P09N] James Brotherston and Cristiano Calcagno. Classical bi: A logic for reasoning about dualising resources. In *POPL '09* [P09], 328–339. http://doi.acm.org/10.1145/1480881.1480923 549091.

[P09O] Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *POPL '09* [P09], 340–353. http://doi.acm.org/10.1145/1480881.1480925 549091.

[P09P] Benoît Montagu and Didier Rémy. Modeling abstract types in modules with open existential types. In *POPL '09* [P09], 354–365. http://doi.acm.org/10.1145/1480881.1480926 549091.

[P09Q] Neelakantan R. Krishnaswami. Focusing on pattern matching. In *POPL '09* [P09], 366–378.
http://doi.acm.org/10.1145/1480881.1480927 549091.

[P09R] Gérard Boudol and Gustavo Petri. Relaxed memory models: An operational approach. In *POPL '09* [P09], 392–403.
http://doi.acm.org/10.1145/1480881.1480930 549091.

[P09S] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL '09* [P09], 416–428. http://doi.acm.org/10.1145/1480881.1480933 549091.

[P10] **POPL '10**: *Proc. 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 45, New York, 2010. Association for Computing Machinery. ISBN 978-1-60558-479-9. 549101.

[P10A] Eric Koskinen, Matthew Parkinson, and Maurice Herlihy. Coarse-grained transactions. In *POPL '10* [P10], 19–30.
http://doi.acm.org/10.1145/1706299.1706304 549101.

[P10B] Patrice Godefroid, Aditya V. Nori, Sriram K. Rajamani, and Sai Deep Tetali. Compositional may-must program analysis: Unleashing the power of alternation. In *POPL '10* [P10], 43–56.
http://doi.acm.org/10.1145/1706299.1706307 549101.

[P10C] Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. Continuity analysis of programs. In *POPL '10* [P10], 57–70.
http://doi.acm.org/10.1145/1706299.1706308 549101.

[P10D] Jean-Baptiste Tristan and Xavier Leroy. A simple, verified validator for software pipelining. In *POPL '10* [P10], 83–92.
http://doi.acm.org/10.1145/1706299.1706311 549101.

[P10E] Adam Chlipala. A verified compiler for an impure functional language. In *POPL '10* [P10], 93–106.
http://doi.acm.org/10.1145/1706299.1706312 549101.

[P10F] Magnus O. Myreen. Verified just-in-time compiler on x86. In *POPL '10* [P10], 107–118.
http://doi.acm.org/10.1145/1706299.1706313 549101.

[P10G] Tachio Terauchi. Dependent types from counterexamples. In *POPL '10* [P10], 119–130. http://doi.acm.org/10.1145/1706299.1706315 549101.

[P10H] Patrick Maxim Rondon, Ming Kawaguchi, and Ranjit Jhala. Low-level liquid types. In *POPL '10* [P10], 131–144. http://doi.acm.org/10.1145/1706299.1706316 549101.

[P10I] Andrew M. Pitts. Nominal system t. In *POPL '10* [P10], 159–170. http://doi.acm.org/10.1145/1706299.1706321 549101.

[P10J] Derek Dreyer, Georg Neis, Andreas Rossberg, and Lars Birkedal. A relational modal logic for higher-order stateful adts. In *POPL '10* [P10], 185–198. http://doi.acm.org/10.1145/1706299.1706323 549101.

[P10K] Philippe Suter, Mirco Dotta, and Viktor Kuncak. Decision procedures for algebraic data types with abstractions. In *POPL '10* [P10], 199–210. http://doi.acm.org/10.1145/1706299.1706325 549101.

[P10L] Stephen Magill, Ming-Hsien Tsai, Peter Lee, and Yih-Kuen Tsay. Automatic numeric abstractions for heap-manipulating programs. In *POPL '10* [P10], 211–222. http://doi.acm.org/10.1145/1706299.1706326 549101.

[P10M] Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. Static determination of quantitative resource usage for higher-order programs. In *POPL '10* [P10], 223–236. http://doi.acm.org/10.1145/1706299.1706327 549101.

[P10N] Aleksandar Nanevski, Viktor Vafeiadis, and Josh Berdine. Structuring the verification of heap-manipulating programs. In *POPL '10* [P10], 261–274. http://doi.acm.org/10.1145/1706299.1706331 549101.

[P10O] Limin Jia, Jianzhou Zhao, Vilhelm Sjöberg, and Stephanie Weirich. Dependent types and program equivalence. In *POPL '10* [P10], 275–286. http://doi.acm.org/10.1145/1706299.1706333 549101.

[P10P] DeLesley S. Hutchins. Pure subtype systems. In *POPL '10* [P10], 287–298. http://doi.acm.org/10.1145/1706299.1706334 549101.

[P10Q] Simon J. Gay, Vasco T. Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Modular session types for distributed object-oriented programming. In *POPL '10* [P10], 299–312. http://doi.acm.org/10.1145/1706299.1706335 549101.

[P10R] Michael Greenberg, Benjamin C. Pierce, and Stephanie Weirich. Contracts made manifest. In *POPL '10* [P10], 353–364. http://doi.acm.org/10.1145/1706299.1706341 549101.

[P10S] Jeremy G. Siek and Philip Wadler. Threesomes, with and without blame. In *POPL '10* [P10], 365–376. http://doi.acm.org/10.1145/1706299.1706342 549101.

[P10T] Tobias Wrigstad, Francesco Zappa Nardelli, Sylvain Lebresne, Johan Östlund, and Jan Vitek. Integrating typed and untyped code in a scripting language. In *POPL '10* [P10], 377–388. http://doi.acm.org/10.1145/1706299.1706343 549101.

[P10U] João Dias and Norman Ramsey. Automatically generating instruction selectors using declarative machine descriptions. In *POPL '10* [P10], 403–416. http://doi.acm.org/10.1145/1706299.1706346 549101.

