Two Ways of Doing One Thing: When an Apparent Design Wart Is a Semantic Boundary

Brendan Sechter

2026-08-07

A system had grown two ways of doing what looked like one thing. The obvious move was to tidy them into one. The tidying turns out to be impossible, and the reason it is impossible is the reason the two ways exist.

The argument that settles it needs no specialist knowledge and fits in a sentence. One of the two cases has two things to report and only one slot to report them in. Whichever thing the slot is given, the other is lost. The other case has only one thing to report, so the single slot is exactly enough. **That is a counting argument, it is decided before any code is written, and it is not the argument an engineer reaches for by default.**

The engineer's instinct is to look at the cases and ask whether they resemble one another. They did. **Every one of the nine measured occurrences had exactly the shape that invited the tidy-up**, and the measurement encouraged precisely the wrong conclusion. The resemblance was real and it was irrelevant, because the defect was never in the instances. **It was in the interface they would have had to share.**

This article is about that distinction, which is between evidence about members of a class and evidence about the channel the class must pass through. The second dominates the first and is cheaper to check.

The article reports the measurement, the way the measurement pointed the wrong direction, and the argument that settled it. It also reports a rule this author shipped one increment earlier which turns out to be **stricter than the property it enforces**, excluding ten of twenty-four cases for no reason. No test found that. It surfaced while gathering data for this article.

How to read this

The general argument is in the opening, in the section called The Argument That Settled It, and in Pattern Extraction. Those three need nothing but attention. The sections between them work the argument through a real case with real numbers, and they use the vocabulary of the trade. Every term is glossed at first use, but a reader who wants the result and not the machinery can take those three and stop.

The mathematics is optional throughout. It appears because the central result is a counting argument, and counting arguments are the one place where a line of notation settles what a paragraph of prose can only assert. Wherever notation appears, the sentence before it says the same thing in words.

What this is a case study of

The setting is compiler design, and specifically the part of a compiler called the **backend**, which is the stage that turns a program the compiler has already understood into instructions a processor can run. The project is Keleusma, whose backend is described in the previous

article in this series. **No compiler background is required.** The shape of the problem is a general one. A system offers one abstraction whose instances divide into two classes with genuinely different observable behaviour, and an engineer, seeing two implementations, tries to collapse them.

The question this article treats is when that instinct is right and when it is a category error. The answer offered is a test, and it is this. **Count the observable events each class produces, and count the channels the proposed unification provides.** If the counts differ, the unification is lossy and no amount of shape analysis on individual instances will rescue it, because the loss lives in the interface and not in the instances.

The instinct to unify is not a bad one. Two ways of doing one thing is a real cost, paid by every consumer who must learn both and by every future change that must be made twice. This article does not argue against tidiness. It argues that tidiness is a claim about the world, and like any other claim it can be checked before it is acted on, cheaply, and that the check is not the one an engineer reaches for by default.

The Setting

A **coroutine** is a computation that can suspend in the middle, hand a value to whoever called it, and later be resumed with a value handed back. **The two-flavour split this article is about is not peculiar to Keleusma**, and the language specifications record it plainly. Python began with generators that only produce values in Python Enhancement Proposal 255 , or PEP 255, and later admitted values back in at the suspension point in PEP 342, which is exactly the step that turns one observable event into two, then separated the coroutine from the generator entirely in PEP 492. Lua exposes asymmetric coroutines in which `resume` and `yield` are distinct operations and a coroutine additionally **returns** when its body completes, so that a caller must distinguish a yield from a return. The European Computer Manufacturers Association standard 262, or ECMA-262, which specifies JavaScript, gives its generator results an explicit `done` flag for the same reason, C# iterators make the distinction syntactic, and C++ separates `co_yield` from `co_return` at the level of the grammar. **Every one of these designs carries a discriminator**, and that is the fact this article rediscovers from the compiler side.

Keleusma has two forms.

A **`yield` function** suspends a bounded number of times and then finishes. It is a total function with interruptions. The type system guarantees it terminates.

A **`loop` function** suspends forever. It is **productively divergent**, which is the property that it never finishes but always makes progress, so a **host**, meaning whatever program is driving the coroutine from outside, can pull values from it indefinitely and each pull terminates. This is the language's stream abstraction and it is the normal way a Keleusma program does work over time.

Both are written with the same `yield` keyword in the body. The difference is the declaration, and the type checker already separates them into different categories with different rules about what may call what.

Notation

The formal machinery below is worth the six symbols it costs, because the central result is a cardinality argument that prose states as an opinion and arithmetic settles in one line. A reader who prefers prose can take the following and skip to the results. **A terminating coroutine emits two values per call and a return slot holds one.**

Let W denote the machine word type, so that $|W| = 2^{64}$ on the present target. Let K denote the set of coroutine **chunks**, a chunk being one compiled unit of code, partitioned by the type checker into

$$K = K_{\Sigma} \sqcup K_{\Upsilon}$$

where K_{Σ} are the divergent stream chunks and K_{Υ} the terminating ones. Write $|K_{\Sigma}| = 24$ and $|K_{\Upsilon}| = 1$ for the present corpus.

An execution of a chunk is driven by an initial argument $a \in W$ and a sequence of resume values $\vec{r} = (r_1, r_2, \ldots)$. It produces an **observable trace**, a sequence over the alphabet $\{\mathsf{Y}(v), \mathsf{F}(w)\}$ in which Y marks a suspension carrying v and F a completion carrying w. For a terminating chunk with n suspensions the trace is finite and ends in a completion,

$$\mathcal{T}_{\Upsilon}(k, a, \vec{r}) \;=\; \big\langle\, \mathsf{Y}(v_1),\, \mathsf{Y}(v_2),\, \ldots,\, \mathsf{Y}(v_n),\, \mathsf{F}(w) \,\big\rangle$$

while for a divergent chunk it is infinite and contains no completion at all,

$$\mathcal{T}_{\Sigma}(k, a, \vec{r}) \;=\; \big\langle\, \mathsf{Y}(v_0),\, \mathsf{Y}(v_1),\, \mathsf{Y}(v_2),\, \ldots \,\big\rangle.$$

That asymmetry is the entire article. The two trace shapes differ not in length but in **alphabet**. One uses both letters and the other uses one.

A **calling convention** is an encoding ε that carries a trace into what a host can observe from native calls. Write $\mathrm{chan}(\varepsilon)$ for the number of distinguishable W-valued channels a single native call exposes under ε, and $\mathrm{obs}(k)$ for the number of distinct observable values a single call of k must convey.

The two conventions

The generator emits native code that a host program calls. When the generated code suspends, the value has to reach the host. Two mechanisms are available.

The **callback convention** declares an external function, here called `kel_yield`, that takes the yielded value and returns the resume value. Generated code calls it at each suspension point and carries on. The host implements it. Control passes to the host and back without the generated function returning, so the generated function's **stack frame** stays live throughout, a stack frame being the scratch space a function reserves for its own working values and gives back when it returns. This is the stackful shape, of which the context-switching functions of the Portable Operating System Interface or POSIX are the oldest standardised instance, and its defining property is exactly that a frame outlives the suspension.

The **return convention** ends the native call at the suspension. The yielded value is the function's return value. The host receives it as an ordinary result, does whatever it does, and calls the function again with the resume value as an argument. No frame persists between suspensions.

The return convention is strictly cheaper. It needs no external symbol, no callback indirection, and crucially **no stack frame alive during host code**, which matters to a project whose entire premise is statically bounded memory. A frame held across an arbitrary amount of host activity is a frame the memory analysis cannot account for.

The compiler infrastructure this backend targets already names both shapes. The coroutine intrinsics of the Low Level Virtual Machine compiler infrastructure or LLVM provide a switch-resume family, in which a coroutine handle is resumed through a function

pointer and the frame persists in an explicit allocation, and a **returned-continuation** family, `llvm.coro.id.retcon`, in which the coroutine returns at each suspension and hands back a continuation to be called next. The two families correspond closely to the callback and return conventions described here. Frame liveness across a suspension is the same property that LLVM stack maps exist to describe, and the WebAssembly stack-switching proposal is the same design question posed for a portable target. **None of these settle the choice**, and that is worth saying, because they establish that both shapes are standard and that the trade is real, not that either is correct here.

How the backend arrived at two

The `yield` function was implemented first, on the callback convention, because it is the general mechanism and it was obviously correct.

The `loop` function was implemented later. Its **bytecode**, meaning the compact intermediate instructions the compiler produces before machine code, has a recognisable shape, being an opening marker, a body, a stack cleanup, and a reset instruction that rewinds execution to just after the marker and hands control to the host. The reset clears every local variable. **That clearing is the key structural fact.** If the body's suspension is the last thing on every path, then no local outlives the suspension, because the reset would have cleared it anyway. Nothing needs to persist, so nothing needs a frame, so the return convention applies.

Two conventions resulted, not by design but by two increments arriving from different directions. That is exactly the situation in which an engineer reaches for a unification.

The Measurement That Pointed The Wrong Way

Before proposing anything, the corpus was measured. The instrument walks every compiled program in the project's shipped examples and self-hosted compiler sources, classifies each coroutine chunk, and reports the distribution.

The distribution of stream chunks

Class	Count
Single suspension, at the end of the body	22
Multiple suspensions at the top level	**0**
Suspensions nested inside conditionals	1
No suspension of its own, delegating to a callee	1
Total stream chunks	**24**

The zero is worth pausing on. An earlier design in this project assumed the general case was a body with several suspension points that would need to be reordered relative to one another. **No such body exists in the corpus.** That design was specified in some detail before the count was taken, and the count deleted it.

The nested case looked like the hard one until it was measured per suspension instead of per chunk. All nineteen of its suspensions sit inside conditionals and **not one sits inside a loop**. A suspension inside a conditional is a control-flow join, where every path still suspends once and ends. A suspension inside a loop crosses a back edge and genuinely needs a frame. The word **nested** had been covering both, and they have entirely different costs.

The distribution of terminating coroutines

Property	Count
`yield` function chunks in the whole corpus	**1**
… whose every suspension is immediately followed by a return	**1**

One. The entire terminating-coroutine population of the corpus is a single chunk, and that chunk has, in nine of nine cases, precisely the shape that appeared to invite the return convention, which is to suspend and then return.

This is where the measurement pointed the wrong way. A distribution of nine out of nine looks like an invitation. It reads as evidence that the shape is natural, that the callback is an accident of implementation order, and that a unification is available at the cost of a shape check. The author of this article drafted exactly that plan.

The Argument That Settled It

The plan was checked against the smallest possible instance before it was implemented, which is the only thing in this article that went right on the first attempt.

Consider a terminating coroutine that suspends once with its argument.

```
yield main(a: Word) -> Word { yield a }
```

It compiles to three instructions, which load the argument, suspend, and return. The suspension instruction pops the value to yield and pushes the value the host resumes with, so the return instruction returns the resume value.

Run it on the reference virtual machine and it produces **two observable events**.

1. **Suspension**, carrying `a`, the yielded value. 2. **Completion**, carrying `r`, the resume value.

A return-based **lowering**, meaning a translation from the intermediate form down to machine code, has **one** return slot. Give the slot the yielded value and the completion is gone. Give it the completion and the suspension is gone. There is no third option, and **no analysis of the chunk's shape changes this**, because the deficiency is in the calling convention and not in the chunk.

The obvious objection is that the completion could arrive on a second call, and the reason it cannot is the part of the return convention that is easiest to skip over. The convention has no resumption point. The host drives the chunk by calling a plain function again, passing the resume value as the argument, so the sequence the host executes is

$$f(a) = v_0, \qquad f(r_1) = v_1, \qquad f(r_2) = v_2, \qquad \ldots$$

and f carries nothing between calls. For a divergent chunk that is exactly right, because the reset instruction rewinds to the same place and clears every local, so re-entering from the top is what resumption means there. **For a terminating chunk it is wrong.** A second call re-runs the body from the beginning and suspends again, so it yields a second time and never reaches the completion at all.

Everything the host will ever learn about one terminating execution therefore has to arrive from the first call, which is why the budget is one slot rather than one slot per event.

The counting argument, stated once

Under the return convention a single native call exposes exactly the return **register**, which is one of the processor's few named storage slots, so $\mathrm{chan}(\varepsilon_{\mathrm{ret}}) = 1$. The chunk above must convey both v and w from one call, so $\mathrm{obs}(k) = 2$. An encoding faithful to the trace must therefore be an injection

$$\varepsilon_{\mathrm{ret}} : W \times W \longrightarrow W$$

and no such injection exists whenever $|W| \geq 2$, by cardinality,

$$|W \times W| \;=\; |W|^2 \;=\; 2^{128} \;>\; 2^{64} \;=\; |W|.$$

The same statement in bits is the one that generalises, and it is the elementary case of the channel-capacity argument Shannon 1948 introduced. An interface of m machine words carries $64m$ bits and a trace of k words needs $64k$, so a faithful encoding requires

$$64k \;\leq\; 64m \qquad\Longleftrightarrow\qquad k \;\leq\; m,$$

and here $k = 2$ against $m = 1$. **The shortfall is 64 bits**, which is to say a whole second word.

The general statement is immediate. For any chunk k and convention ε,

$$\mathrm{obs}(k) \;>\; \mathrm{chan}(\varepsilon) \quad\Longrightarrow\quad \varepsilon \text{ is not injective on } \mathcal{T}(k),$$

and a non-injective encoding is one in which two distinct executions become indistinguishable to the host, which is precisely what it means for a lowering to be wrong.

The deficit does not grow with the number of suspensions, which deserves a check and not an assumption. A terminating chunk with n suspensions emits n yielded values and one completion, so its trace carries $n + 1$ words and the executions it can distinguish number $|W|^{n+1}$. The single call it is granted distinguishes $|W|$ of them, so the collapse factor is

$$\frac{|W|^{n+1}}{|W|} \;=\; |W|^{n},$$

which is catastrophic at every n and no worse in kind at $n = 9$ than at $n = 1$. **The smallest instance was therefore the right one to check**, and checking a larger one would have added nothing.

The pigeonhole is doing all of the work, and it is worth noticing how little it needs. It does not inspect the body, the number of suspensions, the control flow, or the corpus. It needs the alphabet of the trace and the width of the interface, both of which are known before any code is written.

The claim is false in its strong form, and the reference work is what showed it

A return slot is not one word. The System V AMD64 application binary interface , or ABI, which is the document that fixes how functions pass arguments and return results on a given machine, returns a two-eightbyte aggregate in the register pair `RAX:RDX`, and AAPCS64

returns small aggregates in `X0` and `X1`. So $\mathrm{chan}(\varepsilon_{\mathrm{ret}})$ is a property of the chosen signature rather than a constant of the machine, and the honest statement is conditional.

$$\mathrm{chan}(\varepsilon_{\mathrm{ret}}) = 1 \ \text{ if the return type is } W, \qquad \mathrm{chan}(\varepsilon_{\mathrm{ret}}) = 2 \ \text{ if it is } W \times W.$$

Under a two-word return the injection $\varepsilon : W \times W \to W \times W$ exists trivially, and the unification this article rejects becomes **representable**.

Two designs are being run together here and they cost very different amounts, so they are worth separating. Cramming both events into one call needs the pair $W \times W$, which is 128 bits against 64, **a shortfall of a whole word**. Making the convention re-entrant with a discriminator needs only

$$|\{\mathsf{Y}, \mathsf{F}\} \times W| \ = \ 2 \cdot 2^{64} \ = \ 2^{65},$$

a shortfall of exactly one bit. That single bit is what forces a second register, since $\lceil 65/64 \rceil = 2$, and it is the whole reason the register-pair rules in the two application binary interfaces matter to this decision. **The expensive-sounding option costs one bit of information and one register of encoding**, a very different proposition from carrying a second word. The pigeonhole refutes the unification *at the signature the backend currently emits*, not for all time.

That distinction matters and it was not visible from inside the problem. It surfaces a **third option** , which is to widen every coroutine entry point so that it returns a discriminated pair,

$$f : W \longrightarrow (\ \mathsf{tag} \in \{\mathsf{Y}, \mathsf{F}\},\ W\ )$$

so that one convention carries both trace alphabets. The cost is that every call of every coroutine pays a wider return and a host-side discrimination, including the **95.83 percent** of the corpus, twenty-three chunks of twenty-four, that need only one letter.

Whether that is cheaper than two conventions is an empirical question this article does not answer, and the shape of the comparison can be stated without measuring anything. Writing c_{tag} for the per-call cost of the discriminator and c_{meta} for the one-off cost of declaring and dispatching two conventions, the totals over N calls are

$$C_{\mathrm{one}} = c_{\mathrm{tag}} N, \qquad C_{\mathrm{two}} = c_{\mathrm{meta}},$$

so the widened return is cheaper only below

$$N^{*} \ = \ \frac{c_{\mathrm{meta}}}{c_{\mathrm{tag}}}.$$

A per-call term always loses to a constant eventually, and a stream abstraction exists to be called many times, so the crossover is the whole question. **Neither constant is published and this article declines to invent them**, which is why the recommendation below rests on the memory property and not on a cost model.

Two production language runtimes already do exactly this, which is the strongest argument available that the option is practical rather than merely representable. A Kotlin suspending function returns its ordinary result, or a distinguished `COROUTINE_SUSPENDED` sentinel to signal that it suspended instead. That is a discriminated return in the narrowest possible encoding, a single reserved value carved out of the result type. A Rust future's `poll` returns

`Poll::Ready(T)` or `Poll::Pending`, which is the same discrimination made explicit in the type rather than hidden in a sentinel. Both carry the completion and the suspension through one interface, and both pay the tag on every call.

The sentinel form has a hazard the tagged form does not, and it is worth naming because the narrow encoding is the tempting one here. Carving a reserved value out of W shrinks the value space to

$$|W| - 1 \;=\; 2^{64} - 1 \;=\; 18{,}446{,}744{,}073{,}709{,}551{,}615,$$

which is 18,446,744,073,709,551,615 usable values, a loss of 2^{-64} or about 5.421×10^{-20} of the space, and any program able to produce the reserved value as a legitimate yield becomes unrepresentable. **The loss is negligible in measure and total in reachability, and only the second of those matters.** A sentinel is available exactly when the yielded type is a proper subset of W missing at least one element, a statement about the type and not about how unlikely the value is. Kotlin can afford it because the sentinel is a reference no user value aliases. A backend whose yielded values are arbitrary machine words cannot, so for Keleusma the tagged pair is the honest form and the sentinel is unavailable.

The correction is left in the text rather than folded into the argument, because the sequence is the useful part. A counting argument was stated in a form stronger than the evidence supported, and reading the governing interface specification rather than assuming the register width is what caught it.

The nine-out-of-nine measurement was answering a different question than the one that mattered. It established that the **shape** is uniform. It said nothing about the **arity of the interface**, and the arity is what fails.

Why the divergent case is genuinely different

A `loop` function never completes. The reset instruction rewinds and returns control to the host, so there is no completion event at all. The count of observable events is one, the count of channels is one, and the convention fits exactly,

$$\mathrm{obs}(k) = 1 = \mathrm{chan}(\varepsilon_{\mathrm{ret}}) \qquad \text{for all } k \in K_{\Sigma} \text{ admitted below,}$$

so the required encoding is $\varepsilon_{\mathrm{ret}} : W \to W$, for which the identity suffices. The host drives the chunk by iterating the native function, and the correspondence with the virtual machine is

$$f(a) = v_0, \qquad f(r_k) = v_k \;\; (k \geq 1),$$

with **no distinguished first call**, because iteration zero consumes the call argument and iteration k consumes the k-th resume value, exactly as the virtual machine does when it writes the resume value into the entry chunk's parameter slot.

So the two conventions are not an accident awaiting cleanup. They track the boundary between a construct that terminates and one that does not, which is a distinction the language already makes, the type checker already enforces, and the memory analysis already depends on. The backend grew two conventions because there are two things.

The general form of the test

Stated without reference to compilers, the test reads as follows.

> Let a proposed unification map a class of behaviours onto an interface. Count the **distinguishable > outcomes** each behaviour can produce and count the **distinguishable outcomes the interface can carry**. > If the first exceeds the second for any member of the class, the unification is lossy, and evidence about > the **distribution** of members cannot rescue it.

The failure mode this catches is common, and it is **collecting evidence about instances when the defect lives in the interface.** Nine out of nine is a fact about instances. One slot against two events is a fact about the interface. The second dominates the first entirely, and it is cheaper to check.

A Rule Stricter Than The Property It Enforces

Gathering the distribution for this article surfaced a defect in code shipped one increment earlier, which no test had caught because no test asked.

The admissibility rule for the return convention requires that between a suspension and the reset, **nothing runs except block delimiters and a single stack-cleanup instruction**. The intent is to establish that the resume value is discarded, since the return convention does not supply one.

Ten of the twenty-four stream chunks fail this rule. All ten are the same program shape and their tail instruction sequence is the following.

```
PopN(1), Const(0), PopN(1)
```

The source is a body that suspends and then evaluates a trailing constant, so the constant is pushed and immediately discarded. **The sequence has no observable effect and leaves the stack balanced.** It is exactly as safe as the sequence the rule admits. The rule refuses it because the rule was written by enumerating the instructions the author had seen rather than by stating the property.

Write $\delta(\iota) \in \mathbb{Z}$ for the net operand-stack effect of an instruction and $\mathrm{eff}(\iota)$ for its set of side effects. For a tail segment π following a suspension, the property that actually licenses the transformation is

$$\Delta(\pi) \;=\; \sum_{\iota \in \pi} \delta(\iota) \;=\; -1 \qquad \text{and} \qquad \mathrm{eff}(\pi) \;=\; \emptyset,$$

the -1 recording that the segment consumes exactly the resume value the suspension pushed and nothing else. Both candidate tails satisfy it.

$$\Delta\big(\langle \texttt{PopN}(1) \rangle\big) = -1, \qquad \Delta\big(\langle \texttt{PopN}(1), \texttt{Const}(0), \texttt{PopN}(1) \rangle\big) = -1 + 1 - 1 = -1.$$

The shipped rule instead tests membership in a fixed instruction set $A = \{\texttt{PopN}(1)\} \cup D$, with D the block delimiters. Let P denote the set of tail segments satisfying the property and R the set the rule admits. The rule is **sound but not complete**,

$$R \;\subsetneq\; P, \qquad |P \setminus R| \;\geq\; 10,$$

and the containment is what makes the defect a coverage loss rather than a correctness loss. The property is **effect-free and stack-neutral**. The rule is **drawn from a list of two instructions**. Those coincide on the corpus the rule was written against and diverge on the next ten cases, the standard failure of a whitelist standing in for a predicate.

A sound but incomplete rule is the normal condition of static analysis and this instance is not the normal reason for it. The framework of Cousot and Cousot 1977 exists because an exact answer is often uncomputable, so an analysis deliberately accepts imprecision in exchange for soundness. **The imprecision here was not forced.** The property is decidable on straight-line bytecode by summing two integers, so nothing about the problem required an approximation, and the rule is an approximation that nobody chose. **That is a worse position than the classical one and an easier one to fix.**

The cost is measurable and is not correctness. Nothing is mislowered, because the rule refuses rather than admits. The cost is coverage. Writing R for what the rule admits and P for what the property licenses,

$$\frac{|R|}{|K_\Sigma|} = \frac{14}{24} = 58.33\%, \qquad \frac{|P|}{|K_\Sigma|} = \frac{24}{24} = 100\%,$$

so **41.67 percent of the corpus is refused the cheap convention for no reason**, and ten stream chunks that could use it currently do not. The fix is a small generalisation and it is not attempted here, because this article's subject is the convention question and a coverage improvement inside it would be a second claim wearing the first one's clothes.

It is reported because it was found by writing rather than by testing, the second time in this series that assembling a table for publication has surfaced something a green test suite did not.

Method

The instrument compiles every `.kel` source in the project's example directories, the self-hosted compiler stage sources, and the standalone compiler subproject, discarding any that fail to compile. For each resulting module it walks every chunk and classifies it.

Suspension nesting is computed with a depth counter over the block-opening instruction set, which was checked against the full block-structured instruction list and not assumed. A missed opener would report a nested suspension as a top-level one and admit a chunk the transformation is wrong for.

The tail-position walk **follows jumps** rather than scanning the instruction stream linearly. This is necessary, not fastidious. The instructions textually between a deeply nested suspension and the reset include the bodies of sibling branches, which are on different paths and never execute after that suspension. A linear scan answers a question nobody asked.

Suspension nesting is summarised by the enclosing block stack. Writing $B(y)$ for the stack of open blocks at suspension y, the distinction that matters is

$$\mathrm{join}(y) \iff \texttt{Loop} \notin B(y),$$

a suspension that is a control-flow join rather than one crossing a back edge. For the nested chunk the measurement is $\mathrm{join}(y)$ for all 19 of its suspensions, at depths $|B(y)|$ ranging over 2 to 11.

Equivalence between the two lowerings is checked by a differential oracle against the reference virtual machine, comparing **whole sequences of yielded values** rather than final results. Writing $\mathcal{Y}(\cdot)$ for the projection of a trace onto its suspension letters, the assertion is

$$\mathcal{Y}\big(\mathcal{T}^{\mathrm{vm}}(k, a, \vec{r})\big)_{1:n} = \mathcal{Y}\big(\mathcal{T}^{\mathrm{nat}}(k, a, \vec{r})\big)_{1:n}$$

for a caller-supplied bound n, over a finite sample of $\vec{r}$. A stream never produces a final result, so a comparison of final results would be vacuous for exactly the class under study, and the truncation at n is not a weakness of the test but a consequence of the trace being infinite.