[P10V] Trevor Jim, Yitzhak Mandelbaum, and David Walker. Semantics and algorithms for data-dependent grammars. In *POPL '10* [P10], 417–430. http://doi.acm.org/10.1145/1706299.1706347 549101.

[P10W] Niklas Broberg and David Sands. Paralocks: Role-based information flow control and beyond. In *POPL '10* [P10], 431–444. http://doi.acm.org/10.1145/1706299.1706349 549101.

[P10X] Jean-Phillipe Martin, Michael Hicks, Manuel Costa, Periklis Akritidis, and Miguel Castro. Dynamically checking ownership policies in concurrent c/c++ programs. In *POPL '10* [P10], 457–470. http://doi.acm.org/10.1145/1706299.1706351 549101.

[P10Y] Andrzej Filinski. Monads in action. In *POPL '10* [P10], 483–494. http://doi.acm.org/10.1145/1706299.1706354 549101.

[P10Z] Naoki Kobayashi, Naoshi Tabuchi, and Hiroshi Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *POPL '10* [P10], 495–508. http://doi.acm.org/10.1145/1706299.1706355 549101.

[P11] **POPL '11**: *Proc. 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 46, New York, 2011. Association for Computing Machinery. ISBN 978-1-4503-0490-0. 549111.

[P11A] Tahina Ramananandro, Gabriel Dos Reis, and Xavier Leroy. Formal verification of object layout for c++ multiple inheritance. In *POPL '11* [P11], 67–80. http://doi.acm.org/10.1145/1926385.1926395 549111.

[P11B] Wontae Choi, Baris Aktemur, Kwangkeun Yi, and Makoto Tatsuta. Static analysis of multi-staged programs via unstaging translation. In *POPL '11* [P11], 81–92. http://doi.acm.org/10.1145/1926385.1926397 549111.

[P11C] Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg, and Hongseok Yang. Step-indexed kripke models over recursive worlds. In *POPL '11* [P11], 119–132. http://doi.acm.org/10.1145/1926385.1926401 549111.

[P11D] François Pottier. A typed store-passing translation for general references. In *POPL '11* [P11], 147–158. http://doi.acm.org/10.1145/1926385.1926403 549111.

[P11E] Xavier Rival and Bor-Yuh Evan Chang. Calling context abstraction with shapes. In *POPL '11* [P11], 173–186. http://doi.acm.org/10.1145/1926385.1926406 549111.

[P11F] Isil Dillig, Thomas Dillig, and Alex Aiken. Precise reasoning for programs using containers. In *POPL '11* [P11], 187–200. http://doi.acm.org/10.1145/1926385.1926407 549111.

[P11G] Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek, and Philip Wadler. Blame for all. In *POPL '11* [P11], 201–214. http://doi.acm.org/10.1145/1926385.1926409 549111.

[P11H] Christos Dimoulas, Robert Bruce Findler, Cormac Flanagan, and Matthias Felleisen. Correct blame for contracts: No more scapegoating. In *POPL '11* [P11], 215–226. http://doi.acm.org/10.1145/1926385.1926410 549111.

[P11I] Stephanie Weirich, Dimitrios Vytiniotis, Simon Peyton Jones, and Steve Zdancewic. Generative type abstraction and type-level computation. In *POPL '11* [P11], 227–240. http://doi.acm.org/10.1145/1926385.1926411 549111.

[P11J] Aaron Joseph Turon and Mitchell Wand. A separation logic for refining concurrent objects. In *POPL '11* [P11], 247–258. http://doi.acm.org/10.1145/1926385.1926415 549111.

[P11K] Mike Dodds, Suresh Jagannathan, and Matthew J. Parkinson. Modular reasoning for deterministic parallelism. In *POPL '11* [P11], 259–270. http://doi.acm.org/10.1145/1926385.1926416 549111.

[P11L] Bart Jacobs and Frank Piessens. Expressive modular fine-grained concurrency specification. In *POPL '11* [P11], 271–282. http://doi.acm.org/10.1145/1926385.1926417 549111.

[P11M] Nikos Tzevelekos. Fresh-register automata. In *POPL '11* [P11], 295–306. http://doi.acm.org/10.1145/1926385.1926420 549111.

[P11N] Dan R. Ghica and Alex Smith. Geometry of synthesis iii: Resource management through type inference. In *POPL '11* [P11], 345–356. http://doi.acm.org/10.1145/1926385.1926425 549111.

[P11O] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. In *POPL '11* [P11], 357–370. http://doi.acm.org/10.1145/1926385.1926427 549111.

[P11P] Martin Hofmann, Benjamin Pierce, and Daniel Wagner. Symmetric lenses. In *POPL '11* [P11], 371–384. http://doi.acm.org/10.1145/1926385.1926428 549111.

[P11Q] Fritz Henglein and Lasse Nielsen. Regular expression containment: Coinductive axiomatization and computational interpretation. In *POPL '11* [P11], 385–398. http://doi.acm.org/10.1145/1926385.1926429 549111.

[P11R] Byron Cook and Eric Koskinen. Making prophecies with decision predicates. In *POPL '11* [P11], 399–410. http://doi.acm.org/10.1145/1926385.1926431 549111.

[P11S] Michael Emmi, Shaz Qadeer, and Zvonimir Rakamarić. Delay-bounded scheduling. In *POPL '11* [P11], 411–422. http://doi.acm.org/10.1145/1926385.1926432 549111.

[P11T] Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In *POPL '11* [P11], 435–446. http://doi.acm.org/10.1145/1926385.1926435 549111.

[P11U] Jesse A. Tov and Riccardo Pucella. Practical affine types. In *POPL '11* [P11], 447–458. http://doi.acm.org/10.1145/1926385.1926436 549111.

[P11V] Jong-hoon (David) An, Avik Chaudhuri, Jeffrey S. Foster, and Michael Hicks. Dynamic inference of static types for ruby. In *POPL '11* [P11], 459–472. http://doi.acm.org/10.1145/1926385.1926437 549111.