The sample is finite and the claim is universally quantified over $\vec{r}$, so the oracle refutes and does not prove. The resume values in each case are chosen to differ from one another and from the argument, and in the nested case to drive different branches on successive iterations, because a $\vec{r}$ that takes one path every time exercises one return site and licenses nothing about the join.

How the literature survey was assembled

The 35 hand-selected research references were chosen because a step of the argument depends on them, and each was read. The 1,945 harvested references were not chosen that way and were not read individually. **Stating that plainly is the point of this subsection**, because a list of four thousand citations otherwise implies a reading it does not represent.

The harvest issued keyword queries against Crossref and the NASA technical report server across thirteen topic clusters, being coroutines, continuations, effect handlers, calling conventions, code generation, verified compilation, compiler testing, resource analysis, static analysis, concurrency runtimes, bytecode virtual machines, types and semantics, and a general systems cluster. Each returned record was kept only if its title carried a subject anchor, meaning a term specific to computing and not one shared with every discipline. That test discarded 5,313 of the 7,334 retrieved records. Records already cited by hand were discarded again at assembly so that no work appears twice under two anchors.

The count that survives is smaller again. The arithmetic is worth stating and not leaving to be noticed. Of the records the filter admitted, 85 were duplicates holding the same title and year under two identifiers, 2 carried a doubled word in the title where a registry had concatenated a title with a book title, 1 was untitled or undated, and 4 duplicated a hand-selected entry. 1,945 therefore reach the reference list.

The anchor filter is the step most likely to be wrong, and it was wrong twice before it was right.

The first version was inherited from an article on a different subject and tested for the presence of that subject’s vocabulary, so it rejected 2,174 compiler-science titles for containing no aircraft. That failure is loud once looked at and silent otherwise, because a filter that rejects everything reports a small corpus rather than an error.

The second version was written for this subject and **overcorrected into uselessness**. It admitted generic stems, being analysis, implementation, generation, evaluation, system, model, performance and interface, each of which occurs in every discipline that publishes. It admitted 4,305 records, and a sample of them contained rabies control, seismic depth imaging, breeding soundness examination in veterinary medicine, supercontinuum generation in photonics, transport appraisal and fibre art. **A survey listing those has surveyed nothing.** The larger count was the symptom rather than the reassurance.

The version used here requires a term that is computing-specific on its own, so an ambiguous term contributes nothing however many of them a title carries. It admits 2,021 records of the 7,334 retrieved. **The corpus is less than half the size of the one the permissive filter produced.** That reduction is the result rather than a cost of it.

What the harvested list therefore supports is a claim about **coverage**, being that the survey did not select for agreement with its conclusion, since the queries were fixed before the records were seen. What it does not support is any claim about the content of an individual harvested record beyond its title, authors, year and venue as the registry holds them.

Threats to Validity

The corpus is one project's own sources. It over-represents text processing, because the self-hosted compiler is a text processor. A terminating-coroutine population of one is not evidence about the language. It is evidence about this corpus at this date. A user may write many, in shapes not seen here.

The conclusion does not depend on the corpus. This is the important asymmetry and it is why the argument is stated the way it is. The event-counting argument is about the calling convention and holds for a population of one or one million. The measurement is reported because it is what pointed the wrong way, not because the conclusion rests on it.

The classification was written by the party proposing the unification. The instrument that produced the nine-out-of-nine figure and the plan that figure encouraged share an author. The mitigation offered is that the figure is accurate and was not the thing that failed. The reasoning built on it failed, which is the weaker position for the author and the stronger one for the reader.

No claim is made about performance. The return convention is argued to be cheaper on structural grounds, meaning no external symbol, no indirection, and no frame across the suspension. None of that has been measured. The machine-code frame sizes that would settle it are emitted only into a section of the Executable and Linkable Format, or ELF, which this development host does not produce, so the measurement is deferred and not done.

Pattern Extraction

The pattern is available wherever an interface is proposed to cover a class of behaviours.

Two implementations of one abstraction may be tracking a real boundary. Before unifying, ask what distinguishes the two implementations, and check whether the source domain distinguishes it too. Here the type system already had two categories, enforced by a rule about which may call which, and the memory analysis already depended on the distinction. The backend was not being untidy. It was reflecting something upstream. **The tidying instinct treats a difference as noise, and the check is whether anything else in the system treats it as signal.**

Evidence about instances cannot repair a defect in an interface. This is the sharper half, and the reason is a quantifier. Admissibility of a unification is a universal claim over the class,

$$\Phi \;\equiv\; \forall b \in \mathcal{B} : \; \mathrm{obs}(b) \le \mathrm{chan}(\iota),$$

whose negation is existential,

$$\neg\Phi \;\equiv\; \exists b \in \mathcal{B} : \; \mathrm{obs}(b) > \mathrm{chan}(\iota).$$

A single witness refutes Φ, and no quantity of confirming instances establishes it. The nine-out-of-nine figure has the form of a frequency $\widehat{p} = 9/9 = 1$ over observed shapes, and a frequency estimates a proportion. It does not bound a maximum, and Φ is a statement about a maximum,

$$\Phi \iff \max_{b \in \mathcal{B}} \mathrm{obs}(b) \le \mathrm{chan}(\iota).$$

The measurement is weak even taken on its own terms, and it is worth putting a number on how weak. Reading nine successes from nine trials as a sampling exercise,

the exact one-sided lower confidence bound of Clopper and Pearson 1934 on the underlying proportion at confidence $1 - \alpha$ is

$$p_{\min} \;=\; \alpha^{1/n} \;=\; 0.05^{1/9} \;=\; 0.7169,$$

so **the observation is consistent with 28.31 percent of cases failing**, and at 99 percent confidence the bound falls to 0.5995. A run of twenty-four would give 0.8827 and still leave 11.73 percent unexcluded, and licensing a claim of 0.99 at the same confidence would need

$$n \;=\; \frac{\ln \alpha}{\ln 0.99} \;=\; 299$$

consecutive successes. **Nine is not a large number and the arithmetic says so.** The case of no observed failures is the one Hanley and Lippman-Hand 1983 treat under the heading of whether nothing going wrong means everything is all right, where they give the same bound in its approximate form as a rule of three. **Their subject is clinical and the arithmetic is indifferent to that.** That is a separate and much weaker objection than the one this section rests on, because the counting argument does not need the sample to be small. **It would refute the unification from a sample of a million.**

A uniformity measurement over instances is genuinely useful for deciding what to build first, which is the subject of the previous article in this series, because that question really is about proportions. It is useless here. The two questions feel adjacent, and one of them is answerable by counting slots.

A whitelist standing in for a predicate coincides with it on the sample it was drawn from. The tail-position rule is a small instance of a large pattern. Whenever a condition is written by enumerating the cases the author has seen, it will pass every case the author has seen. The discipline is to state the property and then check that the enumeration matches it, in that order, because the reverse order produces a rule that looks correct and is merely well-sampled.

Assembling a table for an audience is a distinct verification activity. Twice now in this series, a figure gathered for publication has exposed something that a passing test suite did not. The mechanism is not mysterious. A test asks a question its author thought to ask. A table forces a row for every case, including the ones nobody thought about, and an empty or surprising cell is a question the author did not have the imagination to ask directly.

The Decision This Informs

The spike was commissioned to inform a choice, and it narrows rather than makes it.

One option is ruled out on evidence. A single return-based convention for both forms **at a one-word return type** is lossy for the terminating form, and no shape analysis rescues it.

Two options remain and both are genuinely open.

One convention with a widened return. Every coroutine entry point returns a discriminated pair, which the System V AMD64 and AAPCS64 register-pair rules make free of memory traffic on the two targets that matter. One convention for the host, one discriminator to check, and the frame property preserved. **This option exists only because the ABI documents were read**, and it was absent from the draft that preceded the reference pass. It has not been costed.

Two conventions, declared per entry point. The artefact states which convention each exported function uses, and the host dispatches accordingly. Costs a consumer-visible dis-

tinction and a metadata field. Preserves the cheap path for the twenty-three of twenty-four chunks that can use it.

One callback convention everywhere. Uniform for the host. Costs every stream chunk an external call and, more seriously, gives up the property that no frame is live during host execution, which is the property the bounded-memory analysis wants. That is not a small concession in a project whose value proposition is definitive worst-case memory.

The memory difference is the one that can be stated exactly. Let $L(\varepsilon)$ denote the set of native frames live while host code executes between suspensions. Then

$$L(\varepsilon_{\text{ret}}) = \emptyset, \qquad L(\varepsilon_{\text{cb}}) = \{\, \text{frame}(k) \,\} \cup \{\, \text{frame}(c) : c \in \text{callers}(k) \,\},$$

because the return convention has ended the call by the time the host runs and the callback convention has not. The worst-case native stack bound therefore differs by

$$\Delta M \;=\; \sum_{\phi \in L(\varepsilon_{\text{cb}})} \text{size}(\phi),$$

a quantity that is bounded and computable but **charged for the whole duration of host activity**, which the analysis does not bound. Under the return convention that term is identically zero.

The host-interaction counts also differ. Writing s for suspensions per iteration, the native call counts per iteration are

$$\tau_{\text{ret}} = 1, \qquad \tau_{\text{cb}} = 1 + s,$$

against a virtual-machine baseline of two host round-trips per iteration, since the reset instruction returns control separately from the suspension. Every stream chunk in the corpus has $s = 1$.

A fourth option this spike did not evaluate, and which is recorded so it is not mistaken for absent, is to make the delegating chunk's callee terminate. The one delegating stream chunk calls a terminating coroutine, and if that call were restructured so the suspension belonged to the stream chunk instead of the callee, the whole corpus would fit one convention. This is a change to the source program and not the backend, it is outside the backend's authority, and whether it is reasonable is a language question this article is not equipped to answer.

Delegation is a specified problem elsewhere and the specification is instructive. PEP 380 defines `yield from` so that a subgenerator's yields pass through to the outermost caller while its **return** value becomes the value of the delegating expression. That is precisely the two-channel discipline this article arrives at, imposed at the language level rather than at the interface level. The delegating construct must distinguish the callee's suspensions from the callee's completion, because they go to different places. The Keleusma delegating chunk has the same structure and currently resolves it by routing suspensions through the callback while discarding the completion, which works only because the completion is unused in this program.

The recommendation is **two conventions declared per entry point**, on the grounds that the memory property is the project's reason for existing and one chunk in twenty-four is a poor reason to give it up. This is a recommendation and not a conclusion, and the option to restructure the source has not been costed.

The Contemporary Literature

The question this article treats sits at the intersection of four literatures that rarely cite one another, being coroutine implementation, continuation-passing transformation, application binary interface design, and worst-case resource analysis. **The first three have answered the representation question repeatedly and the fourth is the reason the answer matters here.** What follows surveys each and marks where the present problem is and is not covered.

Coroutines were classified before they were implemented well

The construct is old. Conway 1963 introduced coroutines to structure a separable compiler, and the motivating example was a compiler passing tokens between phases, close to the present use. The taxonomy that still governs discussion is de Moura and Ierusalimschy 2009, which separates coroutines along three axes, **symmetric or asymmetric**, **first-class or constrained**, and **stackful or stackless**. Keleusma's two forms are both asymmetric and constrained, and the article's question is precisely whether the backend must be stackful.

That taxonomy already contains the article's finding in a different vocabulary. An asymmetric coroutine returns to its resumer, and the resumer must distinguish a suspension from a completion, which Marlin 1980 treats as a defining obligation of the mechanism rather than an implementation detail. The literature has never regarded the discriminator as optional, and it took a backend author rediscovering it from cardinality to notice.

Where the literature does not reach the present question is this. It treats coroutines as a language feature to be provided, not as an interface exposed across a foreign function boundary to a host in another language. The distinction matters because a language runtime can afford a uniform representation with a tag, while a statically bounded artefact linked into a C or Ada program is charged for every byte of that representation by an analysis that must be sound.

The contemporary work on the construct, harvested rather than recalled and listed in full below, is where the classification stopped being the interesting part and the implementation started being it.

- Grolaux and others, 2026, Async/await is an Effective Paradigm for Event Management of User Interfaces
- Castañeda and Rodríguez, 2026, Asynchronous Wait-Free Runtime Verification and Enforcement of Linearizability
- Han and others, 2026, Design and Implementation of Boss AI for 2D Action Games Using Finite State Machines and Coroutine Chaining
- Frąszczak and Frąszczak, 2026, PhishingWebCollector Async python library for automated phishing feed collection
- Magesty and Montandon, 2026, PromiseAwait A Dataset of JavaScript Migrations from Promises to Async/Await
- Williams and Elliott, 2025, Libfork Portable Continuation-Stealing With Stackless Coroutines
- Reitz and Posner, 2025, Stackless vs. Stackful Coroutines A Comparative Study for RDMA-based Asynchronous Many-Task AMT Runtimes
- Posner and others, 2025, Toward Dynamic Resource Management An Asynchronous Many-Task AMT Runtime System leveraging Dynamic Processes with PSets DPP
- Shcherbakov and Shcherbakova, 2024, Comparative analysis of coroutine functionality in modern programming languages
- Zhao and others, 2024, COPS A Coroutine-Based Priority Scheduling Framework Perceived by the Operating System

- Li and others, 2024, GastCoCo Graph Storage and Coroutine-Based Prefetch Co-Design for Dynamic Graph Processing
- Friese and others, 2024, Lamellar A Rust-based Asynchronous Tasking and PGAS Runtime for High Performance Computing
- Mykola, 2024, USING ASYNCHRONOUS PROGRAMMING IN PYTHON TO IMPROVE APPLICATION PERFORMANCE
- Castañeda and Rodríguez, 2023, Asynchronous Wait-Free Runtime Verification and Enforcement of Linearizability
- Mane Ritesh Pratap and Alpona Das, 2023, Designing a Random Password Generator Using Python Programming Language
- Gore and others, 2023, Sentence Generator for English Language using Formal Semantics
- Daiß and others, 2023, Stellar Mergers with HPX-Kokkos and SYCL Methods of using an Asynchronous Many-Task Runtime System with SYCL
- Weber and others, 2022, A closer look at process-based simulation with stackless coroutines
- Suetterlein and others, 2022, Extending an asynchronous runtime system for high throughput applications A case study
- Wang and Huang, 2022, SGPM A coroutine framework for transaction processing
- Wang and huang, 2022, Sgpm A Coroutine Scheduling Model for Wound-Wait Concurrency Control
- Holmen and others, 2021, A Heterogeneous MPI+PPL Task Scheduling Approach for Asynchronous Many-Task Runtime Systems
- Diehl and others, 2021, Performance Measurements Within Asynchronous Task-Based Runtime Systems A Double White Dwarf Merger as an Application
- Zhou and others, 2020, Design and Implementation of Coroutine Scheduling System on SW26010
- PIETERS and SCHRIJVERS, 2020, Faster coroutine pipelines A reconstruction
- Weber and Fischer, 2020, Process-Based Simulation with Stackless Coroutines
- Ataei and Manohar, 2019, AMC An Asynchronous Memory Compiler
- Wagle and others, 2019, Runtime Adaptive Task Inlining on Asynchronous Multitasking Runtime Systems
- Wang, 2019, Web Crawler Scheduler Based on Coroutine
- Corrodi and others, 2018, A semantics comparison workbench for a concurrent, asynchronous, distributed programming language
- Peterson and others, 2017, Addressing Global Data Dependencies in Heterogeneous Asynchronous Runtime Systems on GPUs
- Yoo and Kim, 2017, Coroutine based Algorithms for reducing Memory overhead in Virtual Reality
- Spivey, 2017, Faster coroutine pipelines
- DeBuhr and others, 2017, Scalable Hierarchical Multipole Methods Using an Asynchronous Many-Tasking Runtime System
- Baskaran and others, 2016, Automatic Code Generation and Data Management for an Asynchronous Task-Based Runtime
- Толстікова and others, 2016, Efficiency asynchronous application programming language Python
- Wibowo and others, 2015, Unit test code generator for lua programming language

Continuation-passing style answers the representation question, expensively

The general transformation is known and exact. Reynolds 1972 established continuation passing as a definitional technique, Strachey and Wadsworth 1974 gave it its semantics, and Appel 1992 made it a compiler architecture. Under continuation passing every suspension becomes a call to a continuation and the representation question dissolves, because there are no re-

turns to collide with.

It dissolves the question by paying for it everywhere. The transformation is global, it changes the calling convention of every function and not of coroutines alone, and the resulting closures are heap allocated unless a later analysis recovers the stack discipline. Danvy and Nielsen 2001 show defunctionalisation recovering first-order code from the higher-order form, the step that makes continuation passing implementable without a garbage collector, and it is the step a bounded-memory project would have to trust.

Delimited control is the sharper tool. Danvy and Filinski 1990 introduced `shift` and `reset`, Felleisen 1988 the prompt-based formulation, and Dybvig, Peyton Jones and Sabry 2007 a monadic framework with an implementation strategy. Flatt and others 2007 report adding delimited and composable control to a production system, the closest thing in the literature to a cost report. **A delimited continuation is exactly a coroutine frame**, and the delimiter is exactly the boundary this article's reset instruction draws.

The continuation literature harvested for this survey is listed below. It is the thinnest of the modern clusters, at 15 records from 2015 onward against 38 earlier ones, and the imbalance is itself informative. The representation question was answered in the 1970s and 1990s and the modern work has moved to the handler formulation surveyed further down.

- Todoran and Ciobanu, 2025, Metric Continuation-Passing Semantics for Multiparty Interactions
- Pyzik, 2023, Call-By-Name Is Just Call-By-Value with Delimited Control
- VANDENBROUCKE and SCHRIJVERS, 2023, Disjunctive Delimited Control
- Ikemori and others, 2023, Typed Equivalence of Labeled Effect Handlers and Labeled Delimited Control Operators
- Schuster and others, 2022, A typed continuation-passing translation for lexical effect handlers
- Ishio and Asai, 2022, Type System for Four Delimited Control Operators
- Gibbons, 2021, Continuation-Passing Style, Defunctionalization, Accumulations, and Associativity
- Avanzini and others, 2021, On continuation-passing transformations and expected cost analysis
- Komolov and others, 2020, An empirical study of multi-threading paradigms Reactive programming vs continuation-passing style
- Todoran, 2020, Metric Semantics for Concurrent Languages Designed in Continuation-Passing Style
- FORSTER and others, 2019, On the expressive power of user-defined effects Effect handlers, monadic reflection, delimited control
- Abdallah, 2018, PRISM revisited Declarative implementation of a probabilistic programming language using multi-prompt delimited control
- Todoran and Papaspyrou, 2017, Concurrency Semantics in Continuation-Passing Style
- Schöpp, 2017, Defunctionalisation as modular closure conversion
- Biernacki and others, 2015, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations

The stackless transformation is what production compilers actually ship

The state-machine transformation, in which a coroutine becomes a function over an explicit state record, is the dominant implementation strategy and it is well documented. Syme, Petricek and Lomov 2011 describe the F# asynchronous model, Bierman and others 2012 formalise the C# `async` transformation, and Elizarov and others 2021 give the design rationale for Kotlin coroutines including the suspension sentinel discussed above. Prokopec and Liu 2018 extend the treatment to coroutines with snapshots and give a cost model.

Every one of these reports the same discriminator problem and solves it the same way, by returning a tagged or sentinel value. That convergence is the strongest available evidence for the widened-return option this article identifies, and it was invisible from inside the backend.

The alternative is stackful, which the operating-systems literature has costed thoroughly. von Behren and others 2003 argue for threads with dynamically sized stacks against event-driven code, and the stack-growth machinery they describe is what a stackful coroutine needs. For a project whose memory bound must be static, that machinery is not available.

The code-generation literature this article's backend sits inside is the largest of the survey's topical clusters, at 201 contemporary records, exceeded only by the general systems cluster listed at the end. It is here because the stackless transformation is a code-generation decision before it is a language-design one.

- Pan and others, 2026, A Backend-Agnostic Compiler for Approximate Query Processing with Probabilistic Tensor Algebra
- Zhong and others, 2026, A Multi-Modal Retrieval-Augmented Framework for Compiler Backend Generation with LLMs
- Tirichine and others, 2026, A Reinforcement Learning Environment for Automatic Code Optimization in the MLIR Compiler
- Maldonado and Carrasco-Sáez, 2026, A Reusable J48-Targeting Python Backend for Scikit-Learn Workflows Differential Validation Against WEKA J48
- Zhong and others, 2026, BePilot An AI Programming Assistant for Compiler Backend Development
- Horvat and others, 2026, Comparative Evaluation of Gemini and DeepSeek for LLM-Generated Code Quality and Architectural Robustness in Backend Software Engineering
- Reddy and others, 2026, Compiler-Assisted Instruction Fusion
- Kwon and others, 2026, Compiler-Runtime Co-operative Chain of Verification for LLM-Based Code Optimization
- Biradar and others, 2026, Design and Development of an AI-Powered Code Generation System with Persistent Memory and Multi-Agent Architecture
- Wu and others, 2026, Designing quantum chemistry algorithms with just-in-time compilation
- Alladi and others, 2026, Enabling Automatic Compiler-Driven Vectorization of Transformers
- Burgholzer and others, 2026, Focus Session Paper The MQT Compiler Collection A Blueprint for a Future-Proof Quantum-Classical Compilation Framework
- Zhang and others, 2026, GeoIR-Compiler A Geospatial Intermediate Representation and Compilation Framework for Chinese Urban Spatial Question Answering
- Luo and others, 2026, GeoJSON agents a multi-agent LLM architecture for geospatial analysis-function calling vs. code generation
- Zhang and others, 2026, LEGO-compiler enhancing neural compilation through translation composability
- Fang and others, 2026, LLM-VeriOpt Verification-Guided Reinforcement Learning for LLM-Based Compiler Optimization
- M, 2026, No-Code Backend Orchestrator with Drag and Drop Architecture and AI-Assisted Contextual Code Generation
- Krogstie and others, 2026, PIP Making Andersen's Points-to Analysis Sound and Practical for Incomplete C Programs
- Dickerson and others, 2026, Practical Python FPGA Acceleration with Fast Just-In-Time Compilation and Configuration
- Zhao and others, 2026, QuaMap A Multi-Backend Benchmark Dataset for Quantum Circuit Mapping and Learning-Based Compiler Evaluation
- Lancellotti and others, 2026, Quantum Oracle Synthesis from HDL Designs via Multi

Level Intermediate Representation
- Baradaran and others, 2026, Reusing Legacy Code in Wasm Key Challenges of Compilation and Code Semantics Preservation
- de Ferrière and others, 2026, SecSwift, a Compiler-Based Framework for Software Countermeasures in Cybersecurity
- Rinard, 2026, Testing, Credible Compilation, and Verification in the Axon Verified Compiler in Lean and Claude Code
- Qian and others, 2026, Thinking Fast and Correct Automated Rewriting of Numerical Code through Compiler Augmentation
- Ko and Heo, 2026, TinyGen Portable and Compact Code Generation for Tiny Machine Learning
- Cheng and Wu, 2026, Toward Unified Chinese Multi-Dialectal Speech Recognition via Pinyin Intermediate Representation
- Zhong and others, 2026, Towards Fully Automated Compiler Backend Generation with Multi-Agent Systems How Far Are We?
- Schwarz and others, 2026, TPDE A Fast Adaptable Compiler Back-End Framework
- Yi and others, 2026, Understanding and Finding JIT Compiler Performance Bugs
- Li and others, 2026, Unleashing Triton on CPUs Compilation and Runtime Co-Optimization for Scalable Vector Architectures
- Kang and others, 2026, WAMI Compilation to WebAssembly through MLIR without Losing Abstraction
- Maia and others, 2026, Why Just-In-Time Compilation Matters Evaluating Runtime and Energy Efficiency
- Zhu and Xie, 2025, A Domain-Specific Compiler for Embedded DSP Development Based on Component Code Generation Architecture and Correctness
- Lopoukhine and others, 2025, A Multi-level Compiler Backend for Accelerated Microkernels Targeting RISC-V ISA Extensions
- Sun and others, 2025, Archs A WebAssembly Runtime for Cross-host Heterogeneous Computing in Serverless
- Sun and others, 2025, Automating Target Description Processing for Efficient Compiler Backend Development
- Lee and others, 2025, Bit-Level Semantics Scalable RAG Retrieval with Neurosymbolic Hyperdimensional Computing
- Tamura and others, 2025, Bringing Together Cross-ISA Checkpoint/Restoration and AOT Compilation of WebAssembly Programs
- Meier and others, 2025, CertiCoq-Wasm A Verified WebAssembly Backend for CertiCoq
- Bryant and others, 2025, Certified Knowledge Compilation with Application to Formally Verified Model Counting
- Hammer and others, 2025, Compiler-Like Code Generation for fUML Reducing Overhead in Executable UML
- Kocourek and others, 2025, Copy-and-Patch Just-in-Time Compiler for R
- Ali and others, 2025, Does Coding Style Really Survive Compilation? Stylometry of Executable Code Revisited
- Kaynaroglu and others, 2025, EUTROPY A Python-based software optimized with Just-In-Time compilation for simulating eutrophication dynamics in aquatic systems
- Ding and others, 2025, HFF-JIT A Hybrid Fuzzing Framework for JIT Compiler Vulnerability Detection in JavaScript
- Lin and others, 2025, Intermediate Representation-Based Approach for Code Refactoring and Quality Evaluation
- Rong and others, 2025, IRFuzzer Specialized Fuzzing for LLVM Backend Code Generation
- Souha and others, 2025, Modeling and Automated Code Generation of the Backend of Tourism Applications
- Ravedutti Lucio Machado and others, 2025, P4IRS An intermediate representation and

compiler for parallel and performance-portable particle simulations
- Ramdani and others, 2025, Pengembangan Backend Aplikasi Geoproperty dengan Golang di PT. Nerdvana Solusi Teknologi
- Georgakoudis and others, 2025, Proteus Portable Runtime Optimization of GPU Kernel Execution with Just-in-Time Compilation
- Cieszewski, 2025, PyHLS Intermediate Representation for Versatile High-Level Synthesis
- Pavlovskiy and Platov, 2025, RESEARCH ON CODE GENERATION OF C/C++ COMPILER FOR HIGH-PERFORMANCE COMPUTING IN MICROPROCESSOR SYSTEMS
- Fruehwirth, 2025, Runtime Repeated Recursion Unfolding in CHR A Just-In-Time Online Program Optimization Strategy That Can Achieve Super-Linear Speedup
- Patel and others, 2025, SySTeC A Symmetric Sparse Tensor Compiler
- Gaißert and others, 2025, Tracing Just-in-Time Compilation for Effects and Handlers
- Berger and others, 2025, Translation Validation for LLVM's AArch64 Backend
- Shaikhha and others, 2024, A Tensor Algebra Compiler for Sparse Differentiation
- Choi and others, 2024, Accelerating Sensor Software Performance on Edge Devices Through Just-in-Time Compilation
- Robin and Khan, 2024, An open-source P416 compiler backend for reconfigurable match-action table switches Making networking innovation accessible
- Osborne and others, 2024, Awkward Just-In-Time JIT Compilation A Developer's Experience
- Raju Cherukuri, 2024, Building Scalable Web Applications Best Practices for Backend Architecture
- Zhong and others, 2024, ComBack A Versatile Dataset for Enhancing Compiler Backend Development Efficiency
- Engelke and Schwarz, 2024, Compile-Time Analysis of Compiler Frameworks for Query Compilation
- Geeson and Smith, 2024, Compiler Testing with Relaxed Memory Models
- Han and others, 2024, Enabling Fine-Grained Incremental Builds by Making Compiler Stateful
- Jesus and Weiland, 2024, Evaluating and optimising compiler code generation for NVIDIA Grace
- Drescher and Engelke, 2024, Fast Template-Based Code Generation for MLIR
- Mukherjee and others, 2024, HLS-IRT Hardware Trojan Insertion through Modification of Intermediate Representation During High-Level Synthesis
- Nejjar and others, 2024, LLMs for science Usage for code generation and data analysis
- Jung, 2024, Miri Practical Undefined Behavior Detection for Rust Keynote
- GREEN and WOOD, 2024, REASONING ABOUT THE ACOUSTIC REALISATION OF SEMIVOWELS USING AN INTERMEDIATE REPRESENTATION - THE 'SPEECH SKETCH'
- Duhamel and Pillement, 2024, Runtime Task Scheduling for FPGA-Based Embedded Systems Using Just-in-Time Bitstream Prefetching
- Pham and Odersky, 2024, Stack-Copying Delimited Continuations for Scala Native
- Ramesh and others, 2024, ThriveJIT Dynamic Just-In-Time Compilation for Efficient Execution of Arithmetic Expressions
- Kwon and others, 2024, Translation Validation for JIT Compiler in the V8 JavaScript Engine
- Bauckholt and Holz, 2024, WebAssembly as a Fuzzing Compilation Target Registered Report
- Titzer, 2024, Whose Baseline Compiler is it Anyway?
- Телегин, 2023, AHEAD-OF-TIME and JUST-IN-TIME technologies
- Bhamidipati and Vemuri, 2023, ASPIRE An Intermediate Representation for Abstract Security Policies
- Lim and Debray, 2023, Automatically Localizing Dynamic Code Generation Bugs in JIT

Compiler Back-End
- Blazy, 2023, CompCert A Journey through the Landscape of Mechanized Semantics for Verified Compilation Keynote
- Prinz, 2023, Compilation of Distributed Programs to Services Using Multiple Programming Languages
- Abdelmaksoud and others, 2023, DEL Dynamic Symbolic Execution-based Lifter for Enhanced Low-Level Intermediate Representation
- Shahrokhi and others, 2023, Efficient Query Processing in Python Using Compilation
- Barrière and others, 2023, Formally Verified Native Code Generation in an Effectful JIT Turning the CompCert Backend into a Formally Verified JIT Compiler
- Groß and others, 2023, FUZZILLI Fuzzing for JavaScript JIT Compiler Vulnerabilities
- 2023, Green Supply Chain Management and Competitive Advantage Evidence of Just-in-time Management on Firm Performance SMEs in Indonesia
- Pichler and others, 2023, Hybrid Execution Combining Ahead-of-Time and Just-in-Time Compilation
- Gourdin, 2023, Lazy Code Transformations in a Formally Verified Compiler
- Thangamani and others, 2023, Lifting Code Generation of Cardiac Physiology Simulation to Novel Compiler Technology
- Gu, 2023, LLM-Based Code Generation Method for Golang Compiler Testing
- He and others, 2023, Profile Guided Optimization Transfer-Learning for OpenCL/SYCL Kernel Compilation and Runtime
- Nedoria, 2023, Programming Language for Teaching Compilation and Transformation Technologies
- Wu, 2023, PyTorch 2.0 The Journey to Bringing Compiler Technologies to the Core of PyTorch Keynote
- Pečimúth, 2023, Remote Just-in-Time Compilation for Dynamic Languages
- Moron and Wallentowitz, 2023, Support for Just-in-Time Compilation of WebAssembly for Embedded Systems
- Lopes, 2023, Torchy A Tracing JIT Compiler for PyTorch
- Rivera and others, 2022, A Compiler for Sound Floating-Point Computations using Affine Arithmetic
- Huang and others, 2022, A High-Performance Bidirectional Compiler for Conversion Between SystemC and Verilog
- Stepanov and Itsykson, 2022, Backend Bug Finder a platform for effective compiler fuzzing
- Schmale and others, 2022, Backend compiler phases for trapped-ion quantum computers
- Jain and others, 2022, Coarse Grained FPGA Overlay for Rapid Just-In-Time Accelerator Compilation
- Akanbi and others, 2022, Code Generation Techniques in Compiler Design Conceptual and Structural Review
- Nguyen and McCaskey, 2022, Extending Python for Quantum-classical Computing via Quantum Just-in-time Compilation
- Polito and others, 2022, Interpreter-guided differential JIT compiler unit testing
- Hartley and others, 2022, Just-In-Time Compilation on ARM-A Closer Look at Call-Site Code Consistency
- Katel and others, 2022, MLIR-based code generation for GPU tensor cores
- Serrano, 2022, On JavaScript Ahead-of-Time Compilation Performance Keynote
- Ji and Wang, 2022, Optimizing Aggregate Computation of Graph Neural Networks with on-GPU Interpreter-Style Programming
- Efthymiou and others, 2022, Quantum simulation with just-in-time compilation
- 2022, Supplemental Material for A Just-in-Time Adaptive Intervention to Enhance Physical Activity in the SMARTFAMILY2.0 Trial
- Izawa and others, 2022, Threaded Code Generation with a Meta-Tracing JIT Compiler
- Li and others, 2022, Unleashing the power of compiler intermediate representation to

enhance neural program embeddings
- Mzid and others, 2022, Use of Compiler Intermediate Representation for Reverse Engineering A Case Study for GCC Compiler and UML Activity Diagram
- Ortiz, 2022, Using WebAssembly to Teach Code Generation in a Compiler Design Course
- Rand, 2022, Writing and verifying a Quantum optimizing compiler keynote
- Quiring and others, 2021, 3CPS The Design of an Environment-Focussed Intermediate Representation
- Rivera and others, 2021, An Interval Compiler for Sound Floating-Point Computations
- Papadimitriou and others, 2021, Automatically exploiting the memory hierarchy of GPUs through just-in-time compilation
- Jamieson and Brown, 2021, Compact native code generation for dynamic languages on micro-core architectures
- Xu and Kjolstad, 2021, Copy-and-patch compilation a fast compilation algorithm for high-level languages and bytecode
- Barrière and others, 2021, Formally verified speculation and deoptimization in a JIT compiler
- Demmler and others, 2021, Improved Circuit Compilation for Hybrid MPC via Compiler Intermediate Representation
- Song and Wang, 2021, Monadic Programming Featured Teaching Innovation of Compilation Experiments
- Serrano, 2021, Of JavaScript AOT compilation performance
- Liu and others, 2021, Relaxed Peephole Optimization A Novel Compiler Optimization for Quantum Circuits
- Koehler and Steuwer, 2021, Towards a Domain-Extensible Compiler Optimizing an Image Processing Pipeline on Mobile CPUs
- Chhak and others, 2021, Towards formally verified compilation of tag-based policy enforcement
- Sambasivam and others, 2021, Writing P4 compiler backend for packet processing engines
- Loveless and others, 2020, A performance-optimizing compiler for cyber-physical digital microfluidic biochips
- Saieva and Kaiser, 2020, Binary Quilting to Generate Patched Executables without Compilation
- Felker, 2020, Design of Cyanobyte An Intermediate Representation to Standardize Digital Peripheral Datasheets for Automatic Code Generation
- Blazy, 2020, From Verified Compilation to Secure Compilation a Semantic Approach
- Paul and others, 2020, Improving execution efficiency of just-in-time compilation based query processing on GPUs
- Brennan and others, 2020, JIT Leaks Inducing Timing Side Channels through Just-In-Time Compilation
- Schuiki and others, 2020, LLHD a multi-level intermediate representation for hardware description languages
- Dakkak and others, 2020, The design and implementation of the wolfram language compiler
- Scheidl, 2020, Valent-Blocks Scalable High-Performance Compilation of WebAssembly Bytecode For Embedded Systems
- Liu and others, 2019, Accelerating sequential consistency for Java with speculative compilation
- Finkel and others, 2019, ClangJIT Enhancing C++ with Just-in-Time Compilation
- Hariri and others, 2019, Comparing Mutation Testing at the Levels of Source Code and Compiler Intermediate Representation
- Namakonov and Podkopaev, 2019, Compilation of OCaml memory model into Power
- Enrici and others, 2019, Efficient Data-Flow Analysis of UML/SysML Diagrams for Optimized Model Compilation of Hardware-software Systems

- Nappa and others, 2019, Fast Parallel Equivalence Relations in a Datalog Compiler
- HATABA and others, 2019, Generation of Efficient Obfuscated Code through Just-in-Time Compilation
- Müssig, 2019, Just enough, just in time, just for me
- Schkufza and others, 2019, Just-In-Time Compilation for Verilog
- Bourke and others, 2019, Mechanized semantics and verified compilation for a dataflow synchronous language with reset
- Castro-Lopez and Vega-Lopez, 2019, Multi-target Compiler for the Deployment of Machine Learning Models
- Tine and others, 2019, POSTER Tango An Optimizing Compiler for Just-In-Time RTL Simulation
- Zhang and others, 2019, SNC A Cloud Service Platform for Symbolic-Numeric Computation Using Just-In-Time Compilation
- KIAM TAN and others, 2019, The verified CakeML compiler backend
- Baghdadi and others, 2019, Tiramisu A Polyhedral Compiler for Expressing Fast and Portable Code
- Turcotte and Vitek, 2019, Towards a Type System for R
- Tinnerholm and others, 2019, Towards introducing just-in-time compilation in a Modelica compiler
- Curtis and others, 2018, A compiler for cyber-physical digital microfluidic biochips
- Vandercammen and others, 2018, A flexible framework for studying trace-based just-in-time compilation
- Leopoldseder and others, 2018, Dominance-based duplication simulation DBDS code duplication to enable compiler optimizations
- Caamaño and Guelton, 2018, Easy Jit compiler assisted library to enable just-in-time compilation in C++ codes
- Ottoni, 2018, HHVM JIT a profile-guided, region-based compiler for PHP and Hack
- ., 2018, Just in time and competitive advantage understanding their linkages and impact on operational performance
- Brock and others, 2018, PAYJIT space-optimal JIT compilation and its practical implementation
- Caliskan and others, 2018, When Coding Style Survives Compilation De-anonymizing Programmers from Executable Binaries
- Pape and others, 2017, Adaptive just-in-time value class optimization for lowering memory consumption and improving execution time performance
- Su and others, 2017, Automatic generation of fast BLAS3-GEMM A portable compiler approach
- Mainland, 2017, Better living through operational semantics an optimizing compiler for radio protocols
- Reis and others, 2017, Compiler Techniques for Efficient MATLAB to OpenCL Code Generation
- Basu and others, 2017, Compiler-based code generation and autotuning for geometric multigrid on GPU-accelerated supercomputers
- Rohr and Lindenstruth, 2017, Fast Failure Erasure Encoding Using Just in Time Compilation for CPUs, GPUs, and FPGAs
- Welch and others, 2017, Formalization IDEs Integrated with a Verifying Compiler
- HAMID and ITO, 2017, Image Segmentation to Design Semi-optimized Curve for B-code Generation
- Fumero and others, 2017, Just-In-Time GPU Compilation for Interpreted Languages with Partial Evaluation
- Sharygin and Buchatskiy, 2017, Survey of Just-in-Time Query Compilation Methods
- Fox and others, 2017, Verified compilation of CakeML to multiple machine-code targets
- Zhang and others, 2017, Weak Memory Models Balancing Definitional Simplicity and Implementation Flexibility

- Spampinato and Püschel, 2016, A basic linear algebra compiler for structured matrices
- Tan and others, 2016, A new verified compiler backend for CakeML
- Dissegna and others, 2016, An Abstract Interpretation-Based Model of Tracing Just-in-Time Compilation
- Bulej and others, 2016, Beneath the bytecode
- Martinsen and others, 2016, Combining thread-level speculation and just-in-time compilation in Google's V8 JavaScript engine
- Kwon and Bae, 2016, Development of a Code Generation Support System in Integrated Development Environment of an Educational Compiler
- Suhendra and Bachtiar, 2016, MIGRATION CODE PADA BACKEND CRIMEZONE DARI PHP KE SCALA
- Li and others, 2016, Modular SDN Compiler Design with Intermediate Representation
- Béra and others, 2016, Practical Validation of Bytecode to Bytecode JIT Compiler Dynamic Deoptimization
- Moss and others, 2016, The ARES High-Level Intermediate Representation
- Alexander and Black, 2016, The performance of object encodings in JavaScript
- Plangger and Krall, 2016, Vectorization in PyPy's Tracing Just-In-Time Compiler
- Xu and Gregg, 2015, An Efficient Vectorization Approach to Nested Thread-level Parallelism for CUDA GPUs
- Oh and others, 2015, Bytecode-to-C Ahead-of-Time Compilation for Android Dalvik Virtual Machine
- Refaie and Thyabat, 2015, Effect of just-in-time selling strategy on firms' performance in Jordan
- Lee and others, 2015, Flow-sensitive runtime estimation an enhanced hot spot detection heuristics for embedded Java just-in-time compilers
- Melrose and others, 2015, Just in Time
- Khaldi and others, 2015, LLVM parallel intermediate representation
- Hollingshaus and Daddario, 2015, Performance Philosophy Arrived Just in Time?
- Heck and Zaidman, 2015, Quality criteria for just-in-time requirements just enough, just-in-time?
- Khorasani and others, 2015, Scalable SIMD-Efficient Graph Processing on GPUs
- Heumann and others, 2015, Scalable Task Scheduling and Synchronization Using Hierarchical Effects
- Meybodi, 2015, The links between just-in-time practices and alignment of benchmarking performance measures
- Santhi and others, 2015, The Simian concept Parallel Discrete Event Simulation with interpreted languages and just-in-time compilation

Effect handlers are the modern generalisation, and they carry the same obligation

Algebraic effects subsume coroutines. Plotkin and Pretnar 2009 give the handler formulation, Leijen 2017 a type-directed compilation, Hillerström and Lindley 2016 a row-typed treatment, and Sivaramakrishnan and others 2021 a production retrofit onto OCaml with performance figures.

The relevance here is narrow and worth stating precisely. **An effect handler's return clause and its operation clauses are syntactically distinct**, which is the same discriminator appearing a third time, now in the type system rather than the calling convention. A language that adopted handlers would not face this article's question, because the distinction would already be in the source. Keleusma does not adopt them, and the V0.4 architecture does not propose to.

- Segura, 2026, Algebraic effects for bounded prompt pipelines Lawvere theories, handlers, and outcome traces

- Gillard and others, 2026, Dynamic Wind for OCaml Effect Handlers with Escaping Continuation Support
- Trifanov and Schrijvers, 2026, Staging Effect Handlers for Modular Search
- Gao and Parreaux, 2025, A Lightweight Type-and-Effect System for Invalidation Safety Tracking Permanent and Temporary Invalidation with Constraint-Based Subtype Inference
- van Rooij and Krebbers, 2025, Affect An Affine Type and Effect System
- Asai and Fujii, 2025, Defining Algebraic Effects and Handlers via Trails and Metacontinuations
- Voigt and others, 2025, Dynamic Wind for Effect Handlers
- Ma, 2025, Lexical Effect Handlers Fast by Design, Correct by Proof
- Jaafar and Jaber, 2025, Operational Game Semantics for Generative Algebraic Effects and Handlers
- Ma and others, 2025, Zero-Overhead Lexical Effect Handlers
- Yoshioka and others, 2024, Abstracting Effect Systems for Algebraic Effect Handlers
- Mückenschnabel, 2024, Algebraic Effect Handlers with Bidirectional Type-Checking
- SANADA, 2024, Algebraic effects and handlers for arrows
- Tsuyama and others, 2024, An Intrinsically Typed Compiler for Algebraic Effect Handlers
- Kawamata and others, 2024, Answer Refinement Modification Refinement Type System for Algebraic Effects and Handlers
- HILLERSTRÖM and others, 2024, Asymptotic speedup via effect handlers
- Ma and others, 2024, Lexical Effect Handlers, Directly
- Xie and others, 2024, Parallel Algebraic Effect Handlers
- Isoda and others, 2024, Type-Safe Code Generation with Algebraic Effects and Handlers
- Sanada, 2023, Category-Graded Algebraic Theories and Effect Handlers
- Nguyen and others, 2023, Effect Handlers for Programmable Inference
- Müller and others, 2023, From Capabilities to Regions Enabling Efficient Compilation of Lexical Effect Handlers
- New and others, 2023, Gradual Typing for Effect Handlers
- Sekiyama and Unno, 2023, Temporal Verification with Answer-Effect Modification Dependent Temporal Type-and-Effect System with Delimited Continuations
- Cong and Asai, 2023, Towards a Reflection for Effect Handlers
- Xie and others, 2022, First-class names for effect handlers
- Ghica and others, 2022, High-level effect handlers in C++
- Hamana, 2022, Modular Termination for Second-Order Computation Rules and Application to Algebraic Effect Handlers
- de Vilhena and Pottier, 2021, A separation logic for effect handlers
- Zyuzin and Nanevski, 2021, Contextual modal types for algebraic effects and handlers
- Karachalias and others, 2021, Efficient compilation of algebraic effect handlers
- Xie and Leijen, 2021, Generalized evidence passing for effect handlers efficient compilation of effect handlers to C
- Noguchi and others, 2021, Implementing Algebraic Effects and Handlers in Non-functional Programming Languages
- Punchihewa and Wu, 2021, Safe mutation with algebraic effects
- Liu and others, 2020, A type-and-effect system for object initialization
- Schuster and others, 2020, Compiling effect handlers in capability-passing style
- Xie and Leijen, 2020, Effect handlers in Haskell, evidently
- HILLERSTRÖM and others, 2020, Effect handlers via generalised continuations
- Xie and others, 2020, Effect handlers, evidently
- Brachthäuser and others, 2020, Effects as capabilities effect handlers and lightweight effect polymorphism
- BRACHTHÄUSER and others, 2020, Effekt Capability-passing style for type- and effect-safe, extensible effect handlers in Scala
- Zhang and Myers, 2019, Abstraction-safe effect handlers via tunneling

- Brachthäuser and others, 2018, Effect handlers for the masses
- Leijen, 2018, First class dynamic effect handlers or, polymorphic heaps with dynamic effect handlers
- Biernacki and others, 2017, Handle with care relational interpretation of algebraic effects and handlers
- KAMMAR and PRETNAR, 2017, No value restriction is needed for algebraic effects and handlers
- Leijen, 2017, Structured asynchrony with algebraic effects
- SALEH and SCHRIJVERS, 2016, Efficient algebraic effect handlers for Prolog
- Pretnar, 2015, An Introduction to Algebraic Effects and Handlers. Invited tutorial paper
- Bauer and Pretnar, 2015, Programming with algebraic effects and handlers

Verified compilation constrains what the backend may do

The project's stance is that the backend refuses what it cannot lower rather than approximating it, and that stance has a literature. Leroy 2009 reports CompCert and its formally verified back end in Leroy 2009b, and Kumar and others 2014 report CakeML, a verified implementation of ML with a verified compiler. Necula 1997 formalises proof-carrying code, in which the artefact carries its own evidence, and Sewell and others 2013 report translation validation for a verified operating-system kernel, which is the technique a project unable to verify its compiler can still use.

None of these treats the calling convention as a proof obligation, and that is the gap the present project sits in. CompCert's correctness theorem is about observable behaviour of whole programs under a fixed calling convention, and it does not tell you which convention to choose when a source construct emits two observable event kinds. The theorem shape assumed here, that a lowering is correct when yielded sequences agree, is a coinductive statement over infinite traces of the kind Leroy and Grall 2009 formalise for big-step semantics of diverging programs. **That paper is the right formal setting for the divergent case and this article does not use it as such**, which is a real weakness rather than an omission of politeness.

- Lasnier and others, 2026, Brack A Verified Compiler for Scheme via CakeML
- Lion and Broman, 2026, Making Time Observable Compiler Correctness for Real-Time C Programs
- Nezamabadi and others, 2026, Verified VCG and Verified Compiler for Dafny
- Barbosa and others, 2025, Formally Verified Correctness Bounds for Lattice-Based Cryptography
- Kwon and others, 2025, Optimization-Directed Compiler Fuzzing for Continuous Translation Validation
- Girault, 2025, Toward a Formally Verified Compiler for a Synchronous, Functional, Data-Flow Programming Language
- Melquiond and Moreau, 2024, A Safe Low-Level Language for Computer Algebra and Its Formally Verified Compiler
- Liu and others, 2024, A Verified Compiler for a Functional Tensor Language
- Debnath and others, 2024, ARMOR A Formally Verified Implementation of X.509 Certificate Chain Validation
- Wang and Xie, 2024, Enhancing Translation Validation of Compiler Transformations with Large Language Models
- Monniaux, 2024, Memory Simulations, Security and Optimization in a Verified Compiler
- Chavanon and others, 2024, PfComp A Verified Compiler for Packet Filtering Leveraging Binary Decision Diagrams
- Kanabar and others, 2023, PureCake A Verified Compiler for a Lazy Functional Language
- Derakhshan and others, 2023, Towards End-to-End Verified TEEs via Verified Interface Conformance and Certified Compilers

- Panigrahi and Karfa, 2023, Translation Validation of Information Leakage of Compiler Optimizations
- Nyangaresi and Ma, 2022, A Formally Verified Message Validation Protocol for Intelligent IoT E-Health Systems
- Abdulaziz and Koller, 2022, Formal Semantics and Formally Verified Validation for Temporal Planning
- Han and others, 2021, Parallelizing Compiler Translation Validation Using Happens-Before and Task-Set
- Mittal and others, 2021, Towards an Approach for Translation Validation of Thread-level Parallelizing Transformations using Colored Petri Nets
- Roessle and others, 2019, Formally verified big step semantics out of x86-64 binaries
- Tahat and others, 2019, Scalable Translation Validation of Unverified Legacy OS Code
- Patterson and Ahmed, 2019, The next 700 compiler correctness theorems functional pearl
- Banerjee and Karfa, 2018, Compiler-agnostic Translation Validation
- Lochbihler, 2018, Mechanising a Type-Safe Model of Multithreaded Java with a Verified Compiler
- Bourke and others, 2017, A formally verified compiler for Lustre
- Avigad and others, 2017, A Formally Verified Proof of the Central Limit Theorem
- Fonseca and others, 2017, An Empirical Study on the Correctness of Formally Verified Distributed Systems
- Barthe and others, 2017, Verified Translation Validation of Static Analyses
- Tan and others, 2015, A verified type system for CakeML
- Ngo and others, 2015, Modular translation validation of a full-sized synchronous compiler using off-the-shelf verification tools
- Neis and others, 2015, Pilsner a compositionally verified compiler for a higher-order imperative language
- Chlipala, 2015, Session details Session 4A Compiler Correctness
- Yang and others, 2015, Towards a verified compiler prototype for the synchronous language SIGNAL

Worst-case resource analysis is why the convention matters at all

If native code did not have to carry a bound, the callback convention would simply win on generality. Wilhelm and others 2008 survey the worst-case execution-time problem and the tools that address it, and it remains the standard reference for what a sound timing analysis requires. For the memory side, Regehr, Reid and Webb 2005 eliminate stack overflow by abstract interpretation of machine code, exactly the analysis a native Keleusma artefact must support and exactly what a frame held across arbitrary host execution defeats.