[P11W] Yuan Feng, Runyao Duan, and Mingsheng Ying. Bisimulation for quantum processes. In *POPL '11* [P11], 523–534. http://doi.acm.org/10.1145/1926385.1926446 549111.

[P11X] Robert L. Bocchino, Jr., Stephen Heumann, Nima Honarmand, Sarita V. Adve, Vikram S. Adve, Adam Welc, and Tatiana Shpeisman. Safe nondeterminism in a deterministic-by-default parallel language. In *POPL '11* [P11], 535–548. http://doi.acm.org/10.1145/1926385.1926447 549111.

[P11Y] Norman Ramsey and João Dias. Resourceable, retargetable, modular instruction selection using a machine-independent, type-based tiling of low-level intermediate code. In *POPL '11* [P11], 575–586. http://doi.acm.org/10.1145/1926385.1926451 549111.

[P11Z] C.-H. Luke Ong and Steven J. Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *POPL '11* [P11], 587–598. http://doi.acm.org/10.1145/1926385.1926453 549111.

[P12] **POPL '12**: *Proc. 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2012. Association for Computing Machinery. ISBN 978-1-4503-1083-3. 549121.

[P12A] Stephan van Staden, Cristiano Calcagno, and Bertrand Meyer. Freefinement. In *POPL '12* [P12], 7–18. http://doi.acm.org/10.1145/2103656.2103661 549121.

[P12B] Philippa Anne Gardner, Sergio Maffeis, and Gareth David Smith. Towards a program logic for javascript. In *POPL '12* [P12], 31–44. http://doi.acm.org/10.1145/2103656.2103663 549121.

[P12C] Neelakantan R. Krishnaswami, Nick Benton, and Jan Hoffmann. Higher-order functional reactive programming in bounded space. In *POPL '12* [P12], 45–58. http://doi.acm.org/10.1145/2103656.2103665 549121.

[P12D] Chung-Kil Hur, Derek Dreyer, Georg Neis, and Viktor Vafeiadis. The marriage of bisimulations and kripke logical relations. In *POPL '12* [P12], 59–72. http://doi.acm.org/10.1145/2103656.2103666 549121.

[P12E] Roshan P. James and Amr Sabry. Information effects. In *POPL '12* [P12], 73–84. http://doi.acm.org/10.1145/2103656.2103667 549121.

[P12F] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. In *POPL '12* [P12], 85–96. http://doi.acm.org/10.1145/2103656.2103669 549121.

[P12G] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *POPL '12* [P12], 97–110. http://doi.acm.org/10.1145/2103656.2103670 549121.

[P12H] Phillip Heidegger, Annette Bieniusa, and Peter Thiemann. Access permission contracts for scripting languages. In *POPL '12* [P12], 111–122. http://doi.acm.org/10.1145/2103656.2103671 549121.

[P12I] Ali Sinan Köksal, Viktor Kuncak, and Philippe Suter. Constraints as control. In *POPL '12* [P12], 151–164. http://doi.acm.org/10.1145/2103656.2103675 549121.

[P12J] Thomas H. Austin and Cormac Flanagan. Multiple facets for dynamic information flow. In *POPL '12* [P12], 165–178. http://doi.acm.org/10.1145/2103656.2103677 549121.

[P12K] Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. In *POPL '12* [P12], 203–214. http://doi.acm.org/10.1145/2103656.2103681 549121.

[P12L] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programming languages. In *POPL '12* [P12], 217–230. http://doi.acm.org/10.1145/2103656.2103685 549121.

[P12M] Ravi Chugh, Patrick M. Rondon, and Ranjit Jhala. Nested refinements: A logic for duck typing. In *POPL '12* [P12], 231–244. http://doi.acm.org/10.1145/2103656.2103686 549121.

[P12N] Patrick Cousot and Radhia Cousot. An abstract interpretation framework for termination. In *POPL '12* [P12], 245–258. http://doi.acm.org/10.1145/2103656.2103687 549121.

[P12O] Krystof Hoder, Laura Kovacs, and Andrei Voronkov. Playing in the grey area of proofs. In *POPL '12* [P12], 259–272. http://doi.acm.org/10.1145/2103656.2103689 549121.

[P12P] Antonis Stampoulis and Zhong Shao. Static and user-extensible proof checking. In *POPL '12* [P12], 273–284. http://doi.acm.org/10.1145/2103656.2103690 549121.

[P12Q] Casey Klein, John Clements, Christos Dimoulas, Carl Eastlund, Matthias Felleisen, Matthew Flatt, Jay A. McCarthy, Jon Rafkind, Sam Tobin-Hochstadt, and Robert Bruce Findler. Run your research: On the effectiveness of lightweight mechanization. In *POPL '12* [P12], 285–296. http://doi.acm.org/10.1145/2103656.2103691 549121.

[P12R] Azadeh Farzan and Zachary Kincaid. Verification of parameterized concurrent programs by modular reasoning about data and control. In *POPL '12* [P12], 297–308. http://doi.acm.org/10.1145/2103656.2103693 549121.

[P12S] Matko Botincan, Mike Dodds, and Suresh Jagannathan. Resource-sensitive synchronization inference by abduction. In *POPL '12* [P12], 309–322. http://doi.acm.org/10.1145/2103656.2103694 549121.

[P12T] Uday S. Reddy and John C. Reynolds. Syntactic control of interference for separation logic. In *POPL '12* [P12], 323–336. http://doi.acm.org/10.1145/2103656.2103695 549121.

[P12U] Daniel R. Licata and Robert Harper. Canonicity for 2-dimensional type theory. In *POPL '12* [P12], 337–348. http://doi.acm.org/10.1145/2103656.2103697 549121.

[P12V] Ohad Kammar and Gordon D. Plotkin. Algebraic foundations for effect-dependent optimisations. In *POPL '12* [P12], 349–360. http://doi.acm.org/10.1145/2103656.2103698 549121.