This is the literature that makes the choice non-obvious, and it is the one the coroutine implementation literature does not cite. A runtime that may allocate is free to choose the general mechanism. A bounded artefact is not, and the trade appears only when both literatures are read together.

- Charmanas and others, 2026, A topic-oriented trend analysis framework for Stack Exchange questions Case study on ChatGPT related queries on Stack Overflow
- O Kim and Lee, 2026, Amortised neural acoustic tomography domain-specific encoder design and topology-dependent Eikonal analysis
- Seidler and others, 2026, Wasm-WCET Worst-Case Execution-Time Analysis of WebAssembly Modules on Updatable Resource-Constrained Embedded Devices
- Zolduoarrati and others, 2025, A cross-continental analysis of how regional cues shape top stack overflow contributors
- Kaestner and others, 2025, Determining Worst-Case Execution Time Bounds for Multi-Core Processors

- Lehmann and others, 2025, Hardware/Software Co-Analysis for Worst Case Execution Time Bounds
- Djalali and others, 2025, The Evolution of Software Usability in Developer Communities An Empirical Study on Stack Overflow
- Wambua, 2025, Topics, Trends, and Sentiments in Software Testing An Analysis of Developers' Engagement on Stack Overflow
- Merazga and others, 2025, Worst-Case Execution Time Analysis of a Real-Time System based on Arduino in CAN Network
- Hu and others, 2024, An Abstract Interpretation-Based Framework for WCET Analysis of Parallel Programs
- Thomas and others, 2024, Analyzing Data Flow and Control Flow of Multicore Software A Solution for Efficient Worst-Case Execution Time Analysis
- Arnström and others, 2024, Exact Worst-Case Execution-Time Analysis for Implicit Model Predictive Control
- Veit and Böcskei, 2024, IFRS 9 Classification Aspects Measurement of Sustainability-Linked Loans at Amortised Cost or Fair Value
- Bertholon and others, 2024, Interactive Source-to-Source Optimizations Validated using Static Resource Analysis
- Mizikovskiy, 2024, Methodology of management cost accounting and calculation of the cost of not amortised organizational and technological equipment for the production of industrial enterprise products
- Albert and others, 2024, Synthesis of Sound and Precise Storage Cost Bounds via Unsound Resource Analysis and Max-SMT
- Kumar and others, 2024, Utilizing Machine Learning Techniques for Worst-Case Execution Time Estimation on GPU Architectures
- Ghadesi and others, 2024, What causes exceptions in machine learning applications? Mining machine learning-related stack traces on Stack Overflow
- Samiei and others, 2024, Worst-Case Execution Time Analysis of Real-Time Robotic Algorithms Using Reinforcement Learning
- Paraman and Murthy, 2023, Analysis of benchmark program results of worst case execution time for multithreaded programs
- Mondal and others, 2023, Investigating Technology Usage Span by Analyzing Users' QandA Traces in Stack Overflow
- Swillus and Zaidman, 2023, Sentiment overflow in the testing stack Analyzing software testing posts on Stack Overflow
- Rodriguez Ferrandez and others, 2023, Worst Case Execution Time and Power Estimation of Multicore and GPU Software A Pedestrian Detection Use Case
- ISLAM and others, 2022, An Exploration of npm Package Co-Usage Examples from Stack Overflow A Case Study
- WanXin and others, 2022, CUDA Acceleration of Worst-Case Execution Time Analysis Based On Model Checking
- Battle and others, 2022, Exploring D3 Implementation Challenges on Stack Overflow
- Sheth and Damevski, 2022, Grouping related stack overflow comments for software developer recommendation
- Mahajan and Prasad, 2022, Providing Real-time Assistance for Repairing Runtime Exceptions using Stack Overflow Posts
- Velazquez-Rodriguez and others, 2022, Uncovering Library Features from API Usage on Stack Overflow
- Susca and others, 2022, Worst-Case Execution Time Estimation for Numerical Controllers
- Friers and others, 2021, Amortised Encoding for Large High-Resolution Displays
- van der Hoeven and Lecerf, 2021, Amortized Bivariate Multi-point Evaluation
- Kumar, 2021, Deep Neural Network Approach to Estimate Early Worst-Case Execution Time

- Denzler and others, 2021, Experiences from Adjusting Industrial Software for Worst-Case Execution Time Analysis
- Zhao and others, 2021, Hot question prediction in Stack Overflow
- Costa and others, 2021, Use of Measurements in Worst-Case Execution Time Estimation for Real-Time Systems
- Búr and others, 2021, Worst-case Execution Time Calculation for Query-based Monitors by Witness Generation
- Zhang and others, 2020, A Dynamic Instruction Cache Locking Approach for Minimizing Worst Case Execution Time of a Single Task
- Emerson, 2020, Autoencoding Pixies Amortised Variational Inference with Graph Convolutions for Functional Distributional Semantics
- Moser and Schneckenreither, 2020, Automated amortised resource analysis for term rewrite systems
- Muts and Falk, 2020, Compiler-based WCET prediction performing function specialization
- Nadi and Treude, 2020, Essential Sentences for Navigating Stack Overflow Answers
- Uddin and others, 2020, Mining API usage scenarios from stack overflow
- Fusi and others, 2020, On the Use of Probabilistic Worst-Case Execution Time Estimation for Parallel Applications in High Performance Systems
- Azzahra and others, 2020, PRE STACK DEPTH MIGRATION UNTUK KOREKSI EFEK PULL UP DENGAN MENGGUNAKAN METODE HORIZON BASED DEPTH TOMOGRAPHY PADA LAPANGAN 'A1 DAN A2'
- Mahajan and others, 2020, Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching
- Arcaro and others, 2020, Reliability Test based on a Binomial Experiment for Probabilistic Worst-Case Execution Times
- Meng and others, 2020, Survey on Estimation and Optimization of Worst-case Execution Time with Energy Consumption Constraint
- Ventovaara and others, 2020, Worst-case Execution Time Estimation of Legacy Vehicular Embedded Functions An Industrial Case Study
- Falk and Lokuciejewski, 2019, Correction to A compiler framework for the reduction of worst-case execution times
- Zuepke and Kaiser, 2019, Deterministic Futexes Addressing WCET and Bounded Interference Concerns
- Yildiz and others, 2019, Software UART A Use Case for VSCPU Worst-Case Execution Time Analyzer
- Baltes and others, 2019, SOTorrent Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets
- Kim and others, 2019, WCET-Aware Stack Frame Management of Embedded Systems Using Scratchpad Memories
- Zhang and Deng, 2018, A Depth Variant Seismic Wavelets Extraction Method for Inversion of Post-Stack Depth Domain Seismic Data
- Reinhardt and others, 2018, Augmenting stack overflow with API usage patterns mined from GitHub
- Szydełko, 2018, BONDS BALANCE SHEET VALUATION IN AMORTISED COST CHOSEN ASPECTS
- Wu and Zhang, 2018, Cache-Aware SPM Allocation to Reduce Worst-Case Execution Time for Hybrid SPM-Caches
- Huangfu and Zhang, 2018, Estimating the Worst-Case Execution Time of the Shared Data Cache in Integrated CPU-GPU Architectures
- Carminati and others, 2018, On the use of static branch prediction to reduce the worst-case execution time of real-time applications
- Becker and Chakraborty, 2018, Optimizing Worst-Case Execution Times Using Mainstream Compilers

- Venkanna and others, 2018, PSO based optimization of worst-case execution time for ASIP application
- Becker and others, 2018, Scalable and precise estimation and debugging of the worst-case execution time for analysis-friendly processors a comeback of model checking
- Venkanna and Rao, 2018, Static Worst-Case Execution Time Optimization using DPSO for ASIP Architecture
- Baltes and Diehl, 2018, Usage and attribution of Stack Overflow code snippets in GitHub projects
- Jha and others, 2018, Worst Case Execution Time Estimation for Control Code of Automation Systems
- Aquino and others, 2018, Worst-Case Execution Time Testing via Evolutionary Symbolic Execution
- Fedasyuk and others, 2017, Architecture of a tool for automated testing the worst-case execution time of real-time embedded systems' firmware
- Tabassam and Obermaisser, 2017, Class-based query-optimization for minimizing worst-case execution times of diagnostic queries in embedded real-time systems
- Carminati and others, 2017, Combining loop unrolling strategies and code predication to reduce the worst-case execution time of real-time software
- Hausladen and others, 2017, Integration of Static Worst-Case Execution Time and Stack Usage Analysis for Embedded Systems Software in a Cloud-Based Development Environment
- Abella and others, 2017, Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation
- Jimenez Gil and others, 2017, Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time
- Knoop and others, 2017, Replacing conjectures by positive knowledge Inferring proven precise worst-case execution time bounds using symbolic execution
- Zou and others, 2017, Towards comprehending the non-functional requirements through Developers' eyes An exploration of Stack Overflow using topic analysis
- Mizobuchi and Takayama, 2017, Two improvements to detect duplicates in Stack Overflow
- Puffitsch, 2016, Efficient Worst-Case Execution Time Analysis of Dynamic Branch Prediction
- Mondal and others, 2016, Embedded Emotion-based Classification of Stack Overflow Questions Towards the Question Quality Prediction
- Seo and Kim, 2016, Measuring Method of Worst-case Execution Time by Analyzing Relation between Source Code and Executable Code
- Seo and Kim, 2016, Operator-data type pair based execution environments independent worst-case execution time measuring method
- Soleimani and Balarostaghi, 2016, Seismic image enhancement in post stack depth migration by finite offset CDS stack method
- Mguidich and others, 2016, Time-Accurate ASM as a Refinement Scheme for Worst-Case Execution Time Estimation in Hard Real-Time Systems
- 2016, WORST CASE EXECUTION TIME CALCULATION OF PARALLEL EMBEDDED REAL-TIME SOFTWARE
- Al-Bataineh and others, 2015, Accelerating worst case execution time analysis of timed automata models with cyclic behaviour
- Mushtaq and others, 2015, Calculation of worst-case execution time for multicore processors using deterministic execution
- Yu and Cohen, 2015, Guided Test Generation for Finding Worst-Case Stack Usage in Embedded Systems
- 2015, Incorporating Near-Surface Velocity Anomalies in Pre-Stack Depth Migration Models
- Wang and others, 2015, WCET-Aware Energy-Efficient Data Allocation on Scratchpad

Memory for Real-Time Embedded Systems

Compiler testing supplies the oracle and warns about its limits

The differential method used here is standard. Yang and others 2011 report Csmith and the compiler defects it found, Le, Afshari and Su 2014 introduce equivalence modulo inputs, and Chen and others 2020 survey compiler testing as a field.

The warning these carry is directly applicable. A differential oracle refutes and does not prove, and its power is bounded by the inputs supplied. This article's equivalence assertion is checked over a finite sample of resume sequences chosen by hand, the weakest form of the technique. Equivalence modulo inputs is the natural strengthening and it has not been applied.

- Klimis, 2026, Compilomorphic Fuzzing Turning a Compiler Against Itself
- Boda and others, 2026, CPerfSmith A Randomized C Program Generator for Performance-Oriented Compiler Testing
- Oh and Kim, 2026, Detecting Compiler-Introduced Security Bugs via IR Mutation and Coverage-Guided Fuzzing
- Shirai and others, 2026, Does Programming Language Matter? An Empirical Study of Fuzzing Bug Detection
- Liu and others, 2026, Learning Compiler Fuzzing Mutators from Historical Bugs
- Zhang and others, 2026, LLM-Powered Silent Bug Fuzzing in Deep Learning Libraries via Versatile and Controlled Bug Transfer
- Berlakovich and others, 2026, LOOL Low-Overhead, Optimization-Log-Guided Compiler Fuzzing
- Berlakovich and others, 2026, LOOL Low-Overhead, Optimization-Log-Guided Compiler Fuzzing - RCR Report
- Ren and others, 2026, SpiceFuzz LLM-Based Fuzzing for Spice Circuit Simulator Tools Bug Detection
- Zhang and others, 2026, Transformation-Recipe-Based FPGA Synthesis Compiler Testing
- Zeng and others, 2026, TypeNFuzz Dynamic Type-aware Object Dependence Graph-Guided Fuzzing for JavaScript Library Bug Discovery
- Ranjan and others, 2025, ClosureX Compiler Support for Correct Persistent Fuzzing
- Ali and others, 2025, CrossGuard Runtime-Adaptive LLM Fuzzing for Cross-Contract Vulnerabilities Detection
- Ye and others, 2025, BazzAFL Moving Fuzzing Campaigns Towards Bugs via Grouping Bug-Oriented Seeds
- Liu and others, 2025, BoostPolyGlot A Structured IR Generation-Based Fuzz Testing Framework for GCC Compiler Frontend
- Kim and others, 2025, Chimera Fuzzing P4 Network Infrastructure for Multi-Plane Bug Detection and Vulnerability Discovery
- Gao and others, 2025, Clozemaster Fuzzing Rust Compiler by Harnessing Llms for Infilling Masked Real Programs
- Boonriong and others, 2025, Compiler Fuzzing in Continuous Integration A Case Study on Dafny
- Tang and others, 2025, CTDip a diversity-guided test program synthesis approach for boosting compiler bug detection
- Zhu and others, 2025, Emerging Compiler Testing Based on Test Case Reuse
- Grabowski, 2025, Encoding Triangular Norms on Bounded Trellises with Mizar Proof Assistant
- Feng and others, 2025, Finding Compiler Bugs through Cross-Language Code Generator and Differential Testing
- Boo and Lee, 2025, Finding Device Driver Bugs With Fuzzing PCIe Configuration Input
- Gruner and others, 2025, Finding Information Leaks with Information Flow Fuzzing

- Gruner and others, 2025, Finding Information Leaks with Information Flow Fuzzing-RCR Report
- Qian and others, 2025, FreeWavm Enhanced WebAssembly Runtime Fuzzing Guided by Parse Tree Mutation and Snapshot
- Kim and others, 2025, Fuzzing Acceleration for Memory Safety Bug Discovery with Slicer
- Talaat and others, 2025, GrammLLM Grammar-Guided LLM Test Generation for Compiler Validation
- Ni and Li, 2025, Interleaving Large Language Models for Compiler Testing
- Xie and others, 2025, Kitten A Simple Yet Effective Baseline for Evaluating LLM-Based Compiler Testing Techniques
- Hyatt and Dewey, 2025, Mutation-Based Fuzzing of the Swift Compiler with Incomplete Type Information
- Ricardo and others, 2025, On the Practicality of LLM-Based Compiler Fuzzing
- Wang and others, 2025, Research on Compiler Fuzzing Based on Syntax-semantics Dual-Dimensional Classification
- Kokkonis and others, 2025, ROSA Finding Backdoors with Fuzzing
- Li and others, 2025, Solsmith Solidity Random Program Generator for Compiler Testing
- Hu and others, 2025, SSFuzz Synthesizing and scheduling bug-triggering code segments for history-driven compiler testing
- Zhao and others, 2025, Thread-sensitive fuzzing for concurrency bug detection
- Hazott and others, 2025, Using virtual prototypes and metamorphic testing to verify the hardware/software-stack of embedded graphics libraries
- Wu and others, 2025, WBSan WebAssembly Bug Detection for Sanitization and Binary-Only Fuzzing
- Zhou and others, 2024, C-CORE Clustering by Code Representation to Prioritize Test Cases in Compiler Testing
- Tian and others, 2024, Differential testing solidity compiler through deep contract manipulation and mutation
- Feitosa and Ribeiro, 2024, Differential Testing using Random Well-Typed Haskell Programs
- Georgescu and others, 2024, Evolutionary Generative Fuzzing for Differential Testing of the Kotlin Compiler
- Suo and others, 2024, Fuzzing MLIR Compiler Infrastructure via Operation Dependency Analysis
- Fan and others, 2024, History-driven Compiler Fuzzing via Assembling and Scheduling Bug-triggering Code Segments
- Munley and others, 2024, LLM4VV Developing LLM-driven testsuite for compiler validation
- Schwarcz and others, 2024, LOOL Low-Overhead, Optimization-Log-Guided Compiler Fuzzing Registered Report
- Han and others, 2024, Range Specification Bug Detection in Flight Control System Through Fuzzing
- Wang and Jung, 2024, Rustlantis Randomized Differential Testing of the Rust Compiler
- Li and others, 2024, Simulink Compiler Testing via Configuration Diversification With Reinforcement Learning
- Yang and others, 2024, WhiteFox White-Box Compiler Fuzzing Empowered by Large Language Models
- Ye and others, 2023, A Generative and Mutational Approach for Synthesizing Bug-Exposing Test Cases to Guide Compiler Fuzzing
- Wu and others, 2023, Boosting Compiler Testing via Eliminating Test Programs with Long-Execution-Time
- Lin and others, 2023, DeepDiffer Find Deep Learning Compiler Bugs via Priority-guided Differential Fuzzing
- Tu and others, 2023, Detecting C++ Compiler Front-End Bugs via Grammar Mutation

and Differential Testing
- Utting and others, 2023, Differential Testing of a Verification Framework for Compiler Optimizations Case Study
- Li and Su, 2023, Finding Unstable Code via Compiler-Driven Differential Testing
- Donaldson and others, 2023, Industrial Deployment of Compiler Fuzzing Techniques for Two GPU Shading Languages
- Wang and others, 2023, MLIRSmith Random Program Generation for Fuzzing MLIR Compiler Infrastructure
- Zhong and others, 2023, Neural Network Guided Evolutionary Fuzzing for Finding Traffic Violations of Autonomous Vehicles
- Kim and Hong, 2023, Poster BugOss A Regression Bug Benchmark for Empirical Study of Regression Fuzzing Techniques
- Sharma and others, 2023, RustSmith Random Differential Compiler Testing for Rust
- Ito and others, 2023, Schfuzz Detecting Concurrency Bugs with Feedback-Guided Fuzzing
- Qu and others, 2023, Scope-based Compiler Differential Testing
- Li and others, 2022, ALPHAPROG Reinforcement Generation of Valid Programs for Compiler Fuzzing
- Godboley and others, 2022, AV-AFL A Vulnerability Detection Fuzzing Approach by Proving Non-reachable Vulnerabilities using Sound Static Analyser
- Chen and Suo, 2022, Boosting Compiler Testing via Compiler Optimization Exploration
- Aljaafari and others, 2022, Combining BMC and Fuzzing Techniques for Finding Software Vulnerabilities in Concurrent Programs
- Liu and others, 2022, Coverage-guided tensor compiler fuzzing with joint IR-pass mutation
- Zhong, 2022, Enriching Compiler Testing with Real Program from Bug Report
- Groce and others, 2022, Making no-fuss compiler fuzzing effective
- Tu and others, 2022, Remgen Remanufacturing a Random Program Generator for Compiler Testing
- Wang and others, 2022, Unleashing Coveraged-Based Fuzzing Through Comprehensive, Efficient, and Faithful Exploitable-Bug Exposing
- Kasampalis and others, 2021, Language-parametric compiler validation with application to LLVM
- Hazimeh and others, 2021, Magma A Ground-Truth Fuzzing Benchmark
- Stepanov and others, 2021, Type-Centric Kotlin Compiler Fuzzing Preserving Test Program Correctness by Preserving Types
- Xu and others, 2020, DSmith Compiler Fuzzing through Generative Deep Learning Model with Attention
- Chen and others, 2020, Enhanced compiler bug isolation via memoized search
- Kim and others, 2020, Finding Bugs in File Systems with an Extensible Fuzzing Framework
- Marcozzi and others, 2019, Compiler fuzzing how much does it matter?
- Tang and others, 2019, Compiler testing a systematic literature analysis
- Koroglu and Wotawa, 2019, Fully Automated Compiler Testing of a Reasoning Engine via Mutated Grammar Fuzzing
- Li, 2019, Haskell Compiler Testing Automation Based on Equivalence-Modulo-Inputs Method
- Jeong and others, 2019, Razzer Finding Kernel Race Bugs through Fuzzing
- Zaytsev, 2018, An industrial case study in compiler testing tool demo
- Holmes and Groce, 2018, Causal Distance-Metric-Based Assistance for Debugging after Compiler Fuzzing
- Cummins and others, 2018, Compiler fuzzing through deep learning
- Barany, 2018, Finding missed compiler optimizations by differential testing
- Chen, 2018, Learning to accelerate compiler testing

- Kargén and Shahmehri, 2018, Speeding Up Bug Finding using Focused Fuzzing
- Jay and Miller, 2018, Structured random differential testing of instruction decoders
- Julián-Iranzo and Rubio-Manzano, 2017, A sound and complete semantics for a similarity-based logic programming language
- Lidman and Svenningsson, 2017, Bridging Static and Dynamic Program Analysis using Fuzzy Logic
- Wang and others, 2017, Faster mutation analysis via equivalence modulo states
- Zhang and others, 2017, Skeletal program enumeration for rigorous compiler testing
- ISHIURA, 2016, Compiler Fuzzing
- Alatawi and others, 2016, Generating source inputs for metamorphic testing using dynamic symbolic execution
- Lidbury and others, 2015, Many-core compiler fuzzing

The calling convention itself has a literature, and it is mostly about stability

The article treats a calling convention as a design choice, which is the compiler author's view of it. The larger literature treats it as an interface contract whose principal property is that it must not change, because every separately compiled artefact already linked against it would break. **That is the sense in which adding a discriminator to the return convention is expensive**, and it is a cost the cardinality argument does not see. The cardinality argument says a wider convention is available. The interface literature says a wider convention is a different convention, and that the cost of adopting it falls on everything already compiled.

Keleusma is pre-1.0 and has no external artefacts to break, so the article is right that the cost is not yet being paid. It will be, and the decision made now is the one that will be expensive to revisit.

- Zapanov, 2025, Foreign Function Interface for Managed Runtime Systems with Lightweight Threading
- Nagata and others, 2024, Evaluation of Interoperability of CNN Models between MATLAB and Python Environments Using ONNX Runtime Model
- Chichereau and others, 2024, Fully Integrated Quantum Method for Classical Register Allocation in LLVM
- Terci, 2023, A Learning-Based Coloring Algorithm for Register Allocation Problem
- Panigrahi and Karfa, 2023, An Investigation into the Security of Register Allocation with Spilling and Splitting
- Petrosino and others, 2023, Cross-Paradigm Interoperability Between Jadescript and Java
- Hammond and others, 2023, MPI Application Binary Interface Standardization
- Xiao and others, 2023, Read-Write Dependency Aware Register Allocation
- Fried and others, 2023, Register Allocation for Compressed ISAs in LLVM
- VenkataKeerthy and others, 2023, RL4ReAl Reinforcement Learning for Register Allocation
- Tant and others, 2023, Software Compilation Using FPGA Hardware Register Allocation
- De Paula and Ierusalimschy, 2022, A Foreign Function Interface for Pallene
- Chen and others, 2022, Register allocation compilation technique for ASIP in 5G micro base stations
- Hu and others, 2022, Research on global register allocation for code containing array-unit dual-usage register names
- Wu and others, 2021, Effective Register Allocation for Configurable VLIW Crypto-Processor
- Das and others, 2020, Deep Learning-based Approximate Graph-Coloring Algorithm for Register Allocation
- Doronin, 2019, PROBLEM RESEARCH AND DEVELOPMENT OF A TOOL FOR CHECKING APPLICATION BINARY INTERFACE COMPATIBILITY OF VIRTUAL METHOD TA-

BLES
- Yallop and others, 2018, A modular foreign function interface
- Rath and others, 2018, Interoperability-Guided Testing of QUIC Implementations using Symbolic Execution
- Eisl and others, 2018, Parallel trace register allocation
- Bee and bernardo, 2018, Predicting Fork Visibility Performance on Programming Language Interoperability in Open Source Projects
- Caldwell and Chiba, 2017, Reducing calling convention overhead in object-oriented programming on embedded ARM thumb-2 platforms
- Carlotto and others, 2016, Interoperability of Annotation Schemes Using the Pepper Framework to Display AWA Documents in the ANNIS Interface
- Lozano and others, 2016, Register allocation and instruction scheduling in Unison
- Domagała and others, 2016, Register allocation and promotion through combined instruction scheduling and loop unrolling
- Burroughs, 2016, Register allocation and spilling using the expected distance heuristic
- Eisl and others, 2016, Trace-based Register Allocation in a JIT Compiler
- Sukha, 2015, A Compiler-Runtime Application Binary Interface for Pipe-While Loops
- Inoue and Kaneko, 2015, Bitwidth-aware register allocation and binding for clock period minimization
- Krause, 2015, Bytewise Register Allocation
- Ditu, 2015, Model-Based Function Call Code Generation and Stack Management in Retargetable Compilers Application Binary Interface Modeling of Stack Layout and Function Call Sequence
- Eisl, 2015, Trace register allocation
- You and Chen, 2015, Vector-aware register allocation for GPU shader processors

Static analysis supplies the vocabulary for a rule stricter than its property

The rule this article examines admits 14 of 24 chunks while the property it enforces holds for all 24, which is a soundness-preserving imprecision. The literature that names this condition, measures it and reduces it is large and the article uses only one paper from it.