[P12W] Julien Cretin and Didier Rémy. On the power of coercion abstraction. In *POPL '12* [P12], 361–372. http://doi.acm.org/10.1145/2103656.2103699 549121.

[P12X] Mikolaj Bojanczyk, Laurent Braud, Bartek Klin, and Slawomir Lasota. Towards nominal computation. In *POPL '12* [P12], 401–412. http://doi.acm.org/10.1145/2103656.2103704 549121.

[P12Y] Andrew Cave and Brigitte Pientka. Programming with binders and indexed data-types. In *POPL '12* [P12], 413–424. http://doi.acm.org/10.1145/2103656.2103705 549121.

[P12Z] Jianzhou Zhao, Santosh Nagarakatte, Milo M.K. Martin, and Steve Zdancewic. Formalizing the llvm intermediate representation for verified program transformations. In *POPL '12* [P12], 427–440. http://doi.acm.org/10.1145/2103656.2103709 549121.

[P12Γ] Hongjin Liang, Xinyu Feng, and Ming Fu. A rely-guarantee-based simulation for verifying concurrent program transformations. In *POPL '12* [P12], 455–468. http://doi.acm.org/10.1145/2103656.2103711 549121.

[P12Δ] Aseem Rastogi, Avik Chaudhuri, and Basil Hosmer. The ins and outs of gradual type inference. In *POPL '12* [P12], 481–494. http://doi.acm.org/10.1145/2103656.2103714 549121.

[P12Θ] Martin Hofmann, Benjamin Pierce, and Daniel Wagner. Edit lenses. In *POPL '12* [P12], 495–508. http://doi.acm.org/10.1145/2103656.2103715 549121.

[P12Λ] Tahina Ramananandro, Gabriel Dos Reis, and Xavier Leroy. A mechanized semantics for c++ object construction and destruction, with applications to resource management. In *POPL '12* [P12], 521–532. http://doi.acm.org/10.1145/2103656.2103718 549121.

[P12Ξ] Chucky Ellison and Grigore Roşu. An executable formal semantics of c with applications. In *POPL '12* [P12], 533–544. http://doi.acm.org/10.1145/2103656.2103719 549121.

[P12Π] Sooraj Bhat, Ashish Agarwal, Richard Vuduc, and Alexander Gray. A type theory for probability density functions. In *POPL '12* [P12], 545–556. http://doi.acm.org/10.1145/2103656.2103721 549121.

[P12Σ] Karl Naden, Robert Bocchino, Jonathan Aldrich, and Kevin Bierhoff. A type system for borrowing permissions. In *POPL '12* [P12], 557–570. http://doi.acm.org/10.1145/2103656.2103722 549121.

[P12Υ] Pierre-Yves Strub, Nikhil Swamy, Cedric Fournet, and Juan Chen. Self-certification: Bootstrapping certified typecheckers in f* with coq. In *POPL '12* [P12], 571–584. http://doi.acm.org/10.1145/2103656.2103723 549121.

[P13] **POPL '13**: *Proc. 40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2013. Association for Computing Machinery. ISBN 978-1-4503-1832-7.

[P13A] Steffen Lösch and Andrew M. Pitts. Full abstraction for nominal scott domains. In *POPL '13* [P13], 3–14. http://doi.acm.org/10.1145/2429069.2429073

[P13B] Ross Tate. The sequential semantics of producer effect systems. In *POPL '13* [P13], 15–26. http://doi.acm.org/10.1145/2429069.2429074

[P13C] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming infinite structures by observations. In *POPL '13* [P13], 27–38. http://doi.acm.org/10.1145/2429069.2429075

[P13D] Guy E. Blelloch and Robert Harper. Cache and i/o efficent functional algorithms. In *POPL '13* [P13], 39–50. http://doi.acm.org/10.1145/2429069.2429077

[P13E] Hiroshi Unno, Tachio Terauchi, and Naoki Kobayashi. Automating relatively complete verification of higher-order functional programs. In *POPL '13* [P13], 75–86. http://doi.acm.org/10.1145/2429069.2429081

[P13F] Robert Atkey, Patricia Johann, and Andrew Kennedy. Abstraction and invariance for algebraically indexed types. In *POPL '13* [P13], 87–100. http://doi.acm.org/10.1145/2429069.2429082

[P13G] Véronique Benzaken, Giuseppe Castagna, Kim Nguyen, and Jérôme Siméon. Static and dynamic semantics of nosql languages. In *POPL '13* [P13], 101–114. http://doi.acm.org/10.1145/2429069.2429083

[P13H] Alexis Goyet. The lambda lambda-bar calculus: A dual calculus for unconstrained strategies. In *POPL '13* [P13], 155–166. http://doi.acm.org/10.1145/2429069.2429089

[P13I] Ugo Dal lago and Barbara Petit. The geometry of types. In *POPL '13* [P13], 167–178. http://doi.acm.org/10.1145/2429069.2429090

[P13J] Sam Staton and Paul Blain Levy. Universal properties of impure programming languages. In *POPL '13* [P13], 179–192. http://doi.acm.org/10.1145/2429069.2429091

[P13K] Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In *POPL '13* [P13], 193–206. http://doi.acm.org/10.1145/2429069.2429093

[P13L] Benjamin Delaware, Bruno C. d. S. Oliveira, and Tom Schrijvers. Meta-theory à la carte. In *POPL '13* [P13], 207–218. http://doi.acm.org/10.1145/2429069.2429094

[P13M] Jonghyun Park, Jeongbong Seo, and Sungwoo Park. A theorem prover for boolean bi. In *POPL '13* [P13], 219–232. http://doi.acm.org/10.1145/2429069.2429095

[P13N] Ganesan Ramalingam and Kapil Vaswani. Fault tolerance via idempotence. In *POPL '13* [P13], 249–262. http://doi.acm.org/10.1145/2429069.2429100

[P13O] Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: Multiparty asynchronous global programming. In *POPL '13* [P13], 263–274. http://doi.acm.org/10.1145/2429069.2429101