The article's position within this literature is unusual and worth being explicit about. The classical justification for an over-strict rule is that the exact property is uncomputable, so an approximation is forced. This article's property is decidable on straight-line bytecode by summing two integers, so the imprecision is not forced and the rule is simply cruder than it needs to be. That is a better position to be in than the classical one and it is a different problem.

- Iyenghar and others, 2026, Incremental Static Analysis for Detecting and Refactoring Data Clumps in TypeScript
- Bardonek and Zachariasova, 2026, Leveraging design static analysis for vertical reuse Analytical interpretation and scalability behavior
- Negrini, 2026, Whole-value analysis by abstract interpretation
- Dimovski, 2025, Imperative Program Synthesis by Abstract Static Analysis and SMT Mutations
- Azmi, 2025, LLM-Aware Static Analysis Adapting Program Analysis to Mixed Human/AI Codebases at Scale
- Berg and others, 2025, Manim-DFA Visualising Data Flow Analysis and Abstract Interpretation Algorithms with Automated Video Generation
- Urban and others, 2025, Static analysis by abstract interpretation against data leakage in machine learning
- Simonnet and others, 2024, A Dependent Nominal Physical Type System for Static Analysis of Memory in Low Level Code
- Baek and Lee, 2024, CaLLi OCaml library for static analysis of LLVM bitcode

- Guan and Treude, 2024, Enhancing Source Code Representations for Deep Learning with Static Analysis
- Sharma and Sharma, 2024, Parameterized Static Analysis for Weak Memory Models
- Liu and others, 2024, Research on Higher-Order Operator Transformation Algorithms for Syntax Transformation-Based Model Checking
- Taft, 2024, Sound and precise static analysis using a generalization of static single assignment and value numbering
- Schnakenbeck and others, 2023, A Control Flow based Static Analysis of GRAFCET using Abstract Interpretation
- Coward and others, 2023, Combining E-Graphs with Abstract Interpretation
- Wei and others, 2023, Compiling Parallel Symbolic Execution with Continuations
- Barai and others, 2023, LLVM Static Analysis for Program Characterization and Memory Reuse Profile Estimation
- JANA, 2023, Sensitive Information Leakage Analysis of Database Code by Abstract Interpretation
- Haas and others, 2023, Static Analysis of Memory Models for SMT Encodings
- Bau and others, 2022, Abstract interpretation of Michelson smart-contracts
- Lu and others, 2022, Gradual Soundness Lessons from Static Python
- Yang and others, 2022, Simulink Model Static Analysis Results based on Abstract Interpretation
- Bartsch and others, 2021, Compositional Fault Propagation Analysis in Embedded Systems using Abstract Interpretation
- Shimchik and others, 2021, Improving Accuracy and Completeness of Source Code Static Taint Analysis
- Dimovski, 2021, Lifted termination analysis by abstract interpretation and its applications
- Punnoose, 2020, Ensuring completeness of formal verification with Gap Free Verification
- Wang and others, 2020, RSDS Getting System Call Whitelist for Container Through Dynamic and Static Analysis
- van Tonder and Le Goues, 2020, Tailoring programs for static analysis via program transformation
- Yamane and others, 2020, Verification Method of Safety Properties of Embedded Assembly Program by Combining SMT-Based Bounded Model Checking and Reduction of Interrupt Handler Executions
- Bhushan and Yadav, 2020, Verification of Virtual Machine Architecture in a Hypervisor through Model Checking
- Sato and others, 2019, Combining higher-order model checking with refinement type inference
- Meyer and Wolff, 2019, Decoupling lock-free data structures from memory reclamation for static analysis
- Blaß and Philippsen, 2019, GPU-accelerated fixpoint algorithms for faster compiler analyses
- Loring and others, 2019, Sound regular expression semantics for dynamic symbolic execution of JavaScript
- Dong, 2018, A sound abstract memory model for static analysis of C programs
- Keidel and others, 2018, Compositional soundness proofs of abstract interpreters
- Gerasimov, 2018, Directed Dynamic Symbolic Execution for Static Analysis Warnings Confirmation
- Roşu, 2018, Finite-trace linear temporal logic coinductive completeness
- Sherman, 2018, Redesigning Soot's data-flow analysis framework for abstract interpretation
- Alkhalid and Labiche, 2018, Towards GUI Functional Verification using Abstract Interpretation
- Liang and others, 2017, Improving the precision of static analysis Symbolic execution

based on GCC abstract syntax tree
- Kobayashi and others, 2017, On the relationship between higher-order recursion schemes and higher-order fixpoint logic
- Orlov, 2017, Program Source Code Static Analysis for Memory Access Error Detection Using Backwards Execution
- Kusano and Wang, 2017, Thread-modular static analysis for relaxed memory models
- Miné, 2017, Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation
- Suwa and others, 2017, Verification of code generators via higher-order model checking
- Haase, 2016, Abstract Interpretation of Java Bytecode for Immutability Analysis
- Ouadjaout and others, 2016, Static analysis by abstract interpretation of functional properties of device drivers in TinyOS
- Simon and Kowalewski, 2016, Static analysis of Sequential Function Charts using abstract interpretation
- Bodin and others, 2015, Certified Abstract Interpretation with Pretty-Big-Step Semantics
- Zhang and Koutsoukos, 2015, Improving the Precision of Abstract Interpretation Based Cache Persistence Analysis
- Cousot, 2015, On Various Abstract Understandings of Abstract Interpretation
- Dong, 2015, RSTVL A Sound Abstract Memory Model for Program Static Analysis
- Montenegro and others, 2015, Space consumption analysis by abstract interpretation Reductivity properties
- Bertrane and others, 2015, Static Analysis and Verification of Aerospace Software by Abstract Interpretation
- Cousot, 2015, Verification by abstract interpretation, soundness and abstract induction

The bytecode layer is where the property is actually decidable

The backend consumes a verified bytecode, and the whole argument about stack effects lives at that level and not at the source or the machine level. The virtual-machine literature is therefore where the article's decidability claim has to be checked, and it is also the differential oracle's other side, since the oracle compares the virtual machine against the native code.

- Parashar and others, 2026, A Comparative Architectural and Performance Analysis of WebAssembly and JavaScript for Computationally Intensive Web Applications
- Corrias and others, 2026, An Analysis of Modern Web Security Vulnerabilities Inside WebAssembly Applications
- Israel and others, 2026, Exploring WebAssembly as a Runtime Platform for Safety-Critical Edge Systems
- I.Saetchnikov and others, 2026, Microresonator clusters for spectral analysis with machine-learning interpreter
- Hu and others, 2026, QJWasm A lightweight runtime system for efficient WebAssembly execution in resource-constrained environments
- Wu and others, 2026, Reconfigurable Computing Challenge FPGA-Based WebAssembly Stack Co-Processor
- Choi and Jeon, 2026, Stack-based static WebAssembly binary slicing and mutation for generating valid sub-binaries
- Parra, 2026, SurtGIS A high-performance raster geospatial analysis library in Rust with WebAssembly and Python support
- Blaak and Van Cutsem, 2026, Towards Least-Privilege WebAssembly Applications Transparent Interposition for WebAssembly Components
- Müller and others, 2026, WasmWeaver A Framework for Runtime-Aware WebAssembly Program Generation with Reinforcement Learning
- Massey and Olivier, 2026, WASP Stack protection for WebAssembly

- Ţălu, 2025, A Comparative Study of WebAssembly Runtimes Performance Metrics, Integration Challenges, Application Domains, and Security Features
- Stepanov and Klym, 2025, A QUANTITATIVE ANALYSIS OF WEBASSEMBLY INTEGRATION ARCHITECTURAL PATTERNS, TOOLING, AND PERFORMANCE EVALUATION
- Moskalenko, 2025, Application of WebAssembly for High-Performance Client-Side Media Content Analysis
- Watt, 2025, Concurrency in WebAssembly
- Subramanyan, 2025, CWAMR REIMAGINING A CAPABILITYBASED WEBASSEMBLY RUNTIME VIA CHERI-BASED COMPARTMENTALIZATION
- Lu and others, 2025, Detecting WebAssembly Runtime Bugs With Grammar-Guided Program Mutation
- Jiang and others, 2025, Distinguishability-Guided Test Program Generation for WebAssembly Runtime Performance Testing
- SanthiKumar and others, 2025, Enhancing Machine Learning Performance with AI-Based Virtual Machine Load Balancing
- Schlägl and Groβe, 2025, Fast Interpreter-Based Instruction Set Simulation for Virtual Prototypes
- Lei and others, 2025, Furina A Light-weight WebAssembly Runtime for ICS
- Kang and others, 2025, HybridServe Adaptive WebAssembly-Container Runtime Selection for Edge Serverless Computing
- Mallawarachchi and Jayaweera, 2025, Implementation of Wyltl An Imperative Language with a Dual Interpreter Compiler Architecture
- Marcelino and others, 2025, Lumos Performance Characterization of WebAssembly as a Serverless Runtime in the Edge-Cloud Continuum
- Kakati and Brorsson, 2025, Performance and Usability Implications of Multiplatform and WebAssembly Containers
- MIDZIC and NOVAK, 2025, Performance Comparison of WebAssembly and Phaser in Procedural Maze Generation
- Kim and others, 2025, Pre-trained Models for Bytecode Instructions
- Soluian and Liushenko, 2025, Research of WebAssembly usage for high-performance code development in web applications
- Mao and others, 2025, Research on a Lightweight Full-Stack Edge Execution Optimization Framework Based on Serverless and WebAssembly
- Matsubara and others, 2025, Seamless Self-Healing in WebAssembly Container Orchestration with Runtime-Neutral Checkpointing
- Nakata and Matsubara, 2025, Self-Hosted WebAssembly Runtime for Runtime-Neutral Checkpoint/Restore in Edge-Cloud Continuum
- Satya Teja Muddada, 2025, Serverless 2.0 Unlocking Performance and Portability with WebAssembly
- Bhoyar and others, 2025, Sign Language Interpreter using Long Short-Term Memory LSTM
- Parwez and others, 2025, Signease Machine Learning Based Sign Language Interpreter
- Dico and Tata Sutabri, 2025, Virtual Desktop Infrastructure Sebagai Pendukung Perkuliahan Dengan Algoritma Virtual Machine
- Liu and others, 2025, WebAssembly for Container Runtime Are We There Yet?
- Karpovich and Gosudarev, 2025, WebAssembly performance in the Node.js environment
- Titzer, 2025, WebAssembly How Low Can a Bytecode Go?
- Krishnamoorthy, 2025, WEBASSEMBLY REVOLUTIONIZING WEB PERFORMANCE AND EXPANDING FRONTIERS OF BROWSER-BASED APPLICATIONS
- Heinrich and others, 2024, A Categorical Data Approach for Anomaly Detection in WebAssembly Applications
- Ménétrey and others, 2024, A Comprehensive Trusted Runtime for WebAssembly With Intel SGX

- Massidda and others, 2024, Bringing Binary Exploitation at Port 80 Understanding C Vulnerabilities in WebAssembly
- A and others, 2024, Design of Mechatronics Based Virtual Telepresence for Robotic System Using Raspberry Python Interpreter
- Silva and others, 2024, Efficient Data Exchange between WebAssembly Modules
- Bunkenburg and Wu, 2024, Making a Curry Interpreter using Effects and Handlers
- Michaud and others, 2024, Robust Stack Smashing Protection for WebAssembly
- Harnes and Morrison, 2024, SoK Analysis Techniques for WebAssembly
- Zhao and others, 2024, Wapplique Testing WebAssembly Runtime via Execution Context-Aware Bytecode Mutation
- Nikhil Sripathi Rao, 2024, WebAssembly Revolutionizing Web User Interface Development through Performance and Cross-Language Integration
- Wang and others, 2023, A Comprehensive Study of WebAssembly Runtime Bugs
- Verma and others, 2023, Array Bytecode Support in MicroJIT
- Rosà and others, 2023, Automated Runtime Transition between Virtual and Platform Threads in the Java Virtual Machine
- Park and others, 2023, Bespoke Virtual Machine Orchestrator An Approach for Constructing and Reconfiguring Bespoke Virtual Machine in Private Cloud Environment
- Zhang and others, 2023, Characterizing and Detecting WebAssembly Runtime Bugs
- Lowther and others, 2023, CHERI Performance Enhancement for a Bytecode Interpreter
- Pockstaller and others, 2023, Comparing the Energy Consumption of WebAssembly and JavaScript in Mobile Browsers
- Marcelino and Nastic, 2023, CWASI A WebAssembly Runtime Shim for Inter-function Communication in the Serverless Edge-Cloud Continuum
- He and others, 2023, Neural-FEBI Accurate function identification in Ethereum Virtual Machine bytecode
- Vepuri and Jiang, 2023, Performance Analysis of Virtual Machine Monitoring System
- Chung Seo and Kim, 2023, Portable and Efficient Implementation of CRYSTALS-Kyber Based on WebAssembly
- Rokotyanskaya and Abramov, 2023, Studying WebAssembly and comparison of its performance with JavaScript
- Johnson and others, 2023, WaVe a verifiably secure WebAssembly sandboxing runtime
- Romano and Wang, 2023, When Function Inlining Meets WebAssembly Counterintuitive Impacts on Runtime Performance
- Kovačević and others, 2022, Automatic compiler/interpreter generation from programs for Domain-Specific Languages Code bloat problem and performance improvement
- Othman and El Ghoul, 2022, BuHamad - The first Qatari virtual interpreter for Qatari Sign Language
- Kyriakou and Tselikas, 2022, Complementing JavaScript in High-Performance Node.js and Web Applications with Rust and WebAssembly
- Sahkhar and others, 2022, Efficient Cloudlet Allocation to Virtual Machine to Impact Cloud System Performance
- Lehmann and Pradel, 2022, Finding the Dwarf Recovering Precise Types from WebAssembly Binaries
- Anderton, 2022, Investigating Sign Language Interpreter Rendering and Guiding Methods in Virtual Reality 360-Degree Content
- Sai and others, 2022, Machine learning-based malware detection using stacking of opcodes and bytecode sequences
- Jodogne, 2022, Rendering Medical Images using WebAssembly
- Stiévenart and others, 2022, Static stack-preserving intra-procedural slicing of webassembly binaries
- Chmiel and Spinolo, 2022, Testing the impact of remote interpreting settings on interpreter experience and performance

- Thorpe and others, 2022, Verification of Cyber Emulation Experiments Through Virtual Machine and Host Metrics
- Menetrey and others, 2022, WaTZ A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone
- De Macedo and others, 2022, WebAssembly versus JavaScript Energy and Runtime Performance
- Thorpe and others, 2022, WiP Verification of Cyber Emulation Experiments Through Virtual Machine and Host Metrics
- Zhang and Yin, 2021, A Virtual Machine Placement Strategy Based on Virtual Machine Selection and Integration
- Wang, 2021, Can “micro VM” become the next generation computing platform? Performance comparison between light weight Virtual Machine, container, and traditional Virtual Machine
- M and Murugesh, 2021, Comparative Study of Binary Classification Algorithms to Analyze the Students Performance on Virtual Machine
- Penev and Dimitrov, 2021, Design of a Virtual Machine for Training Compilers
- Hara and others, 2021, Machine-learning Approach using Solidity Bytecode for Smart-contract Honeypot Detection in the Ethereum
- De Macedo and others, 2021, On the Runtime and Energy Performance of WebAssembly Is WebAssembly superior to JavaScript yet?
- Menetrey and others, 2021, Twine An Embedded Trusted Runtime for WebAssembly
- -, 2021, WebAssembly for High-Performance Web Applications A Study on Execution Speed and Efficiency
- Ke and Chen, 2020, Instruction Verification of Ethereum Virtual Machine by Formal Method
- Bockisch and others, 2020, Java Bytecode Verification with OCL Why, How and Whenc
- E., 2020, Modified Support Vector Machine based Efficient Virtual Machine Consolidation Procedure for Cloud Data Centers
- Wahyudi and Miswanto, 2020, VIRTUAL MACHINE FORENSIC ANALYSIS and RECOVERY VMFAR SEBAGAI FRAMEWORK UNTUK ANALISIS BUKTI DIGITAL PADA VIRTUAL MACHINE
- Pinckney and others, 2020, Wasm/k delimited continuations for WebAssembly
- V and others, 2019, A WEIGHTED ENSEMBLE OF AUTOMATIC ALGORITHMS FOR VIRTUAL MACHINE PERFORMANCE PREDICTION IN CLOUD
- Choi and Hong, 2019, Design and implementation of virtual machine control and streaming scheme using Linux kernel-based virtual machine hypercall for virtual mobile infrastructure
- Zheng and Xia, 2019, Exploring mixed integer programming reformulations for virtual machine placement with disk anti-colocation constraints
- Cabrera Arteaga and others, 2019, Scalable comparison of JavaScript V8 bytecode traces
- Salim and others, 2019, Towards a WebAssembly standalone runtime on GraalVM
- Moliavko and others, 2019, uJVM Lightweight Java Virtual Machine for embedded systems
- Achour and others, 2018, A Constraint-Based Verification Approach for Java Bytecode Programs
- Park and others, 2018, A formal verification tool for Ethereum VM bytecode
- Liu and Li, 2018, A novel virtual machine scheduling policy based on performance prediction model
- Sohrabi and others, 2018, A Novel Virtual Machine Selection Policy for Virtual Machine Consolidation
- Kim and Lee, 2018, A Study on the Code Generator for a Virtual Machine Code based JavaScript Compiler
- Melo Alves and others, 2018, An Interference-Aware Virtual Machine Placement Strategy for High Performance Computing Applications in Clouds

- Achour and Benattou, 2018, Constraint Based Testing and Verification of Java Bytecode Programs
- Jamil, 2018, Design of a Real-Time Interpreter for Arabic Sign Language
- Attrapadung and others, 2018, Efficient Two-level Homomorphic Encryption in Prime-order Bilinear Groups and A Fast Implementation in WebAssembly
- Kuang and others, 2018, Enhance virtual-machine-based code obfuscation security through dynamic bytecode scheduling
- Yang, 2018, Formal Process Virtual Machine for Smart Contracts Verification
- Hale and others, 2018, Interpreter performance in police interviews. Differences between trained interpreters and untrained bilinguals
- Dobravec, 2018, JAVA BYTECODE INSTRUCTION USAGE COUNTING WITH ALGATOR
- V. Samuel Blessed Nayagam and Shajin Nargunam, 2018, Secure Data Verification and Virtual Machine Monitoring
- Madsen and others, 2018, Tail call elimination and data representation for functional languages on the Java virtual machine
- Sianipar and others, 2018, Virtual Machine Integrity Verification in Crowd-Resourcing Virtual Laboratory
- Kim and Lee, 2017, A Study on the Light-Weight Virtual Machine Code for IoT Virtual Machine
- Haas and others, 2017, Bringing the web up to speed with WebAssembly
- Son and others, 2017, Design and Implementation of the RSIL to LLVM IR Translator for Verification of the Intermediate Code on IoT Virtual Machine
- Lee and others, 2017, Design and implementation of the secure compiler and virtual machine for developing secure IoT services
- Lee, 2017, Exploring a relationship between students' interpreting self-efficacy and performance triangulating data on interpreter performance assessment
- Sheinidashtegol and Galloway, 2017, Performance Impact of DDoS Attacks on Three Virtual Machine Hypervisors
- 2017, The Utilization of Cloud Computing as Virtual Machine
- 2017, Virtual Machine Consolidation using Load Balancing algorithm in Cloud Data Center
- Son and Lee, 2016, A Study on the Interpreter for the Light-Weighted Virtual Machine on IoT Environments
- 2016, Analytics of Application Resource Utilization within the Virtual Machine
- Han, 2016, Building the validity foundation for interpreter certification performance testing
- 2016, Comparative Study of Virtual Machine Migration Techniques and Challenges in Post Copy Live Virtual Machine Migration
- Cheng and others, 2016, Formalised EMFTVM bytecode language for sound verification of model transformations
- Tokumoto and others, 2016, MuVM Higher Order Mutation Analysis Virtual Machine for C
- 2016, Network-Aware Virtual Machine Placement in the Cloud
- CLERC, 2016, OCaml-Java The Java Virtual Machine as the target of an OCaml compiler
- Zhou and Mu, 2016, Representative Virtual Machine Templates An optimized virtual machine templates management mechanism for an Cloud system based on K-medoids Clustering
- O'Loughlin and Gillam, 2015, Addressing Issues of Cloud Resilience, Security and Performance through Simple Detection of Co-locating Sibling Virtual Machine Instances
- Nanthaamornphong and others, 2015, Bytecode-based class dependency extraction tool Bytecode-CDET
- Upadhyaya and Rajan, 2015, Effectively mapping linguistic abstractions for message-passing concurrency to threads on the Java virtual machine
- Gunadi, 2015, Formal Certification of Non-interferent Android Bytecode DEX Bytecode

- Salapura and Harper, 2015, High Performance Virtual Machine Recovery in the Cloud
- Pan and others, 2015, Nonvolatile main memory aware garbage collection in high-level language virtual machine
- Galloway and others, 2015, Performance Metrics of Virtual Machine Live Migration
- Ruan and Chen, 2015, Performance-to-Power Ratio Aware Virtual Machine VM Allocation in Energy-Efficient Clouds
- Salapura and Harper, 2015, Remote Restart for a High Performance Virtual Machine Recovery in a Cloud
- 2015, The Deasibility and Properties of Dividing Virtual Machine Resources using the Virtual Machine Cluster as the Unit in Cloud Computing
- Hui Zhao and others, 2015, Virtual machine placement based on the VM performance models in cloud

Types and semantics decide whether the question arises at all

The observation the article contributes is that two source forms have different trace alphabets, which is a statement about the source language's type system before it is a statement about the backend. A language whose types separate a stream from a terminating coroutine can select the convention per form. A language whose types do not cannot, and must widen the convention globally or pay for a general mechanism. **The type system is therefore the mechanism by which the choice becomes available**, and the literature on what type systems can express about control effects is where the boundary of that availability is drawn.