[P13P] Luís Caires and João C. Seco. The type discipline of behavioral separation. In *POPL '13* [P13], 275–286. http://doi.acm.org/10.1145/2429069.2429103

[P13Q] Thomas Dinsdale-Young, Lars Birkedal, Philippa Gardner, Matthew Parkinson, and Hongseok Yang. Views: Compositional reasoning for concurrent programs. In *POPL '13* [P13], 287–300. http://doi.acm.org/10.1145/2429069.2429104

[P13R] Jonas B. Jensen, Nick Benton, and Andrew Kennedy. High-level separation logic for low-level code. In *POPL '13* [P13], 301–314. http://doi.acm.org/10.1145/2429069.2429105

[P13S] Delphine Demange, Vincent Laporte, Lei Zhao, Suresh Jagannathan, David Pichardie, and Jan Vitek. Plan b: A buffered memory model for java. In *POPL '13* [P13], 329–342. http://doi.acm.org/10.1145/2429069.2429110

[P13T] Aaron J. Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. Logical relations for fine-grained concurrency. In *POPL '13* [P13], 343–356. http://doi.acm.org/10.1145/2429069.2429111

[P13U] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *POPL '13* [P13], 357–370. http://doi.acm.org/10.1145/2429069.2429113

[P13V] Cedric Fournet, Nikhil Swamy, Juan Chen, Pierre-Evariste Dagand, Pierre-Yves Strub, and Benjamin Livshits. Fully abstract compilation to javascript. In *POPL '13* [P13], 371–384. http://doi.acm.org/10.1145/2429069.2429114

[P13W] Andrew D. Gordon, Mihhail Aizatulin, Johannes Borgstrom, Guillaume Claret, Thore Graepel, Aditya V. Nori, Sriram K. Rajamani, and Claudio Russo. A model-learner pattern for bayesian reasoning. In *POPL '13* [P13], 403–416. http://doi.acm.org/10.1145/2429069.2429119

[P13X] Kohei Suenaga, Hiroyoshi Sekine, and Ichiro Hasuo. Hyperstream processing systems: Nonstandard modeling of continuous-time signals. In *POPL '13* [P13], 417–430. http://doi.acm.org/10.1145/2429069.2429120

[P13Y] Dimitrios Vytiniotis, Simon Peyton Jones, Koen Claessen, and Dan Rosén. Halo: Haskell to logic through denotational semantics. In *POPL '13* [P13], 431–442. http://doi.acm.org/10.1145/2429069.2429121

[P13Z] Ali Sinan Koksal, Yewen Pu, Saurabh Srivastava, Rastislav Bodik, Jasmin Fisher, and Nir Piterman. Synthesis of biological models from mutation experiments. In *POPL '13* [P13], 469–482. http://doi.acm.org/10.1145/2429069.2429125

[P13Γ] Aquinas Hobor and Jules Villard. The ramifications of sharing in data structures. In *POPL '13* [P13], 523–536. http://doi.acm.org/10.1145/2429069.2429131

[P13Δ] Ruy Ley-Wild and Aleksandar Nanevski. Subjective auxiliary state for coarse-grained concurrency. In *POPL '13* [P13], 561–574. http://doi.acm.org/10.1145/2429069.2429134

[P14] **POPL '14**: *Proc. 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2014. Association for Computing Machinery. ISBN 978-1-4503-2544-8.

[P14A] Giuseppe Castagna, Kim Nguyen, Zhiwu Xu, Hyeonseung Im, Sergueï Lenglet, and Luca Padovani. Polymorphic functions with set-theoretic types: Part 1: Syntax, semantics, and evaluation. In *POPL '14* [P14], 5–17. http://doi.acm.org/10.1145/2535838.2535840

[P14B] Scott Kilpatrick, Derek Dreyer, Simon Peyton Jones, and Simon Marlow. Backpack: Retrofitting haskell with interfaces. In *POPL '14* [P14], 19–31. http://doi.acm.org/10.1145/2535838.2535884

[P14C] Chris Casinghino, Vilhelm Sjöberg, and Stephanie Weirich. Combining proofs and programs in a dependently typed language. In *POPL '14* [P14], 33–45. http://doi.acm.org/10.1145/2535838.2535883

[P14D] Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In *POPL '14* [P14], 61–72. http://doi.acm.org/10.1145/2535838.2535873

[P14E] Devin Coughlin and Bor-Yuh Evan Chang. Fissile type analysis: Modular checking of almost everywhere invariants. In *POPL '14* [P14], 73–85. http://doi.acm.org/10.1145/2535838.2535855

[P14F] Martin Bodin, Arthur Chargueraud, Daniele Filaretti, Philippa Gardner, Sergio Maffeis, Daiva Naudziuniene, Alan Schmitt, and Gareth Smith. A trusted mechanised javascript specification. In *POPL '14* [P14], 87–100. http://doi.acm.org/10.1145/2535838.2535876

[P14G] Robbert Krebbers. An operational and axiomatic semantics for non-determinism and sequence points in c. In *POPL '14* [P14], 101–112. http://doi.acm.org/10.1145/2535838.2535878

[P14H] Vijay D'Silva, Leopold Haller, and Daniel Kroening. Abstract satisfaction. In *POPL '14* [P14], 139–150. http://doi.acm.org/10.1145/2535838.2535868

[P14I] Arthur Azevedo de Amorim, Nathan Collins, André DeHon, Delphine Demange, Cătălin Hriţcu, David Pichardie, Benjamin C. Pierce, Randy Pollack, and Andrew Tolmach. A verified information-flow architecture. In *POPL '14* [P14], 165–178. http://doi.acm.org/10.1145/2535838.2535839

[P14J] Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. Cakeml: A verified implementation of ml. In *POPL '14* [P14], 179–191. http://doi.acm.org/10.1145/2535838.2535841

[P14K] Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella-Béguelin. Probabilistic relational verification for cryptographic implementations. In *POPL '14* [P14], 193–205. http://doi.acm.org/10.1145/2535838.2535847

[P14L] Lindsey Kuper, Aaron Turon, Neelakantan R. Krishnaswami, and Ryan R. Newton. Freeze after writing: Quasi-deterministic parallel programming with lvars. In *POPL '14* [P14], 257–270. http://doi.acm.org/10.1145/2535838.2535842

[P14M] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: Specification, verification, optimality. In *POPL '14* [P14], 271–284. http://doi.acm.org/10.1145/2535838.2535848

[P14N] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *POPL '14* [P14], 297–308. http://doi.acm.org/10.1145/2535838.2535872

[P14O] Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic pcf. In *POPL '14* [P14], 309–320. http://doi.acm.org/10.1145/2535838.2535865

[P14P] Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. Tabular: A schema-driven probabilistic programming language. In *POPL '14* [P14], 321–334. http://doi.acm.org/10.1145/2535838.2535850

[P14Q] Ilya Sergey, Dimitrios Vytiniotis, and Simon Peyton Jones. Modular, higher-order cardinality analysis in theory and practice. In *POPL '14* [P14], 335–347. http://doi.acm.org/10.1145/2535838.2535861

[P14R] Andrew Cave, Francisco Ferreira, Prakash Panangaden, and Brigitte Pientka. Fair reactive programming. In *POPL '14* [P14], 361–372. http://doi.acm.org/10.1145/2535838.2535881

[P14S] Nathan Chong, Alastair F. Donaldson, and Jeroen Ketema. A sound and complete abstraction for reasoning about parallel prefix sums. In *POPL '14* [P14], 397–409. http://doi.acm.org/10.1145/2535838.2535882

[P14T] Andrew Miller, Michael Hicks, Jonathan Katz, and Elaine Shi. Authenticated data structures, generically. In *POPL '14* [P14], 411–423. http://doi.acm.org/10.1145/2535838.2535851

[P14U] Nikhil Swamy, Cedric Fournet, Aseem Rastogi, Karthikeyan Bhargavan, Juan Chen, Pierre-Yves Strub, and Gavin Bierman. Gradual typing embedded securely in javascript. In *POPL '14* [P14], 425–437. http://doi.acm.org/10.1145/2535838.2535889

[P14V] Fan Long, Stelios Sidiroglou-Douskos, Deokhwan Kim, and Martin Rinard. Sound input filter generation for integer overflow errors. In *POPL '14* [P14], 439–452. http://doi.acm.org/10.1145/2535838.2535888

[P14W] James Brotherston and Jules Villard. Parametric completeness for separation theories. In *POPL '14* [P14], 453–464. http://doi.acm.org/10.1145/2535838.2535844

[P14X] Zhé Hóu, Ranald Clouston, Rajeev Goré, and Alwen Tiu. Proof search for propositional abstract separation logics via labelled sequents. In *POPL '14* [P14], 465–476. http://doi.acm.org/10.1145/2535838.2535864

[P14Y] Wonyeol Lee and Sungwoo Park. A proof system for separation logic with magic wand. In *POPL '14* [P14], 477–490. http://doi.acm.org/10.1145/2535838.2535871

[P14Z] Robert Atkey. From parametricity to conservation laws, via noether's theorem. In *POPL '14* [P14], 491–502. http://doi.acm.org/10.1145/2535838.2535867

[P14Γ] Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In *POPL '14* [P14], 503–515. http://doi.acm.org/10.1145/2535838.2535852

[P14Δ] Andrzej S. Murawski and Nikos Tzevelekos. Game semantics for interface middleweight java. In *POPL '14* [P14], 517–528. http://doi.acm.org/10.1145/2535838.2535880

[P14Θ] Swarat Chaudhuri, Azadeh Farzan, and Zachary Kincaid. Consistency analysis of decision-making programs. In *POPL '14* [P14], 555–567. http://doi.acm.org/10.1145/2535838.2535858

[P14Λ] Danfeng Zhang and Andrew C. Myers. Toward general diagnosis of static errors. In *POPL '14* [P14], 569–581. http://doi.acm.org/10.1145/2535838.2535870

[P14Ξ] Sheng Chen and Martin Erwig. Counter-factual typing for debugging type errors. In *POPL '14* [P14], 583–594. http://doi.acm.org/10.1145/2535838.2535863

[P14Π] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. Symbolic optimization with smt solvers. In *POPL '14* [P14], 607–618. http://doi.acm.org/10.1145/2535838.2535857

[P14Σ] Nick Benton, Martin Hofmann, and Vivek Nigam. Abstract effects and proof-relevant logical relations. In *POPL '14* [P14], 619–631. http://doi.acm.org/10.1145/2535838.2535869

[P14Υ] Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. In *POPL '14* [P14], 633–645. http://doi.acm.org/10.1145/2535838.2535846

[P14Φ] Michele Pagani, Peter Selinger, and Benoît Valiron. Applying quantitative semantics to higher-order quantum computing. In *POPL '14* [P14], 647–658. http://doi.acm.org/10.1145/2535838.2535879

[P14Ψ] Richard A. Eisenberg, Dimitrios Vytiniotis, Simon Peyton Jones, and Stephanie Weirich. Closed type families with overlapping equations. In *POPL '14* [P14], 671–683. http://doi.acm.org/10.1145/2535838.2535856

[P15] **POPL '15**: *Proc. 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2015. Association for Computing Machinery. ISBN 978-1-4503-3300-9.