- Arendsee, 2026, morloc a workflow language for multi-lingual programming under a common type system
- DOWNEN and ARIOLA, 2025, A contextual formalization of structural coinduction
- Baramashetru and others, 2025, A Type System for Data Privacy Compliance in Active Object Languages
- Correnson and Finkbeiner, 2025, Coinductive Proofs for Temporal Hyperliveness
- Kolesar and others, 2025, Coinductive Proofs of Regular Expression Equivalence in Zero Knowledge
- Di Lavore and others, 2025, Coinductive Streams in Monoidal Categories
- Kidney and Wu, 2025, Formalising Graph Algorithms with Coinduction
- Lee and others, 2025, React-tRace A Semantics for Understanding React Hooks An Operational Semantics and a Visualizer for Clarifying React Hooks
- Hyvernat, 2025, The Size-Change Principle for Mixed Inductive and Coinductive types
- Hyvernat, 2025, Totality for Mixed Inductive and Coinductive Types
- Dinikeev, 2025, Type system for a statically typed concatenative programming language with first class function support
- Smith and Zhang, 2024, A Pure Demand Operational Semantics with Applications to Program Analysis
- Chawla and others, 2024, COMPARATIVE STUDY OF PROPOFOL AUTO-COINDUCTION VERSUS KETAMINE PROPOFOL COINDUCTION USING PRIMING PRINCIPLE BY BISPECTRAL INDEX ANALYSIS FOR DAY CARE SURGERY
- Grabmayer, 2023, A Coinductive Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity
- Francalanza and Tabone, 2023, ElixirST A session-based type system for Elixir modules
- Castagna and others, 2023, The Design Principles of the Elixir Type System
- Milano and others, 2022, A flexible type system for fearless concurrency
- Mastorou and others, 2022, Coinduction inductively mechanizing coinductive proofs in Liquid Haskell
- Sangiorgi, 2022, From enhanced coinduction towards enhanced induction
- Lambert and others, 2022, Leveraging Compiler-Based Translation to Evaluate a Diversity of Exascale Platforms

- Kanatov and Zouev, 2022, Unified type system for the modern general-purpose programming language
- Mihelic and others, 2021, A denotational semantics of a concatenative/compositional programming language
- De and others, 2021, Canonical proof-objects for coinductive programming infinets with infinitely many cuts
- Kuperberg and others, 2021, Coinductive Algorithms for Büchi Automata
- Kupke and Rot, 2021, Expressive Logics for Coinductive Predicates
- Dagnino, 2021, Foundations of regular coinduction
- Farkas and others, 2021, Improving productivity in large scale testing at the compiler level by changing the intermediate language from C++ to Java
- Sato, 2021, Proof Assistant and Type Theory
- 2020, A Fibrational Method of Indexed Coinductive Data Types
- Cassola and others, 2020, A Gradual Type System for Elixir
- Czajka, 2020, A new coinductive confluence proof for infinitary lambda calculus
- Zakowski and others, 2020, An equational theory for weak bisimulation via generalized parameterized coinduction
- Czajka, 2020, An operational interpretation of coinductive types
- Dagnino, 2020, Coaxioms flexible coinductive definitions by inference systems
- Zúñiga and Bel-Enguix, 2020, Coinductive Natural Semantics for Compiler Verification in Coq
- Abe, 2019, A type system for data independence of loop iterations in a directive-based PGAS language
- Inoue and Igarashi, 2019, A type system for first-class layers with inheritance, subtyping, and swapping
- Biernacki and others, 2019, Bisimulations for Delimited-Control Operators
- Hirschowitz, 2019, Familial monads and structural operational semantics
- Yesua and others, 2019, Keamanan Penggunaan Propofol Auto-Coinduction Dibandingkan Dengan Midazolam Coinduction Berdasarkan Perubahan Hemodinamik Pada Induksi Anestesi Pasien Yang Dilakukan General Anestesi
- Nishizaki, 2019, ML Polymorphism of Linear Lambda Calculus with First-class Continuations
- Pelsmaeker and others, 2019, Towards language-parametric semantic editor services based on declarative type system specifications
- Bosse, 2018, A Unified System Modelling and Programming Language based on JavaScript and a Semantic Type System
- Gupta and Lewis, 2018, Neural Compositional Denotational Semantics for Question Answering
- Lucanu, 2018, Proving Reachability Properties by Coinduction Extended Abstract
- Sokhatskyi and Maslianko, 2018, The systems engineering of consistent pure language with effect type system for certified applications and higher languages
- Komendantskaya and Li, 2018, Towards Coinductive Theory Exploration in Horn Clause Logic Position Paper
- Goncharov and others, 2018, Unguarded Recursion on Coinductive Resumptions
- Liu and others, 2017, Analyzing divergence in bisimulation semantics
- Aristizábal and others, 2017, Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation
- Harlin and others, 2017, Impact of Using a Static-Type System in Computer Programming
- Aparanji and others, 2017, INDUCTION OF PROPOFOL WITH COINDUCTION OF PROPOFOL, MIDAZOLAM VERSUS PROPOFOL AUTO COINDUCTION- A COMPARATIVE STUDY
- Nishizaki, 2017, Linear lambda calculus with non-linear first-class continuations

- Nishizaki, 2017, Type Inference of Linear Lambda Calculus with First-Class Continuations
- Berger and Spreen, 2016, A coinductive approach to computing with compact sets
- Swierstra and others, 2016, A Lazy Language Needs a Lazy Type System
- Sculthorpe and others, 2016, A Modular Structural Operational Semantics for Delimited Continuations
- Pous, 2016, Coinduction All the Way Up
- KOZEN and SILVA, 2016, Practical coinduction
- Pattinson and Schröder, 2016, Program Equivalence is Coinductive
- Rot and others, 2016, Proving language inclusion and equivalence by coinduction
- Bruza, 2016, Syntax and operational semantics of a probabilistic programming language with scopes
- Ciaffaglione, 2016, Towards Turing computability via coinduction
- Basold and Geuvers, 2016, Type Theory based on Dependent Inductive and Coinductive Types
- Nakata and Uustalu, 2015, A Hoare logic for the coinductive trace-based big-step semantics of While
- Gan and Li, 2015, Coinduction functor in representation stability theory
- Pous, 2015, Coinductive techniques, from automata to coalgebra
- Bonchi and Pous, 2015, Hacking nondeterminism with induction and coinduction
- Schmidt-Schauß and Sabel, 2015, Improvements in a functional core language with call-by-need operational semantics

Concurrency runtimes are where the general mechanism is affordable

The callback convention holds a frame across arbitrary host execution, which is what a runtime with a scheduler does routinely and what a statically bounded artefact cannot do. The contrast is the article's central trade seen from the other side, so the runtime literature is listed here as the case where the general answer is correct.

- Sharma and Reddy, 2025, Analysis of FreeRTOS and Contiki Scheduler Performance with Scalable Task Loads on a Uniform Platform
- Sharma, 2025, Green Threads Human Connection with Nature in Richard Powers' The Overstory
- Zhang and others, 2025, Refactoring for Java-Structured Concurrency
- Altassan and Ahmad, 2024, Green threads of change Unravelling the gendered and experienced moderators in the sustainable symphony of green HR practices and environmental responsibility
- Sadanand Giri and others, 2024, Green threads of progress Natural fibers reshaping wastewater cleanup strategies, a review
- Kim and Park, 2023, A Multi-core Based Real-time Scheduler Supporting Periodic and Sporadic Threads and Processes
- Serafin and others, 2023, Pipestitch An energy-minimal dataflow architecture with lightweight threads
- Jagnik, 2023, Structured Concurrency in Java
- Ciesko and Roussel, 2023, User-Level Threading for HPC Applications
- Zou and others, 2022, Buddy Stacks Protecting Return Addresses with Efficient Thread-Local Storage and Runtime Re-Randomization
- Ueno and Ohori, 2022, Concurrent and parallel garbage collection for lightweight threads on multicore processors
- Shiina and others, 2021, Lightweight preemptive user-level threads
- Karsten and Barghi, 2020, User-level Threading
- Carvalho and others, 2019, A dataflow runtime environment and static scheduler for edge, fog and in-situ computing

- Iwasaki and others, 2018, Lessons Learned from Analyzing Dynamic Promotion for User-Level Threading
- Kalebe and others, 2017, A library for scheduling lightweight threads in Internet of Things microcontrollers
- Huynh and Taura, 2017, Delay Spotter A Tool for Spotting Scheduler-Caused Delays in Task Parallel Runtime Systems

The historical layer

The clusters above are drawn from work published in 2015 or later, which is the survey's contemporary window. The older records the harvest returned are listed here in one place instead of distributed through the sections above, because for most of these clusters the pre-2015 work is the foundation the contemporary work cites rather than a competing account. There are 679 of them.

- Liu and others, 2011, Coroutine-Based Synthesis of Efficient Embedded Software From SystemC Models
- Kristensen and others, 1987, Coroutine Sequencing in BETA
- Kearns and Lou Soffa, 1983, The implementation of retention in a coroutine environment
- Bird, 1982, An implementation of a code generator specification language for table driven code generators
- Samet, 1980, A Coroutine Approach to Parsing
- Moody and Richards, 1980, A coroutine mechanism for BCPL
- Pauli and Soffa, 1980, Coroutine behaviour and implementation
- Donegan and others, 1979, A code generator generator language
- Pritchard, 1976, A proof rule for multiple coroutine systems
- Hanson, 1976, A simple variant of the boundary-tag algorithm for the allocation of coroutine environments
- Wang and Dahl, 1971, Coroutine sequencing in a block structured environment
- DOWNEN and ARIOLA, 2014, Delimited control and computational effects
- Ilik, 2014, Proofs in continuation-passing style
- Ilik, 2013, Continuation-passing style models complete for intuitionistic logic
- Ilik, 2013, Type Directed Partial Evaluation for Level-1 Shift and Reset
- Kiselyov, 2012, Delimited control in OCaml, abstractly and concretely
- Ilik, 2012, Delimited control operators prove Double-negation Shift
- Thielecke, 2012, Functional semantics of parsing actions, and left recursion elimination as continuation passing
- Kerneis and Chroboczek, 2011, Continuation-Passing C, compiling threads to events through continuations
- Yu and Haque, 2011, Decentralised web-services orchestration with continuation-passing messaging
- TARAU, 2011, The BinProlog experience Architecture and implementation choices for continuation passing Prolog and first-class logic engines

- Kameyama and Tanaka, 2010, Equational axiomatization of call-by-name delimited control
- Garcia and others, 2010, Lazy Evaluation and Delimited Control
- Santo and others, 2009, Continuation-Passing Style and Strong Normalisation for Intuitionistic Sequent Calculi
- Masuko and Asai, 2009, Direct implementation of shift and reset in the MinCaml compiler
- Garcia and others, 2009, Lazy evaluation and delimited control
- Guerrini and Masini, 2009, Proofs, tests and continuation passing style
- Yu, 2009, Scalable Services Orchestration with Continuation-Passing Messaging
- Shan, 2007, A static simulation of dynamic delimited control
- Yu and Yang, 2007, Continuation-passing enactment of distributed recoverable workflows
- Biernacki and others, 2006, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations
- BIERNACKI and DANVY, 2006, THEORETICAL PEARL A simple proof of a folklore theorem about delimited control
- Biernacki and others, 2005, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations
- Biernacki and others, 2005, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations Preliminary Version
- Biernacki and Danvy, 2005, A Simple Proof of a Folklore Theorem about Delimited Control
- Asai, 2004, Offline partial evaluation for shift and reset
- Thielecke, 2003, From control effects to typed continuation passing
- Berdine and others, 2002, Linear Continuation-Passing
- Asai, 2002, Online partial evaluation for shift and reset
- Kelsey, 1995, A correspondence between continuation passing style and static single assignment form
- Hatcliff and Danvy, 1994, A generic account of continuation-passing styles
- Okasaki and others, 1994, Call-by-need and continuation-passing style
- De Bosschere and Tarau, 1994, High performance continuation passing style Prolog-to-C mapping
- Sabry and Felleisen, 1994, Is continuation-passing useful for data flow analysis?
- Sabry and Felleisen, 1993, Reasoning about programs in continuation-passing style
- Lawall and Danvy, 1993, Separating stages in the continuation-passing style transformation
- Appel and Shao, 1992, Callee-save registers in continuation-passing style
- Sabry and Felleisen, 1992, Reasoning about programs in continuation-passing style
- Appel and Jim, 1989, Continuation-passing, closure-passing style
- Lindley, 2014, Algebraic effects and effect handlers for idioms and arrows

- Bauer and Pretnar, 2014, An Effect System for Algebraic Effects and Handlers
- Wu and others, 2014, Effect handlers in scope
- Pretnar, 2014, Inferring Algebraic Effects
- Plotkin and Pretnar, 2013, Handling Algebraic Effects
- Ahman and Staton, 2013, Normalization by Evaluation and Algebraic Effects
- Brady, 2013, Programming and reasoning with algebraic effects and dependent types
- Johann and others, 2010, A Generic Operational Metatheory for Algebraic Effects
- Plotkin and Pretnar, 2008, A Logic for Algebraic Effects
- Hyland and others, 2007, Combining algebraic effects with continuations
- Power, 2006, The Universal Algebra of Computational Effects Lawvere Theories and Monads
- Wang and others, 2014, Register allocation for hybrid register architecture in nonvolatile processors
- Zhang and others, 2013, Register Allocation by Incremental Graph Colouring for Clustered VLIW Processors
- Tavares and others, 2011, Decoupled graph-coloring register allocation with hierarchical aliasing
- Colombet and others, 2011, Graph-coloring and treescan register allocation using repairing
- Lintzmayer and others, 2011, Register Allocation with Graph Coloring by Ant Colony Optimization
- Subha, 2010, A register allocation algorithm
- Tang and others, 2010, Balanced Bipartite Graph Based Register Allocation for Network Processors in Mobile and Wireless Networks
- Odaira and others, 2010, Coloring-based coalescing for graph coloring register allocation
- Subha, 2009, A Modified Linear Scan Register Allocation Algorithm
- Wang and others, 2009, Reducing Code Size by Graph Coloring Register Allocation and Assignment Algorithm for Mixed-Width ISA Processor
- Baev, 2009, Techniques for Region-Based Register Allocation
- Rong, 2009, Tree register allocation
- 2009, Tutorial on SSA-Based Register Allocation
- Falk, 2009, WCET-aware register allocation based on graph coloring
- Hong and Ramanujam, 2008, Address Register Allocation in Digital Signal Processors
- Mahajan and Ali, 2008, Hybrid evolutionary algorithm for graph coloring register allocation
- Guo and others, 2007, A Phase-Coupled Compiler Backend for a New VLIW Processor Architecture Using Two-step Register Allocation
- Wu and Li, 2007, Extending Traditional Graph-Coloring Register Allocation Exploiting Meta-heuristics for Embedded Systems

- Gao and Shi, 2005, An improved approach of register allocation via graph coloring
- Chase, 2005, Session details Register allocation
- Smith and others, 2004, A generalized algorithm for graph-coloring register allocation
- Chaitin, 2004, Register allocation and spilling via graph coloring
- Wall, 2004, Register windows vs. register allocation
- Boyapati, 2004, Session details Register allocation
- Park and others, 2001, Register Allocation for Banked Register File
- KIM and others, 2000, REGISTER ALLOCATION IN HYPER-BLOCK FOR EPIC PROCESSORS
- de Werra and others, 1999, On a graph-theoretical model for cyclic register allocation
- Zhou, 1996, Parameter passing and control stack management in Prolog implementation revisited
- Elof Frank, 1995, Constrained Register Allocation in Bus Architectures
- Jui-Ming Chang, 1995, Register Allocation and Binding for Low Power
- Eichenberger and Davidson, 1995, Register allocation for predicated code
- Briggs and others, 1994, Improvements to graph coloring register allocation
- Norris and Pollock, 1994, Register allocation over the program dependence graph
- Callahan and Koblenz, 1991, Register allocation via hierarchical graph coloring
- Nickerson, 1990, Graph coloring register allocation for processors with multi-register operands
- Wall, 1988, Register windows vs. register allocation
- Wall, 1986, Global register allocation at link time
- Larus and Hilfinger, 1986, Register allocation in the SPUR Lisp compiler
- Chaitin, 1982, Register allocation and spilling via graph coloring
- Sites, 1979, Machine-independent register allocation
- Kolek and others, 2013, Adding microMIPS backend to the LLVM compiler infrastructure
- MolinaFraticelli, 2012, Auto Code Generation for Simulink-Based Attitude Determination Control System
- Sanmorino, 2012, Development of computer assisted instruction CAI for compiler model The simulation of stack on code generation
- Inoue and others, 2011, A trace-based Java JIT compiler retrofitted from a method-based compiler
- Yi, 2011, Automated programmable control and parameterization of compiler optimizations
- Pałka and others, 2011, Testing an optimising compiler by generating random lambda terms
- Frenkel and others, 2011, Towards a Modular and Accessible Modelica Compiler Backend
- Myreen, 2010, Verified just-in-time compiler on x86

- Cordes and others, 2009, A Fast and Precise Static Loop Analysis Based on Abstract Interpretation, Program Slicing and Polytope Models
- Chen Zhao and others, 2009, Automated test program generation for an industrial optimizing compiler
- Böhme and others, 2009, HOL-Boogie-An Interactive Prover-Backend for the Verifying C Compiler
- Ly and others, 2009, Reduced Model for a PEMFC Stack Automated Code Generation and Verification
- Wagstaff and others, 2008, Automatic Code Generation for Instrument Flight Software
- Leroy, 2006, Formal certification of a compiler back-end or
- Surakka and others, 2005, Towards compiler backend optimization for low energy consumption at instruction level
- Klein and Strecker, 2004, Verified bytecode verification and type-certifying compilation
- Suganuma and others, 2003, A region-based compilation technique for a Java just-in-time compiler
- Yoshikawa and others, 2003, Random program generator for Java JIT compiler test system
- Mann, 2003, The Compiler Design Handbook Optimisation and Machine Code Generation
- Necula, 2000, Translation validation for an optimizing compiler
- ten Hagen and others, 1996, Codesign of a parallel architecture and an optimizing compiler backend SIN rete processing as a case study
- Bhasker, 1988, Implementation of an optimizing compiler for VHDL
- Fraser and Wendt, 1986, Integrating code generation and optimization
- Ching, 1986, Program analysis and code generation in an APL/370 compiler
- Powell, 1984, A portable optimizing compiler for Modula-2
- Karr, 1984, Code generation by coagulation
- Carter, 1982, Further analysis of code generation for a single register machine
- Hura, 1982, Optimization of assembly code generation in a compiler
- Cattell and others, 1979, Code generation in a machine-independent compiler
- Rudmik and Lee, 1979, Compiler design for efficient code generation and program optimization
- Couch and Hamm, 1977, Semantic Structures for Efficient Code Generation on a Stack Machine
- Painter, 1970, Effectiveness of an optimizing compiler for arithmetic expressions
- Larus, 2011, Session details Compiler correctness
- Chlipala, 2010, A verified compiler for an impure functional language
- Avigad and others, 2007, A formally verified proof of the prime number theorem
- Schneider and others, 2006, A Verified Compiler for Synchronous Programs with Local Declarations

- Brady and Hammond, 2006, A verified staged interpreter is a verified compiler
- Berghofer and Strecker, 2004, Extracting a formally verified, fully executable compiler from a proof assistant
- Rushby, 2002, Formally Verified Hardware Encapsulation Mechanism for Security, Integrity, and Safety
- Mohnen, 1997, A COMPILER CORRECTNESS PROOF FOR THE STATIC LINK TECHNIQUE BY MEANS OF EVOLVING ALGEBRAS
- Naik, 2013, Session details Compiler validation
- Tao and others, 2010, An Automatic Testing Approach for Compiler Based on Metamorphic Testing Technique
- Zaks and Pnueli, 2008, Program analysis for compiler validation
- Nair and Sarasamma, 2006, Formal Semantics of Ciset Relational Operators
- Kossatchev and Posypkin, 2005, Survey of compiler testing methods
- Jefferson and others, 1994, Ada Compiler Validation Summary Report Certificate Number 940902S1.11376. UNISYS Corporation IntegrAda for Windows NT, Version 1.0. Intel Deskside Server for Intel Pentium 60 MHz = . Intel Deskside Server with Intel Pentium 60 MHz
- Jefferson and others, 1994, Ada Compiler Validation Summary Report Certificate Number 940902S1.11377 UNISYS Corporation. IntegrAda for Windows NT, Version 1.0. Intel Deskside Server with Intel 80486DX266 = Intel Deskside Server with Intel 80486DX266
- VISTA RESEARCH CORP TUCSON AZ, 1994, Ada Compiler Validation Summary Report Certificate Number 940223W1. 11338 Green Hills Software, Inc. Green Hills Optimizing Ada Compiler, 1.8.7 SPARCstation 10 under SunOS, Release 4.1.3
- VISTA RESEARCH CORP TUCSON AZ, 1994, Ada Compiler Validation Summary Report Certificate Number 940305W1. 11335 TLD Systems, Ltd. TLD Comanche VAX/1960 Ada Compiler System, Version 4.1.1 VAX Cluster under VMS 5.5 = Tronix JIAWG Execution Vehicle i960MX under TLD Real Time Executive, Version 4.1.1
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1992, Ada Compiler Validation Procedures, Version 3.1
- Lehman, 1992, Ada Compiler Validation Support Fiscal Year 1991
- Hook and Lehman, 1992, Ada Compiler Validation Support Fiscal Year 1992
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1991, Ada Compiler Validation Summary Report Certificate Number 910612W1. 11168 Telesoft, IBM Ada/370, Version 1.2.0 without Optimization IBM 3080, V / SP HPO Rel 5.0 Unopt Host and Target
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1990, Ada Compiler Validation Procedures. Version 2.1
- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . ANSI
- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . ANSI, BACKUP, TAR
- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . BACKUP

- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . TAR
- Hook and Heilbrunner, 1989, Ada Compiler Validation Procedures
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Procedures Version 2.0
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report. CONVEX Computer Corporation, CONVEX Ada, Version 1.1 C210, Host and Target 890508W1.10077
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report. Meridian Software Systems, Inc., AdaVantage, Version 3.0, IBM PS/2 Model 80 with Floating Point Co-Processor Host and Target 890405W1.10049
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report. Verdix Corporation, VAda-110-2323, Version 5.5, Sequent Balance 8000 Host and Target , 890216W1.10029
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1989, Ada Compiler Validation Summary Report Certificate Number 880624S1. 09132, Control Data Corporation CYBER 180 Ada Compiler, Version 1.1 HOST and TARGET COMPUTER CYBER 180-930-31
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report Certificate Number 890329I1. 10076 SYSTEAM KG, SYSTEAM Ada Compiler VAX/VMS x MC68020/05-9 Version 1.81
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Trade Name Compiler Validation Summary Report. Certificate Number 880620W1.09092, Encore Computer Corporation, Encore Verdix Ada Development System, Version 5.5, Encore Multimax 320
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Tradename Compiler Validation Summary Report Certificate Number 880201W1.09019 Verdix Corporation VAda-010-2323, Version 5.5 Sequent Balance 8000
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Tradename Compiler Validation Summary Report Certificate Number 880429W1.09053 Telesoft, Inc. TeleGen2 Ada Compiler for VAX/VMS to 1750A, Version 3.22 MicroVAX 2 to MIL-STD-1750A ECSPO RAID Simulator
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Tradename Compiler Validation Summary Report Certificate Number 880815W1.09143 Rational VAX-VMS, Version 2.0.45 Rational R1000 Series 200 Model 20 and VAX-11/750 Host and Target
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1988, Ada Compiler Validation Summary Report Certificate Number 880708S1. 09151, SoftTech, Inc., Ada 86, Version 3.21 VAX 11/780-11/785 Host and Intel IAPX 80386R Target
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1988, Ada Compiler Validation Summary Report Certificate Number 880719S1. 09154, Naval Underwater Systems Command, ADAUYK43 ALS/N Ada/L , Version 1.0, VAX 11/785 Host and AN/UYK-43 Target
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1988, Ada Compiler Validation Summary Report Compiler Name ADE/32 Revision 3.00. Certificate Number 880527S1.09113. Host MV/20000 under AOS/VS, Revision 7.56. Target ROLM HAWK/32 under AOS/VS, Revision 7.56
- Hook, 1987, Export Control of the Ada Trade Name Compiler Validation Capability ACVC

- Dixon, 1982, A Pascal compiler testing facility
- Samet, 1977, A normal form for compiler testing
- Baird and Johnson, 1977, COBOL Compiler Validation System, 1974. Version 3.0
- Samet, 1976, Compiler testing via symbolic interpretation
- INFORMATION MANAGEMENT INC SAN FRANCISCO CA, 1970, USER'S MANUAL COBOL COMPILER VALIDATION SYSTEM
- ELECTRONIC SYSTEMS DIV HANSCOM AFB MA, 1970, USER'S MANUAL JOVIAL COMPILER VALIDATION SYSTEM
- Squire and Funkhouser, 2014, "A Bit of Code" How the Stack Overflow Community Creates Quality Postings
- Kong and others, 2014, An Overview of Worst-Case Execution Time Estimation for Embedded Programs
- Henry and others, 2014, How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics
- Debenham and Westlake, 2014, Pre-stack depth migration for improved imaging under seafloor canyons 2D case study of Browse Basin, AustraliaFN1
- Bao and others, 2014, PWCET Power-Aware Worst Case Execution Time Analysis
- Brandner and Jordan, 2014, Refinement of worst-case execution time bounds by graph pruning
- Hardy and Puaut, 2014, Static probabilistic worst case execution time estimation for architectures with faulty instruction caches
- Kong and Chen, 2013, A Worst-Case Execution Time Analysis Approach Based on AOE Networks
- - and others, 2013, Bounding the Worst-Case Execution Time for the Fixed-Priority Preemptive Systems Based on the Preemption Points
- Freier and Jian-Jia Chen, 2013, Prioritization for real-time embedded systems on dual-core platforms by exploiting the typical- and worst-case execution times
- Wu and Zhang, 2013, Reducing worst-case execution time of hybrid SPM-caches
- Kleinsorge and others, 2013, Simple analysis of partial worst-case execution paths on general control flow graphs
- Hardy and Puaut, 2013, Static probabilistic worst case execution time estimation for architectures with faulty instruction caches
- Allamanis and Sutton, 2013, Why, when, and what Analyzing Stack Overflow questions by topic, type, and code
- Wu and Zhang, 2012, A Model Checking Based Approach to Bounding Worst-Case Execution Time for Multicore Processors
- Kong and Jiang, 2012, A Worst-case execution time analysis approach based on independent paths for ARM programs
- Whiteside and others, 2012, Directional Imaging Stack DIS for Shot Based Pre-stack Depth Migrations
- Harmon and others, 2012, Fast, Interactive Worst-Case Execution Time Analysis With Back-Annotation

- Lo and Suh, 2012, Worst-case execution time analysis for parallel run-time monitoring
- Hepp and Schoeberl, 2012, Worst-Case Execution Time Based Optimization of Real-Time Java Programs
- Zhang and others, 2011, A Case Study of Pre-stack Depth Migration Application over a Salt Dome Area
- Lu and others, 2011, A new way about using statistical analysis of worst-case execution times
- Lu and others, 2011, A trace-based statistical worst-case execution time analysis of component-based real-time embedded systems
- Zolda and others, 2011, Context-Sensitive Measurement-Based Worst-Case Execution Time Estimation
- Marref, 2011, Fully-automatic derivation of exact program-flow constraints for a tighter worst-case execution-time analysis
- Louise, 2011, Improving Branch Prediction Related WCET Abstract Interpretation
- Ermedahl and Puschner, 2011, Preface to the special issue on worst-case execution-time analysis
- Wheeler and others, 2011, Video subset selection for measurement based Worst Case Execution Time analysis
- Huber and others, 2011, Worst-case execution time analysis-driven object cache design
- Falk and Lokuciejewski, 2010, A compiler framework for the reduction of worst-case execution times
- Bartlett and others, 2010, Accurate Determination of Loop Iterations for Worst-Case Execution Time Analysis
- Kirner and others, 2010, Beyond loop bounds comparing annotation languages for worst-case execution time analysis
- Ding and Zhang, 2010, Loop-Based Instruction Prefetching to Reduce the Worst-Case Execution Time
- Wang and others, 2010, Stack Bound Inference for Abstract Java Bytecode
- Lv and others, 2010, Static worst-case execution time analysis of the µC/OS-II real-time kernel
- Schoeberl and others, 2010, Worst-case execution time analysis for a Java processor
- Zhang and Yan, 2009, Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches
- Ermedahl and others, 2009, Deriving the Worst-Case Execution Time Input Values
- Berten and others, 2009, Managing Imprecise Worst Case Execution Times on DVFS Platforms
- Kirner and others, 2009, Precise Worst-Case Execution Time Analysis for Processors with Timing Anomalies
- Mitra and Givargis, 2009, Session details Microfluidics, worst-case execution time, and cache optimization
- Hanzich and others, 2009, Subsalt Imaging through Pre-Stack Depth Migration - A Case Study from the North Red Sea

- Williams and Roger, 2009, Test generation strategies to measure worst-case execution time
- Tan, 2009, The worst-case execution time tool challenge 2006
- LEE and others, 2009, Visualization and Formalization of User Constraints for Tight Estimation of Worst-Case Execution Time
- Harmon and others, 2008, A Modular Worst-case Execution Time Analysis Tool for Java Processors
- Yan and Zhang, 2008, Analyzing the worst-case execution time for instruction caches with prefetching
- Nayvelt and Bear, 2008, Case study Anisotropic Pre-Stack Depth Migration on the Louisiana Shelf
- Bate and Kazakov, 2008, New Directions in Worst-Case Execution Time analysis
- Kirner and Puschner, 2008, Obstacles in Worst-Case Execution Time Analysis
- Hunt and others, 2008, Using global data flow analysis on bytecode to aid worst case execution time analysis for real-time Java programs
- Ferdinand and Heckmann, 2008, Worst-Case Execution Time - A Tool Provider's Perspective
- Mohan, 2008, Worst-case execution time analysis of security policies for deeply embedded real-time systems
- Harmon and Klefstad, 2007, A Survey of Worst-Case Execution Time Analysis for Real-Time Java
- Ji and others, 2007, Automated Worst-Case Execution Time Analysis Based on Program Modes
- Aissa and others, 2007, Bringing Worst Case Execution Time Awareness to an Open Smart Card OS
- Nemer and others, 2007, Improving the Worst-Case Execution Time Accuracy by Inter-Task Instruction Cache Analysis
- Harmon and Klefstad, 2007, Interactive Back-annotation of Worst-case Execution Time Analysis for Java Microprocessors
- Kirner and Schoeberl, 2007, Modeling the Function Cache for Worst-Case Execution Time Analysis
- Kaestner, 2007, Safe worst-case execution time analysis by abstract interpretation of executable code
- Harmon and Klefstad, 2007, Toward a Unified Standard for Worst-Case Execution Time Annotations in Real-Time Java
- Auchterlonie and others, 2007, Velocity Model Building for Pre-Stack Depth Migration - An Onshore Libya Case Study
- Choi and Han, 2006, Optimal register reassignment for register stack overflow minimization
- Gustafsson, 2006, The Worst Case Execution Time Tool Challenge 2006
- Plasterie and Chagalov, 2006, Wave Equation versus Kirchhoff pre-stack depth migration algorithms ? An Australian case study

- Jong-In Lee and others, 2005, A Hybrid Framework of Worst-Case Execution Time Analysis for Real-Time Embedded System Software
- Egreteau and Thierry, 2005, Attenuating the Effects of Pre-Stack Depth Migration for AVA Analysis
- Ritchie and others, 2005, Challenges and opportunities in pre-stack depth imaging of legacy seismic data an overthrust belt case study
- Park and others, 2005, Implementation of Worst Case Execution Time Analysis Tool For Embedded Software based on XScale Processor
- Askim and others, 2004, 4D Seismic Analysis Using Pre Stack Depth Migration
- Schuele and Schneider, 2004, Abstraction of assembler programs for symbolic worst case execution time analysis
- Corti and Gross, 2004, Approximation of the worst-case execution time using structural analysis
- Tadepalli and others, 2003, 3D pre-stack depth imaging and well ties A case history of depth imaging in Gulf of Mexico
- Reshef and Roth, 2003, Anisotropy Corrections after Pre-Stack Depth Migration
- Ermedahl and others, 2003, Clustered calculation of worst-case execution times
- L. Freitas da Luz and C. Ribeiro Cruz, 2003, The CRS stack as a tool for pre-stack depth migration
- Engblom and others, 2003, Worst-case execution-time analysis for embedded real-time systems
- 2002, 3D Pre-Stack Depth Migration V0 Analysis and Monte Carlo Automatic Velocity Picking in Depth
- Blieberger, 2002, Data-Flow Frameworks for Worst-Case Execution Time Analysis
- Morford, 2001, Analysis Of The Effects Of Varying The Anti - Alias Filter In Kirchoff Pre-Stack Depth Migration
- Lemaistre and others, 2001, Automatic and Continuous Image Gather Analysis after Pre-Stack Depth Migration
- Hanitzsch and others, 2001, Pre-Stack Depth Migration for Time-to-Depth Conversion - the Ultimate Tool?
- Petkovski and Bradey, 2001, The success of pre-stack depth migration over the Anama structure in the Papuan Foreland Basin, PNG a case history
- Morford, 2000, A case study Using 3d pre-stack depth migration to image sub-salt sediments and fault zones in Marbella, Mexico
- Bernat and Burns, 2000, An Approach to Symbolic Worst-Case Execution Time Analysis
- Stappert and Altenbernd, 2000, Complete worst-case execution time analysis of straight-line hard real-time programs
- Allen and others, 2000, Subsalt Imaging using 3D Pre-Stack Depth Migration in the UK Southern North Sea - a Case History
- Sen and others, 2000, Velocity analysis using pre-stack depth migration and 3D tomogrphy A case study over steeply dipping salt

- Colin and Puaut, 2000, Worst Case Execution Time Analysis for a Processor with Branch Prediction
- Morford, 1999, A Case Study Using 3D Pre-Stack Depth Migration To Improve The Sub-Salt Image In Marbella, Mexico, Gulf Of Mexico
- Oates and others, 1998, The Application of Pre-Stack Depth Migration using Topographic Analysis to Aid in the
- Puschner, 1998, Worst-case execution-time analysis at low cost
- Puschner, 1997, Worst-Case Execution Time Analysis at Low Cost
- Shih and Chen, 1996, Iterative Pre-Stack Depth Migration With Velocity Analysis
- Boding, 1996, Pre-stack depth migration in three dimensions with the imaging wave machine
- Levy, 1996, Should caches be split or shared? Analysis using the superposition of bursty stack depth processes
- Roberts, 1995, 3D Post Stack Depth Migration in the UK Southern North Sea - a Case Study
- Nilsen and Rygg, 1995, Worst-case execution time analysis on modern processors
- Wenes and others, 1994, A Practical implementation of a 3D pre-stack depth migration algorithm
- Jyh-Charn Liu and Hung-Ju Lee, 1994, Deterministic upperbounds of the worst-case execution times of cached programs
- C. Robinson and others, 1994, Prospect definition by pre-stack depth migration of a grid of seismic lines - A case history
- P. Jeannot and Berranger, 1994, Ray-mapped focusing - A migration velocity analysis for Kirchoff pre-stack depth imaging
- P. Jeannot, 1994, Robust Kirchhoff pre-stack depth imaging with semi-gridded rays
- Hinkley and others, 1994, Velocity Model Building for 3D Pre-Stack Depth Migration - a Case Study
- Landa and Sorin, 1993, Fast pre-stack depth migration by CRP stacking
- R. Granli, 1993, Imaging salt with pre-stack depth migration
- Zhang and others, 1993, Pipelined processors and worst case execution times
- Cabrera and others, 1992, 3-D Pre-Stack Depth Migration Implementation and Case History
- J. Berkhout, 1992, True amplitude aspects of pre-stack depth migration
- G. Western and Ball, 1991, 3D Pre-stack depth migration in the Gulf of Sueza. A case history
- Tarjan, 1985, Amortized Computational Complexity
- Cousot and Cousot, 2014, A galois connection calculus for abstract interpretation
- Gao and others, 2014, A Method of Binary Code Variable Interval Analysis Based on Abstract Interpretation
- Ravanbakhsh and Sankaranarayanan, 2014, Infinite horizon safety controller synthesis through disjunctive polyhedral abstract interpretation

- Chudnov and others, 2014, Information Flow Monitoring as Abstract Interpretation for Relational Logic
- Kowalewski and others, 2013, Model checking and abstract interpretation as building blocks of advanced program analysis techniques
- Genaim and Zanardini, 2013, Reachability-based acyclicity analysis by Abstract Interpretation
- Ranzato, 2013, Session details Abstract interpretation
- Anderson and Loginov, 2013, Static analysis of machine code for supply-chain risk management
- Cousot and Cousot, 2012, An abstract interpretation framework for termination
- Chaumette and others, 2011, Automated extraction of polymorphic virus signatures using abstract interpretation
- Cousot and Cousot, 2011, Grammar semantics, analysis and parsing by abstract interpretation
- Hua and others, 2010, Model-Based Intrusion Detection by Abstract Interpretation
- Bertrane and others, 2010, Static Analysis and Verification of Aerospace Software by Abstract Interpretation
- Blanc and Kadobayashi, 2010, Towards revealing JavaScript program intents using abstract interpretation
- De Francesco and others, 2010, Using abstract interpretation to add type checking for interfaces in Java bytecode verification
- Giacobazzi, 2008, Abstract Interpretation in Code Security
- Bernardeschi and others, 2008, Decomposing bytecode verification by abstract interpretation
- Anier, 2008, Motion recognition with abstract interpretation and HMM
- LI, 2008, Program Verification Techniques Based on the Abstract Interpretation Theory
- Di Pierro and others, 2008, Relational Analysis and Precision via Probabilistic Abstract Interpretation
- Musumbu, 2008, Static Checking by Means of Abstract Interpretation
- Cortesi, 2008, Widening Operators for Abstract Interpretation
- Seo and others, 2007, Goal-directed weakening of abstract interpretation results
- Feret and others, 2007, Reachability Analysis of Biological Signalling Pathways by Abstract Interpretation
- 2007, The Role of Abstract Interpretation in Formal Methods
- Henriksen and Gallagher, 2006, Abstract Interpretation of PIC Programs through Logic Programming
- Christodorescu and Jha, 2006, Static Analysis of Executables to Detect Malicious Patterns
- Dalla Preda and Giacobazzi, 2005, Control code obfuscation by abstract interpretation
- Bayley and Shiel, 2005, JVM Bytecode Verification Without Dataflow Analysis

- Leuschel, 2004, A framework for the integration of partial evaluation and abstract interpretation of logic programs
- Basin and others, 2003, Bytecode Verification by Model Checking
- Cousot and Cousot, 2002, Systematic design of program transformation frameworks by abstract interpretation
- Cousot and Cousot, 2001, A Case Study in Abstract Interpretation Based Program Transformation
- Schmidt, 2000, Abstract interpretation and program modelling
- Baggett, 2000, An Abstract Interpretation of the Wavelet Dimension Function Using Group Representations
- Qian, 2000, Standard fixpoint iteration for Java bytecode verification
- Cousot and Cousot, 2000, Temporal abstract interpretation
- Huch, 1999, Verification of Erlang programs using abstract interpretation and model checking
- Dams and others, 1997, Abstract interpretation of reactive systems
- Cortesi and others, 1997, Complementation in abstract interpretation
- Debray, 1995, Abstract interpretation and low-level code optimization
- Cousot and Cousot, 1995, Formal language, grammar and set-constraint-based program analysis by abstract interpretation
- Deutsch, 1995, Semantic models and abstract interpretation techniques for inductive data structures and pointers
- Monsuez, 1995, Using abstract interpretation to define a strictness type inference system
- Marriott and others, 1994, Denotational abstract interpretation of logic programs
- Le Charlier and Van Hentenryck, 1994, Experimental evaluation of a generic abstract interpretation algorithm for PROLOG
- Mycroft, 1993, Completeness and predicate-based abstract interpretation
- Coppo and Ferrari, 1993, Type inference, abstract interpretation and strictness analysis
- Muller and Zhou, 1992, Abstract interpretation in weak powerdomains
- Palsberg and Schwartzbach, 1992, Binding Time Analysis Abstract Interpretation vs. Type Inference
- Janssens and Bruynooghe, 1992, Deriving descriptions of possible values of program variables by means of abstract interpretation
- Cortesi and Filé, 1991, Abstract interpretation of logic programs
- McNerney, 1991, Verifying the correctness of compiler transformations on basic blocks using abstract interpretation
- Burn, 1990, A relationship between abstract interpretation and projection analysis
- Nielson, 1988, Strictness analysis and denotational abstract interpretation
- Nielson, 1987, Strictness analysis and denotational abstract interpretation
- Hu and Zhao, 2014, Analysis on Process Code schedule of Android Dalvik Virtual Machine

- Mamdouh and others, 2014, On-demand distributed on-card bytecode verification
- Jiang and Li, 2014, Using Contour Marking Bytecode Verification Algorithm on the Java Card
- You and Lu, 2012, A markup language for java bytecode
- Kim and others, 2012, Generating Verification Conditions from BIRS Code using Basic Paths for Java Bytecode Verification
- Santone, 2011, Clone detection through process algebras and Java bytecode
- Male and others, 2011, Formalisation and implementation of an algorithm for bytecode verification of @NonNull types
- Haikun Liu and others, 2011, Live Virtual Machine Migration via Asynchronous Replication and State Synchronization
- DONG and others, 2011, Logic System for Bytecode Program Modular Certification
- Moret and others, 2011, Polymorphic bytecode instrumentation
- Bauml and Brada, 2011, Reconstruction of Type Information from Java Bytecode for Component Compatibility
- Chen and others, 2010, Implementation of Bytecode-based Software Watermarking for Java Programs
- 2010, JSIMIL - A Java Bytecode Clone Detector
- Rudolph and Thiemann, 2010, Mnemonics type-safe bytecode generation at run time
- DeVries and others, 2009, ActionScript bytecode verification with co-logic programming
- DeVries and others, 2009, ActionScript bytecode verification with co-logic programming abstract only
- Chi and others, 2009, An Improved Bytecode Verification Algorithm on Java Card
- Chen and others, 2009, Bytecode Generation for XQuery Compiler
- Gal and others, 2008, Java bytecode verification via static single assignment form
- Bavera and Bonelli, 2008, Type-based information flow analysis for bytecode languages with variable object field policies
- Shi and others, 2008, Virtual machine showdown
- Liu, 2007, Bytecode Verification for Enhanced JVM Access Control
- Huynh and Roychoudhury, 2007, Memory model sensitive bytecode verification
- Huang and others, 2006, Adaptiveness in well-typed Java bytecode verification
- Burdy and Pavlova, 2006, Java bytecode specification and verification
- Bernardeschi and others, 2006, Using Control Dependencies for Space-Aware Bytecode Verification
- Gal and others, 2005, Integrated Java Bytecode Verification
- Kot and Kozen, 2005, Kleene Algebra and Bytecode Verification
- Klohs and Kastens, 2005, Memory Requirements of Java Bytecode Verification on Limited Devices
- Hansen and Siveroni, 2005, Towards Verification of Well-Formed Transactions in Java Card Bytecode

- Klein, 2005, Verified Java Bytecode Verification Verified Java Bytecode Verification
- Bernardeschi and others, 2004, Checking secure information flow in Java bytecode by code transformation and standard bytecode verification
- Peng and others, 2004, Code sharing among states for stack-caching interpreter
- Barbuti and Cataudella, 2004, Java bytecode verification on Java cards
- Siveroni, 2004, Operational semantics of the Java Card Virtual Machine
- Coglio, 2004, Simple verification technique for complex Java bytecode subroutines
- Freund and Mitchell, 2003, A Type System for the Java Bytecode Language and Verifier
- Coglio, 2003, Improving the official specification of Java bytecode verification
- Nipkow, 2003, Java Bytecode Verification
- Avvenuti and others, 2003, Java bytecode verification for secure information flow
- Leroy, 2003, Java Bytecode Verification Algorithms and Formalizations
- Rose, 2003, Lightweight Bytecode Verification
- Leroy, 2002, Bytecode verification on Java smart cards
- Barbuti and others, 2002, Fixing the Java bytecode verifier by a suitable type domain
- Knoblock and Rehof, 2001, Type elaboration and subtype completion for Java bytecode
- Klein and Nipkow, 2001, Verified lightweight bytecode verification
- Knoblock and Rehof, 2000, Type elaboration and subtype completion for Java bytecode
- O'Callahan, 1999, A simple, comprehensive type system for Java bytecode subroutines
- Stata and Abadi, 1999, A type system for Java bytecode subroutines
- Freund and Mitchell, 1999, A type system for object initialization in the Java bytecode language
- Goldberg, 1998, A specification of Java loading and bytecode verification
- Stata and Abadi, 1998, A type system for Java bytecode subroutines
- Freund and Mitchell, 1998, A type system for object initialization in the Java bytecode language
- Freund and Mitchell, 1998, A Type System for Object Initialization In the Java Bytecode Language summary
- Shih, 1998, An operational semantic approach to continuation style interpreter of logic programs
- Hoskins, 1988, The design and implementation of a Karel compiler and interpreter
- Gundersen and others, 2013, Atomic Lambda Calculus A Typed Lambda-Calculus with Explicit Sharing
- Liu and others, 2013, Linking Algebraic Semantics and Operational Semantics for Web Services Using Maude
- Zhu and others, 2012, Linking operational semantics and algebraic semantics for a probabilistic timed shared-variable language
- Wang and Zhu, 2011, Animating the Approach of Deriving Operational Semantics from Algebraic Semantics for Web Services

- Colvin and Hayes, 2011, Structural operational semantics through context-dependent behaviour
- Zhu and others, 2009, Animating the Link Between Operational Semantics and Algebraic Semantics for a Probabilistic Timed Shared-Variable Language
- Vu, 2008, Denotational semantics for thread algebra
- Zhu and others, 2008, From algebraic semantics to denotational semantics for Verilog
- Yang and Duan, 2008, Operational semantics of Framed Tempura
- Zhu and others, 2007, Algebraic Approach to Operational Semantics and Observation-Oriented Semantics for a Timed Shared-Variable Language with Probability
- Edalat and Pattinson, 2007, Denotational semantics of hybrid automata
- Verdejo and Martí-Oliet, 2006, Executable structural operational semantics in Maude
- 2004, A structural approach to operational semantics
- Popeea and Chin, 2004, A type system for resource protocol verification and its correctness proof
- Aceto and Fokkink, 2004, Guesteditors'introduction Specialissueon Structural Operational Semantics
- Mosses, 2004, Modular structural operational semantics
- PLOTKIN, 2004, The origins of structural operational semantics
- Zhu, 2001, Denotational semantics of programming languages and compiler generation in PowerEpsilon
- Puchol and others, 1998, An Operational Semantics and Compiler for Real-Time Specifications1
- Royer, 1986, Transformations of denotational semantics in semantics directed compiler generation
- Mazaher and Berry, 1985, Deriving a compiler from an operational semantics written in VDL
- Clinger, 1984, The scheme 311 compiler an exercise in denotational semantics
- Raskovsky, 1982, Denotational semantics as a specification of code generators
- Bodwin and others, 1982, Experience with an experimental compiler generator based on denotational semantics
- Jones, 1980, Compiler Generation from Denotational Semantics
- Zhu and others, 2009, Locality-Based Normal Form Approach to Linking Algebraic Semantics and Operational Semantics for an Event-Driven System-Level Language
- Shukla and others, 2014, A Formal Approach to the Provably Correct Synthesis of Mission Critical Embedded Software for Multi Core Embedded Platforms
- Son and others, 2014, A Reversing Technique for Symbol Table Verification on Compiler Constructions
- Wang and Yang, 2014, Applied Technology in Front-End Implementation of Tiger Compiler Using Hacs Language
- Pathiran and Prakash, 2014, Design and implementation of a model-based PI-like control scheme in a reset configuration for stable single-loop systems

- Matsikoudis and Stergiou, 2014, First Draft of the act Programming Language
- Xu and Zhang, 2014, Formal verification of software safety criteria using Event-B
- Poonguzhali and Vinodha, 2014, IMPLEMENTATION OF ANTI-RESET WINDUP SCHEME IN PI CONTROLLER FOR SPHERICAL TANK PROCESS
- Balu and Saraswathi, 2014, Implementation of SAAS Compiler in Intranet
- Krebs and Schmitz, 2014, Jaccie A Java-based compiler-compiler for generating, visualizing and debugging compiler components
- Odegard and others, 2014, Model-Based GN and C Simulation and Flight Software Development for Orion Missions beyond LEO
- Radonić and others, 2014, One solution of loop invariant code motion compiler optimisation
- Keaton and Seacord, 2014, Performance of Compiler-Assisted Memory Safety Checking
- Abdulsalam and others, 2014, Program energy efficiency The impact of language, compiler and implementation choices
- Radooevii and Magdalenii, 2014, Python Implementation of Source Code Generator Based on Dynamic Frames
- Hussain and others, 2014, RUGRAT Evaluating program analysis and testing tools and compilers with large generated random benchmark applications
- Tardieu, 2014, Session details Compiler optimizations
- Boldo and others, 2013, A Formally-Verified C Compiler Supporting Floating-Point Arithmetic
- Kistel and Vandenhouten, 2013, A metamodel-based ASN.1 editor and compiler for the implementation of communication protocols
- Saraf and Dashora, 2013, An Optimal Code Heuristic Approach for Compiler Optimization using Graph Coloring Technique
- Gorringe and Jain, 2013, Architectural considerations for implementation of the ATML standards in an open systems architecture runtime environment OSA-RTS using a graphical environment
- Dong, 2013, Design and Implementation of Compiler Subsystem for Object Oriented Publish/Subscribe Systems
- Schwaab and Siek, 2013, Modular type-safety proofs in Agda
- Barash and Shchur, 2013, RNGSSELIB Program library for random number generation. More generators, parallel streams of random numbers and Fortran compatibility
- Chatterjee, 2013, Runtime Systems for Extreme Scale Platforms
- Dejtrakulwong and others, 2013, Sensitivity analysis and cascaded interpretation scheme for subtle seismic signatures in thin shaly-sand reservoirs
- Bond, 2013, Session details Garbage collection, runtime, and cache management
- 2012, A Compositional Scheme and Framework for Safety Critical Systems Verification
- Nair, 2012, A Formal Semantics for Ciset and Ciset Relation Operators
- Pike and others, 2012, Experience Report A Do-It-Yourself High-Assurance Compiler

- Grechanik, 2012, Random benchmark application generation for evaluating program analysis and testing tools
- Aung and others, 2011, Compiler-assisted technique for rapid performance estimation of FPGA-based processors
- Schliephake and others, 2011, Design and Implementation of a Runtime System for Parallel Numerical Simulations on Large-Scale Clusters
- Agathos and others, 2011, Design and Implementation of OpenMP Tasks in the OMPi Compiler
- Bansal and Singh, 2011, Dual Stack Implementation of Mobile IPv6 Software Architecture
- Magdalenić and others, 2011, Implementation Model of Source Code Generator
- Zaitsev and Guliaiev, 2011, Stack E6 and Its Implementation within Linux Kernel
- Larkins and Jones, 2011, Targeting FPGA-based processors for an implementation-driven compiler construction course
- Snavely, 2011, Test and Evaluation of Architecture-Aware Compiler Environment
- Jiang and others, 2010, A Debugging Approach for Java Runtime Exceptions Based on Program Slicing and Stack Traces
- Eachempati and others, 2010, An open-source compiler and runtime implementation for Coarray Fortran
- Manjunath, 2010, Colored Steganography Implementation Scheme of Images using Transform Technique
- Chalin, 2010, Engineering a Sound Assertion Semantics for the Verifying Compiler
- Rosenblum and others, 2010, Extracting compiler provenance from program binaries
- Orlic, 2010, Implementation by capture with executable UML
- Jiang and Yan, 2010, Implementation of Static Web-Pages Generator Using JavaScript
- Desai, 2009, A Novel Technique for Orchestration of Compiler Optimization Functions Using Branch and Bound Strategy
- Rodríguez and others, 2009, CPPC a compiler-assisted tool for portable checkpointing of message-passing applications
- Canedo and others, 2009, Design and implementation of a queue compiler
- Denney and Fischer, 2009, Generating Code Review Documentation for Auto-Generated Mission-Critical Software
- Hiroyuki, 2009, Idiom Recognition and Program Scheme Recognition Based Program Transformations for Performance Tuning–Beyond Compiler Optimizations
- Oiwa, 2009, Implementation of the memory-safe full ANSI-C compiler
- Pasareanu and others, 2009, Model Based Analysis and Test Generation for Flight Software
- Seshia and Rakhlin, 2009, Quantitative Analysis of Embedded Software Using Game-Theoretic Learning
- Reb and others, 2008, A JML Compiler Based on AspectJ
- Bushnell and others, 2008, Automatic Testcase Generation for Flight Software