[P15A] Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In *POPL '15* [P15], 3–16. http://doi.acm.org/10.1145/2676726.2676970

[P15B] Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. Integrating linear and dependent types. In *POPL '15* [P15], 17–30. http://doi.acm.org/10.1145/2676726.2676969

[P15C] Kristina Sojakova. Higher inductive types as homotopy-initial algebras. In *POPL '15* [P15], 31–42. http://doi.acm.org/10.1145/2676726.2676983

[P15D] Minh Ngo, Fabio Massacci, Dimiter Milushev, and Frank Piessens. Runtime enforcement of security policies on black box reactive programs. In *POPL '15* [P15], 43–54. http://doi.acm.org/10.1145/2676726.2676978

[P15E] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. Higher-order approximate relational refinement types for mechanism design and differ-

ential privacy. In *POPL '15* [P15], 55–68.
http://doi.acm.org/10.1145/2676726.2677000

[P15F] Hamid Ebadi, David Sands, and Gerardo Schneider. Differential privacy: Now it's getting personal. In *POPL '15* [P15], 69–81.
http://doi.acm.org/10.1145/2676726.2677005

[P15G] Aseem Rastogi, Nikhil Swamy, Cédric Fournet, Gavin Bierman, and Panagiotis Vekris. Safe & efficient gradual typing for typescript. In *POPL '15* [P15], 167–180.
http://doi.acm.org/10.1145/2676726.2676971

[P15H] Michael Greenberg. Space-efficient manifest contracts. In *POPL '15* [P15], 181–194.
http://doi.acm.org/10.1145/2676726.2676967

[P15I] Taro Sekiyama, Yuki Nishida, and Atsushi Igarashi. Manifest contracts for datatypes. In *POPL '15* [P15], 195–207.
http://doi.acm.org/10.1145/2676726.2676996

[P15J] Roberto Giacobazzi, Francesco Logozzo, and Francesco Ranzato. Analyzing program analyses. In *POPL '15* [P15], 261–273.
http://doi.acm.org/10.1145/2676726.2676987

[P15K] Gordon Stewart, Lennart Beringer, Santiago Cuellar, and Andrew W. Appel. Compositional compcert. In *POPL '15* [P15], 275–287. http://doi.acm.org/10.1145/2676726.2676985

[P15L] Giuseppe Castagna, Kim Nguyen, Zhiwu Xu, and Pietro Abate. Polymorphic functions with set-theoretic types: Part 2: Local type inference and type reconstruction. In *POPL '15* [P15], 289–302.
http://doi.acm.org/10.1145/2676726.2676991

[P15M] Ronald Garcia and Matteo Cimini. Principal type schemes for gradual programs. In *POPL '15* [P15], 303–315.
http://doi.acm.org/10.1145/2676726.2676992

[P15N] Luísa Lourenço and Luís Caires. Dependent information flow types. In *POPL '15* [P15], 317–328.
http://doi.acm.org/10.1145/2676726.2676994

[P15O] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for netkat. In *POPL '15* [P15], 343–355.
http://doi.acm.org/10.1145/2676726.2677011

[P15P] Vilhelm Sjöberg and Stephanie Weirich. Programming up to congruence. In *POPL '15* [P15], 369–382.
http://doi.acm.org/10.1145/2676726.2676974

[P15Q] Kazunori Tobisawa. A meta lambda calculus with cross-level computation. In *POPL '15* [P15], 383–393.
http://doi.acm.org/10.1145/2676726.2676976

[P15R] Sam Staton. Algebraic effects, linearity, and quantum programming languages. In *POPL '15* [P15], 395–406.
http://doi.acm.org/10.1145/2676726.2676999

[P15S] Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proof spaces for unbounded parallelism. In *POPL '15* [P15], 407–420.
http://doi.acm.org/10.1145/2676726.2677012

[P15T] Davide Sangiorgi. Equations, contractions, and unique solutions. In *POPL '15* [P15], 421–432.
http://doi.acm.org/10.1145/2676726.2676965

[P15U] Ashutosh Gupta, Thomas A. Henzinger, Arjun Radhakrishna, Roopsha Samanta, and Thorsten Tarrach. Succinct representation of concurrent trace sets. In *POPL '15* [P15], 433–444.
http://doi.acm.org/10.1145/2676726.2677008

[P15V] Denis Bogdănaş and Grigore Roşu. K-java: A complete semantics of java. In *POPL '15* [P15], 445–456.
http://doi.acm.org/10.1145/2676726.2676982

[P15W] Michael D. Adams. Towards the essence of hygiene. In *POPL '15* [P15], 457–469. http://doi.acm.org/10.1145/2676726.2677013

[P15X] Matt Brown and Jens Palsberg. Self-representation in girard's system u. In *POPL '15* [P15], 471–484.
http://doi.acm.org/10.1145/2676726.2676988

[P15Y] Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In *POPL '15* [P15], 489–501.
http://doi.acm.org/10.1145/2676726.2677001

[P15Z] Fei He, Xiaowei Gao, Bow-Yaw Wang, and Lijun Zhang. Leveraging weighted automata in compositional reasoning about concurrent probabilistic systems. In *POPL '15* [P15], 503–514.
http://doi.acm.org/10.1145/2676726.2676998

[P15Γ] Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In *POPL '15* [P15], 515–526.
http://doi.acm.org/10.1145/2676726.2676993

[P15Δ] Osbert Bastani, Saswat Anand, and Alex Aiken. Specification inference using context-free language reachability. In *POPL '15* [P15], 553–566. http://doi.acm.org/10.1145/2676726.2676977

[P15Θ] Pieter Agten, Bart Jacobs, and Frank Piessens. Sound modular verification of c code executing in an unverified context. In *POPL '15* [P15], 581–594.
http://doi.acm.org/10.1145/2676726.2676972

[P15Λ] Ronghui Gu, Jérémie Koenig, Tahina Ramananandro, Zhong Shao, Xiongnan (Newman) Wu, Shu-Chun Weng, Haozhong Zhang, and Yu Guo. Deep specifications and certified abstraction layers. In *POPL '15* [P15], 595–608.
http://doi.acm.org/10.1145/2676726.2676975

[P15Ξ] Adam Chlipala. From network interface to multithreaded web applications: A case study in modular program verification. In *POPL '15* [P15], 609–622.
http://doi.acm.org/10.1145/2676726.2677003

[P15Π] Karl Crary and Michael J. Sullivan. A calculus for relaxed memory. In *POPL '15* [P15], 623–636.
http://doi.acm.org/10.1145/2676726.2676984

[P15Σ] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *POPL '15* [P15], 637–650.
http://doi.acm.org/10.1145/2676726.2676980

[P16] **POPL '16**: *Proc. 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 2016. Association for Computing Machinery. ISBN 978-1-4503-3549-2.

[P16A] Matt Brown and Jens Palsberg. Breaking through the normalization barrier: A self-interpreter for f-omega. In *POPL '16* [P16], 5–17. http://doi.acm.org/10.1145/2837614.2837623

[P16B] Yufei Cai, Paolo G. Giarrusso, and Klaus Ostermann. System f-omega with equirecursive types for datatype-generic programming. In *POPL '16* [P16], 30–43.
http://doi.acm.org/10.1145/2837614.2837660

[P16C] Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. A theory of effects and resources: Adjunction models and polarised calculi. In *POPL '16* [P16], 44–56.
http://doi.acm.org/10.1145/2837614.2837652

[P16D] Akihiro Murase, Tachio Terauchi, Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Temporal verification of higher-order functional programs. In *POPL '16* [P16], 57–68.
http://doi.acm.org/10.1145/2837614.2837667

[P16E] James Brotherston, Nikos Gorogiannis, Max Kanovich, and Reuben Rowe. Model checking for symbolic-heap separation logic with inductive predicates. In *POPL '16* [P16], 84–96.
http://doi.acm.org/10.1145/2837614.2837621

[P16F] Eric Koskinen and Junfeng Yang. Reducing crash recoverability to reachability. In *POPL '16* [P16], 97–108.
http://doi.acm.org/10.1145/2837614.2837648

[P16G] Dominique Devriese, Marco Patrignani, and Frank Piessens. Fully-abstract compilation by approximate back-translation. In *POPL '16* [P16], 164–177.
http://doi.acm.org/10.1145/2837614.2837618

[P16H] Jeehoon Kang, Yoonseung Kim, Chung-Kil Hur, Derek Dreyer, and Viktor Vafeiadis. Lightweight verification of separate compilation. In *POPL '16* [P16], 178–190.
http://doi.acm.org/10.1145/2837614.2837642

[P16I] Ed Robbins, Andy King, and Tom Schrijvers. From minx to minc: Semantics-driven decompilation of recursive datatypes. In *POPL '16* [P16], 191–203.
http://doi.acm.org/10.1145/2837614.2837633

[P16J] Florian Lorenzen and Sebastian Erdweg. Sound type-dependent syntactic language extension. In *POPL '16* [P16], 204–216. http://doi.acm.org/10.1145/2837614.2837644

[P16K] Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoue, and Santiago Zanella-Béguelin. Dependent types and multi-monadic effects in f*. In *POPL '16* [P16], 256–270. http://doi.acm.org/10.1145/2837614.2837655

[P16L] Johannes Borgström, Andrew D. Gordon, Long Ouyang, Claudio Russo, Adam Ścibior, and Marcin Szymczak. Fabular: Regression formulas as probabilistic programming. In *POPL '16* [P16], 271–283. http://doi.acm.org/10.1145/2837614.2837653

[P16M] Fan Long and Martin Rinard. Automatic patch generation by learning correct code. In *POPL '16* [P16], 298–312. http://doi.acm.org/10.1145/2837614.2837617

[P16N] Mohsen Lesani, Christian J. Bell, and Adam Chlipala. Chapar: Certified causally consistent distributed key-value stores. In *POPL '16* [P16], 357–370. http://doi.acm.org/10.1145/2837614.2837622

[P16O] Hongjin Liang and Xinyu Feng. A program logic for concurrent objects under fair scheduling. In *POPL '16* [P16], 385–399. http://doi.acm.org/10.1145/2837614.2837635

[P16P] Cezara Drăgoi, Thomas A. Henzinger, and Damien Zufferey. Psync: A partially synchronous language for fault-tolerant distributed algorithms. In *POPL '16* [P16], 400–415. http://doi.acm.org/10.1145/2837614.2837650

[P16Q] Sheng Chen and Martin Erwig. Principal type inference for gadts. In *POPL '16* [P16], 416–428. http://doi.acm.org/10.1145/2837614.2837665

[P16R] Ronald Garcia, Alison M. Clark, and Éric Tanter. Abstracting gradual typing. In *POPL '16* [P16], 429–442. http://doi.acm.org/10.1145/2837614.2837670

[P16S] Matteo Cimini and Jeremy G. Siek. The gradualizer: A methodology and algorithm for generating gradual type systems. In *POPL '16* [P16], 443–455. http://doi.acm.org/10.1145/2837614.2837632

[P16T] Dominic Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *POPL '16* [P16], 568–581. http://doi.acm.org/10.1145/2837614.2837634

[P16U] Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment for higher-order session types. In *POPL '16* [P16], 582–594. http://doi.acm.org/10.1145/2837614.2837662

[P16V] Davide Sangiorgi and Valeria Vignudelli. Environmental bisimulations for probabilistic higher-order languages. In *POPL '16* [P16], 595–607. http://doi.acm.org/10.1145/2837614.2837651

[P16W] Ori Lahav, Nick Giannarakis, and Viktor Vafeiadis. Taming release-acquire consistency. In *POPL '16* [P16], 649–662. http://doi.acm.org/10.1145/2837614.2837643

[P16X] Koko Muroya, Naohiko Hoshino, and Ichiro Hasuo. Memoryful geometry of interaction ii: Recursion and adequacy. In *POPL '16* [P16], 748–760. http://doi.acm.org/10.1145/2837614.2837672

[P16Y] Jonathan Frankle, Peter-Michael Osera, David Walker, and Steve Zdancewic. Example-directed synthesis: A type-theoretic interpretation. In *POPL '16* [P16], 802–815. http://doi.acm.org/10.1145/2837614.2837629