- YU, 2008, Design and implementation of NC code compiler based on ANTLR
- Leinenbach and Petrova, 2008, Pervasive Compiler Verification From Verified Programs to Verified Systems
- Zanardini, 2008, The Semantics of Abstract Program Slicing
- Lucchi and Mazzara, 2007, A pi-calculus based semantics for WS-BPEL
- Chalin, 2007, A Sound Assertion Semantics for the Dependable Systems Evolution Verifying Compiler
- Bohnet and Dollner, 2007, CGA Call Graph Analyzer - Locating and Understanding Functionality within the Gnu Compiler Collection's Million Lines of Code
- Chen and others, 2007, Design of a Certifying Compiler Supporting Proof of Program Safety
- TOKUMORI and others, 2007, Development of Metadata Generation Support System Using Compiler Compiler
- de Niz, 2007, Diagrams and Languages for Model-Based Software Engineering of Embedded Systems UML and AADL
- Dubach and others, 2007, Fast compiler optimisation evaluation using code-feature based performance prediction
- Costagliola and others, 2007, Visual language implementation through standard compiler-compiler techniques
- Heimbigner, 2006, A Tamper-Resistant Programming Language System
- Butterfield and Woodcock, 2006, A "Hardware Compiler" Semantics for Handel-C
- Azeemi, 2006, Compiler Directed Battery-Aware Implementation of Mobile Applications
- Bicarregui and others, 2006, The verified software repository a step towards the verifying compiler
- Sadat, 2005, A Compiler Driven Simulation Technique for the Analysis of Digital Logic Circuit
- Guyer and Lin, 2005, Broadway A Compiler for Exploiting the Domain-Specific Semantics of Software Libraries
- 2005, Fifth IEEE International Workshop on Source Code Analysis and Manipulation
- Denney and Fischer, 2005, Formal Safety Certification of Aerospace Software
- Xia and DiVito, 2005, Software Certification for Temporal Properties With Affordable Tool Qualification
- Shin and others, 2004, AIRES Automatic Integration of Reusable Embedded Software, Methodologies, Toolkit, and Experiments
- 2004, Fourth IEEE International Workshop on Source Code Analysis and Manipulation
- Amarasinghe, 2004, Session details Compiler and simulator construction
- Necula and Lee, 2004, The design and implementation of a certifying compiler
- Pla, 2004, Weapon System Software Technology Support WSSTS . Delivery Order 0008 Real-Time Java for Embedded Systems RTJES
- Dold and others, 2003, A Completely Verified Realistic Bootstrap Compiler

- 2003, ART An Implementation on the Active_object RunTime Systems Applicable for the Embedded Systems
- 2003, Proceedings Third IEEE International Workshop on Source Code Analysis and Manipulation
- Uh, 2003, Session details Compiler optimizations
- Hsu and Kremer, 2003, The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction
- League, 2002, A Type-Preserving Compiler Infrastructure
- Goos, 2002, Compiler Verification and Compiler Architecture
- 2002, Proceedings Second IEEE International Workshop on Source Code Analysis and Manipulation
- Orlitsky, 2002, Scalar versus vector quantization worst case analysis
- Kandemir, 2001, A compiler technique for improving whole-program locality
- Zendra and Colnet, 2001, Coping with aliasing in the GNU Eiffel Compiler implementation
- Kennedy and Syme, 2001, Design and implementation of generics for the .NET Common language runtime
- MacMillen, 2001, Nimble Compiler Environment for Agile Hardware. Volume 1
- Calzarossa and others, 2001, Performance issues of an HPF-like compiler
- Auguston and others, 2001, Visual Meta-Programming Language
- Lin and Padua, 2000, Compiler analysis of irregular memory accesses
- Havelund and others, 2000, Formal Analysis of the Remote Agent Before and After Flight
- Orr and Henderson, 2000, Space Shuttle Software Development and Certification
- Frigo, 1999, A fast Fourier transform compiler
- van Deursen, 1999, Modern Compiler Implementation in Java
- Nielsen, 1998, Compiler Support for Message Passing Systems
- 1998, Compiler technology Tools, translators and language implementation
- Stöhr and O'Boyle, 1998, First Fast Sink A compiler algorithm for barrier placement optimisation
- 1998, Modern compiler implementation in Java Revised and expanded edition
- Haspert and Beauregard, 1998, The Commercialization of a Rapid Prototyping Development Tool for Real-Time Embedded Software Intensive, Process, and Resource Management Systems
- Necula and Lee, 1998, The design and implementation of a certifying compiler
- Nielsen, 1997, Compiler Support for Message Passing Systems
- Stoonkisto and Subhlok, 1997, Coordinating Foreign Modules with a Parallelizing Compiler
- Kelly, 1997, Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems Volume II A Practitioner's Companion
- 1997, Modern compiler implementation in C Basic techniques

- 1997, Modern compiler implementation in Java Basic techniques
- 1997, Modern compiler implementation in ML Basic techniques
- Cousot, 1997, Program analysis
- Pratt, 1997, Second Calculus of Binary Relations as a Concurrent Programming Language
- Badler, 1996, A Task Networking and Visual Programming Language for Jack
- Ferrante and Allard, 1996, Introducing a CPS style optimizer into an existing compiler
- Cousot, 1996, Program analysis
- Evansi and Sulaiman, 1996, Solving optimisation problems using neucomp-a neural network compiler
- Porcher, 1995, Benchmarking the POMPC compiler on the Connection Machine CM-2
- 1995, Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems A Practitioner's Companion - Volume 2
- 1995, Formal Methods Specification and Verification Guidebook for Software and Computer Systems Planning and Technology Insertion - Volume 1
- Graham, 1995, Information Technology. Programming Language. The SQL Ada Module Description Language SAMeDL
- Davenport, 1995, Object-Oriented Visual Programming Language. Phase 1
- Ertl, 1995, Stack caching for interpreters
- Oliva and others, 1995, The VLISP verified PreScheme compiler
- Brodersen, 1994, Anatomy of a Silicon Compiler
- Chi, 1994, Compiler Optimization Technique for Data Cache Prefetching Using a Small CAM Array
- Atapattu, 1994, Design of a Parallel Object Oriented Programming Language
- Surati, 1993, A Parallelizing Compiler Based on Partial Evaluation
- Rogers, 1993, Ada Embedded Computer Software Support AECSS
- Plishka and Ifarraguerri, 1993, Evaluation of Alsys 037 Ada Compiler
- Butler and Johnson, 1993, Formal Methods for Life-Critical Software
- Bailin and others, 1993, Model-based reasoning for system and software engineering The Knowledge From Pictures KFP environment
- Gaál, 1993, Parallel compiler generation
- Herring and others, 1993, Research in Presistent Simulation Development of the Persistent ModSim Object-Oriented Programming Language
- Consel and Cheng Khoo, 1993, Semantics-directed generation of a prolog compiler
- Ching and Katz, 1993, The testing of an APL compiler
- Mary-Anne K Posenau, 1993, Unstructured Grid Generation Techniques and Software
- Hoang and Rabaey, 1992, A compiler for multiprocessor DSP implementation
- Harper and Pfenning, 1992, A Module System for a Programming Language Based on the LF Logical Framework

- Russinoff, 1992, A verified prolog compiler for the Warren Abstract Machine
- Ooashi and others, 1992, ASL program written in abstract sequential machine style and its compiler
- Palsberg, 1992, Provably Correct Compiler Generation
- Gifford and others, 1992, Report on the FX-91 Programming Language
- Schmitz, 1992, The visual compiler-compiler SIC abstract
- Leavitt and Terrell, 1991, Ada Compiler Evaluation Capability
- Leavitt and Terrell, 1991, ADA Compiler Evaluation Capability User's Guide, Release 2.0
- Leavitt and Terrell, 1991, Ada Compiler Evaluation Capability. Release 2.0
- Hird, 1991, Formal specification and verification of Ada software
- Lane and Poorman, 1991, Preserving software investment using new fortran compiler technology
- Shivers, 1991, The semantics of Scheme control-flow analysis
- Ramkumar and Kale, 1990, A Chare kernel implementation of a parallel Prolog compiler
- Wilson, 1990, Ada/Ed Compiler, Version 1.10 UNIX
- Wilson, 1990, Ada/Ed Compiler, Version 1.10 VAX
- Gupta, 1990, An Incremental Type Inference System for the Programming Language Id
- Heuring and others, 1990, Automatic Compiler Construction
- Liangliang and Yungui, 1990, Clause representations in a compiler-based prolog database
- Giorgi and Le Métayer, 1990, Continuation-based parallel implementation of functional programming languages
- Sharp, 1990, Pythia A Parallel Compiler for Delirium
- Rosing and others, 1990, The DINO Parallel Programming Language
- Woronow, 1989, Correction for a "FORTRAN program for generation of multivariate normally distributed random variables"
- Harrison, 1989, Research, Development, Training and Education Using the Ada Programming Language
- Vegdahl and Pleban, 1989, The runtime environment for Scheme, a Scheme implementation on the 88000
- Horwat, 1988, A Concurrent Smalltalk Compiler for the Message-Driven Processor
- Weiner and Ramakrishman, 1988, A piggy-back compiler for Prolog
- Harmon, 1988, An Ada implementation of Marsaglia's universal random number generator
- Tuck, 1988, An Optimally Portable SIMD Single-Instruction Multiple-Data Programming Language
- Keutzer and Wolf, 1988, Anatomy of a hardware compiler
- Andrews and others, 1988, Design and implementation of the UW Illustrated compiler

- Sinharoy, 1988, EPL - Equational Programming Language Parsing and Dimension Propagation
- Lehman and others, 1988, Sources of Compiler Capability Information in Validation Summary Reports
- Manna, 1988, TABLOG The Deductive Tableau Programming Language
- Ghosh and Kulatilake, 1987, A FORTRAN program for generation of multivariate normally distributed random variables
- Lee and Pleban, 1987, A realistic compiler generator based on high-level semantics another progress report
- Donohoe, 1987, A Survey of Real-Time Performance Benchmarks for the Ada Programming Language
- Bonar and Liffick, 1987, A Visual Programming Language for Novices
- Kingsley, 1987, The implementation of a state machine compiler
- Guarna and Jr, 1987, VPC - A Proposal for a Vector Parallel C Programming Language
- Scott, 1986, The Interface Between Distributed Operating System and High-Level Programming Language. Revision
- Sager, 1985, A technique for creating small fast compiler frontends
- Grover, 1985, Guidelines for a Minimal Ada Runtime Environment
- Touzeau, 1984, A Fortran compiler for the FPS-164 scientific computer
- Schmidt and Völler, 1984, A multi-language compiler system with automatically generated codegenerators
- Howe, 1984, A Study of the Feasibility of Duplicating JAMPS Applications Software in the Ada Programming Language
- Blower, 1984, An efficient implementation of visibility in Ada
- Pleban, 1984, Compiler prototyping using formal semantics
- Milos and others, 1984, Direct implementation of compiler specifications or the pascal p-code compiler revisited
- Mössenböck, 1984, Ein einfacher Compiler-Compiler für Mikrocomputer / A simple compiler-compiler for microcomputer
- Robbins, 1984, Engineering a high-capacity Pascal compiler for high performance
- Aigrain and others, 1984, Experience with a Graham-Glanville style code generator
- Christopher and others, 1984, Using dynamic programming to generate optimized code in a Graham-Glanville style code generator
- Schmeck, 1983, Algebraic semantics of recursive flowchart schemes
- CARNEY and LABAUGH, 1983, Efficient compiler implementation for a spaceborne image processing demonstration system
- Reiss, 1983, Generation of Compiler Symbol Processing Mechanisms from Specifications
- Paulson, 1982, A semantics-directed compiler generator
- Ganzinger and others, 1982, A truly generative semantics-directed compiler generator
- Moor, 1982, An applicative compiler for a parallel machine

- Auslander and Hopkins, 1982, An overview of the PL.8 compiler
- Fusaoka and Hirayama, 1982, Compiler chip
- Koskimies and others, 1982, Compiler construction using attribute grammars
- Sethi, 1982, Control flow aspects of semantics directed compiling Summary
- Falis, 1982, Design and implementation in Ada of a runtime task supervisor
- Kipps, 1982, Experience with porting techniques on a COBOL 74 compiler
- Gallaher, 1982, Investigate Capability of Ada Higher Order Programming Language for Developing Machine Independent Software
- Seyfer, 1982, Tailoring testing to a specific compiler—experiences
- Marshall, 1982, The linear graph package, a compiler building environment
- Jones and Christiansen, 1981, Control Flow Treatment in a Simple Semantics-Directed Compiler Generator
- Hart and McClanahan, 1981, JOVIAL J73 Compiler Validator
- Loy, 1981, Notes on the Implementation of MUSBOX A Compiler for the Systems Concepts Digital Synthesizer
- 1981, Ruggedized minicomputer hardware and software topics, 1981 Proceedings of the 4th ROLM MIL-SPEC Computer User's Group Conference
- Leverett and others, 1979, An Overview of the Production Quality Compiler-Compiler Project
- Bonyun, 1979, Euclid Compiler for PDP-11
- Feldman, 1979, Implementation of a portable Fortran 77 compiler using modern tools
- Bonkowski and others, 1979, Porting the Zed compiler
- Guessarian, 1979, Program transformations and algebraic semantics
- Deransart, 1979, Proof by semantic attributes of a LISP compiler
- Abrahams, 1979, The CIMS PL/I compiler
- Pleban, 1979, The use of transition matrices in a recursive-descent compiler
- Evans and others, 1978, A Compiler Compiler and Methodology for Problem Oriented Language Compiler Implementors
- Payne, 1978, A formalised technique for expressing compiler exercisers
- Bonyun and Holt, 1978, EUCLID Compiler for PDP-11
- Lynn, 1978, Interactive Compiler Proving Using Hoare Proof Rules
- Johnson, 1978, Tools For Automatic Compiler Generation Panel Discussion
- Williams and Bulmer, 1978, Use of a formal notation for static semantics in compiler design
- Baird and Oliver, 1977, Programming Language Standards – Who Needs Them
- Lange and others, 1977, Specification for a STARAN Programming Language
- Fisher, 1976, A Common Programming Language for the Department of Defense–Background and Technical Requirements

- Krzemień and Lukasiewicz, 1976, Automatic generation of lexical analyzers in a compiler-compiler
- Goodenough and others, 1976, Evaluation of ALGOL 68, JOVIAL J3B, PASCAL, SIMULA 67, and TACPOL vs. TINMAN Requirements for a Common High Order Programming Language
- Ganzinger and others, 1976, MUG1 - an incremental compiler-compiler
- Harrison and others, 1976, Theoretical results in compiler design and implementation Tutorial Session
- Snyder, 1975, A Portable Compiler for the Language C
- Newey, 1975, Formal Sematics of LISP with Applications to Program Correctness
- Gorelik and Khukhlaev, 1975, Implementation of the incremental fortran compiler
- Zelkowitz, 1975, Third generation compiler design
- Laliotis, 1973, Implementation aspects of the symbol hardware compiler
- 1973, Implementation of a Pascal compiler for the CII IRIS 80 computer
- Malcolm, 1971, PL360 Revised . A Programming Language for the IBM360
- Cowan and Graham, 1970, Design characteristics of the WATFOR compiler
- Cheatham and Standish, 1970, Optimization aspects of compiler- compilers
- Finkelstein, 1968, A compiler optimization technique
- Moore, 1968, Data Processing with the Compiler Compiler
- Pankhurst, 1968, GULP—A compiler-compiler for verbal and graphic languages
- Bayer and others, 1968, MPL MATHEMATICAL PROGRAMMING LANGUAGE
- Trout, 1967, A compiler—compiler system
- Mondshein, 1967, VITAL COMPILER SYSTEM REFERENCE MANUAL
- Feldman, 1966, A formal semantics for computer languages and its application in a compiler-compiler
- Campbell and Beck, 1965, THE FORAST PROGRAMMING LANGUAGE FOR ORDVAC AND BRLESC REVISED
- BOOK and others, 1963, A ONE PASS JOVIAL COMPILER
- KELLY, 1963, ADVANCED MYSTIC- A COMPILER FOR MANAGEMENT CONTROL OF COMPUTER PROGRAMMING
- Jervis, 1963, MOBILE. A MOBIDIC COBOL COMPILER

The surrounding systems literature

The harvest returned a large body of work that is adjacent rather than central, being compiler, runtime and systems research that shares the article's vocabulary without addressing its question. **It is listed rather than discarded because the selection that produced it is reported in full**, and a survey that presents only the records supporting its thesis has selected twice, once by query and once by judgement, while reporting one selection.

- Kuroda and Yuen, 2026, A Concurrent Extension of Reversible Imperative Programming Language with Runtime

- Attrot and others, 2026, A Pattern Generation Language for MLIR Compiler Matching and Rewriting
- bounpaserth, 2026, A Study of the Computer Programming Language Implementation in Computer Engineering Students using the Flowgorithm Platform versus Common Programming
- Sun and Staron, 2026, Agentic Pipelines in Embedded Software Engineering Emerging Practices and Challenges
- Iida and others, 2026, An Efficient Runtime Verification Toolkit for Self-Adaptive Systems Addressing Runtime System Model Changes
- Jadhav and others, 2026, Analysis of Compiler-Level Static and Dynamic Features for Automated Bug Prediction Using Transformer Models
- Aisyiyah and Eviyanti, 2026, Android-based Programming Language to Natural Language Translator App
- Qin and others, 2026, Augmenting LLM Code Translation with Compiler Analysis for C to Triton Kernel Generation
- Kahn and others, 2026, Big-Stop Semantics Small-Step Semantics in a Big-Step Judgment
- Kasaraneni and Nandivada, 2026, Compact Representation and Interleaved Solving for Scalable Constraint-Based Points-to Analysis
- Cheng and others, 2026, Denotation-based Compositional Compiler Verification
- Deng and others, 2026, Design and Implementation of an OCaml-Based Standalone SystemVerilog Preprocessor Compliant with IEEE 1800-2023
- Altan, 2026, Error-Resilient Quantum Compiler Design for Efficient Qubit Mapping, Gate Optimization, and Noise Mitigation in NISQ-Era Devices
- Wu and others, 2026, Fluctuation-guided adaptive random compiler for Hamiltonian simulation
- Arakaki and Hirokawa, 2026, Foundational Design of Multi-Modal Typed Programming Language for Quantum-Classical Hybrid Computing System
- Duţu and others, 2026, From Runtime Reflection to Compile-Time Specialization A Template-Based Approach to Runtime Libraries
- Jackson, 2026, I/O Optimisation at the Compiler Level IOOpt
- Lai and others, 2026, Interaction-aware multi-objective optimization method for LLVM compiler option sequences
- Xiang and others, 2026, LoopHint A Compiler-Assisted Loop Branch Predictor for Embedded DSPs
- 2026, NOCI-COMPILER A THEORETICAL ARCHITECTURE FOR THE SEMANTIC TRANSLATION OF NOCICEPTIVE SIGNALS INTO A DIGITAL PAIN ALPHABET
- Kishorbhai and Patel, 2026, Online Code Compiler A Modern Web Based Programming Platform
- Du, 2026, Performance Verification of BFS For Unweighted Maze Solving A Comparative Analysis with DFS and A* Via Ocaml Implementation
- Dong and others, 2026, Presynthesis Towards Scaling Up Program Synthesis with Finer-Grained Abstract Semantics
- Aziz and Labiche, 2026, ProtoSYCL A Sample Implementation of a SYCL Compiler for Conformance Test Suite Development
- Wang and others, 2026, Random test generators demystified Differences and potential for compiler reliability
- Maity and Ghose, 2026, SAGE A Compiler-assisted Reinforcement Learning-based Offloading Approach under Near-memory Processing Paradigm
- 2026, Syntax-Directed Semantics in Programming Language Design
- Arriaga and others, 2026, Tempo An ML-KEM to PAKE Compiler Resilient to Timing Attacks
- Avigad and others, 2025, A Proof-Producing Compiler for Blockchain Applications

- Petchartee, 2025, A Universal Quantum Compiler GPT Multi-Framework Optimization and Translation Using Large Language Models
- Delaët and others, 2025, Abstract machines and small-step semantics a winning ticket for proof automation?
- Fan and others, 2025, Adaptive random compiler for Hamiltonian simulation
- Pasupuleti, 2025, AI-Guided Quantum Compiler Design Using Superalgebraic Symmetries
- Bai and others, 2025, APCer An Agile Physical Compiler for Multi-Port Register File
- Liu and Lu, 2025, Bi-directional Taint Flow Analysis A High-precision Static Detection Approach for Java Deserialization Vulnerabilities
- Heo and others, 2025, Bit-level compiler optimization for ultra low-power embedded systems
- Puranik, 2025, Bridging Formal Methods and Software Engineering Through a Tagless-Final Embedded DSL for Program Semantics
- Zhang, 2025, Comparative Implementation of Binary Tree and Recursive Backtracking Maze Generation Algorithms in OCaml
- Pandey, 2025, Compiler Design and Its Construction
- Wu and others, 2025, Compiler Optimization Testing Based on Optimization-Guided Equivalence Transformations
- Matteo R. Donelli, 2025, Compiler-Assisted Optimization Using Neural Code Embeddings for Heterogeneous Architectures
- Jadhav and Falk, 2025, Compiler-level DMA-aware multi-objective dynamic SPM allocation
- Astarte, 2025, Conceptualising Programming Language Semantics
- Chen and others, 2025, De-duplicating Silent Compiler Bugs via Deep Semantic Representation
- Hensley and Elgazzar, 2025, DESIGN AND IMPLEMENTATION OF THE MOREHEAD-AZALEA COMPILER MAC
- Ghuzdewan, 2025, Development of a Python-Based Program for Pareto Analysis in Construction Project Cost Management
- He and others, 2025, Evaluating Program Semantics Reasoning with Type Inference in System F
- Seassau and others, 2025, Formal Semantics and Program Logics for a Fragment of OCaml
- Yang and others, 2025, Formal Verification of a Custom Compiler for a Fully Homomorphic Encryption Accelerator
- He and Zhong, 2025, From Bug Reports to Workarounds The Real-World Impact of Compiler Bugs
- Bourke and others, 2025, Functional Stream Semantics for a Synchronous Block-Diagram Compiler
- S and others, 2025, Impact of Peer-Based Feedback Mechanisms on Understanding Programming Language
- Minato and others, 2025, Implementation and Evaluation of a System Call Moving Target Defense Applied Multiple Times at Runtime for Binary Injections
- Sack, 2025, Interfacing Programming Language Semantics and Pragmatics What Does “Hello, World” Mean?
- Brant and others, 2025, IoT-CODIFT Compiler Optimization DIFT for IoT and Embedded Devices
- Olteanu and Oprişa, 2025, LambdaGo A Functional Extension of the Go Programming Language
- Li and others, 2025, Lightweight and Holistic-Scalable Serverless Secure Container Runtime for High-Density Deployment and High-Concurrency Startup
- Cummins and others, 2025, LLM Compiler Foundation Language Models for Compiler Optimization

- Chaplygin, 2025, Modeling and implementation of Common LISP functional language compiler
- Qassir, 2025, MyDSL Front-End Compiler Design for a User-Friendly Language Supporting Hybrid Meta-Heuristics
- Recharla, 2025, Parallel Sparse Matrix Algorithms in OCaml v5 Implementation, Performance, and Case Studies
- Midtgaard, 2025, Property-Based Testing of OCaml 5's Runtime System
- 2025, Quantum Software Engineering Algorithm Design, Error Mitigation, and Compiler Optimization for Fault-Tolerant Quantum Computing
- Abu-Yosef and Kong, 2025, Scalable Data-Flow Modeling and Validation of Distributed-Memory Algorithms
- Austen and others, 2025, Sharing Is Scaring Linking Cloud File-Sharing to Programming Language Semantics
- Zelenova, 2025, Static Memory Layout for Real-Time Operating Systems
- Applis and others, 2025, Suspicious Types and Bad Neighborhoods Filtering Spectra with Compiler Information
- Tan and others, 2025, The Burden of Proof Automated Tooling for Rapid Iteration on Large Mechanised Proofs
- Rowland and Perugini, 2025, The Formal Semantics and Implementation of a Domain-Specific Language for Mixed-Initiative Dialogs
- S.Venkatesan, 2025, TOWARDS AUTONOMOUS CODE OPTIMIZATION A REINFORCEMENT LEARNING FRAMEWORK FOR COMPILER DESIGN
- Carlos Paradis and others, 2025, Towards Streamlining Auditing for Compliance With Requirements in Open-Source Software at NASA
- Sholihin and Hidayati, 2024, A Forward Chaining Expert System for Personalized Programming Language Selection
- Narkthong and others, 2024, ALLI/O Diagram An Action-based Visual Programming Language for Embedded System
- Liu and others, 2024, An efficient schedulability analysis based on worst-case interference time for real-time systems
- Santos and others, 2024, Assessing the Impact of Compiler Optimizations on GPUs Reliability
- Sanusi and others, 2024, Assuring Correctness, Testing, and Verification of X-Compiler by Integrating Communicating Stream X-Machine
- Johnson and others, 2024, Automating Pruning in Top-Down Enumeration for Program Synthesis Problems with Monotonic Semantics
- Hück and others, 2024, Compiler-Aided Correctness Checking of CUDA-Aware MPI Applications
- Jeong and others, 2024, Conflict-aware compiler for hierarchical register file on GPUs
- Jiang and others, 2024, DCIM Compiler - Physical Design Generator
- Chen and others, 2024, Design and Implementation of an Aspect-Oriented C Programming Language
- Zhao and others, 2024, Design and Implementation of the MTP Compiler
- Niu and others, 2024, FAIR Flow Type-Aware Pre-Training of Compiler Intermediate Representations
- Talamali and others, 2024, Formal Specification and Verification of MQTT Protocol Using CoQ Proof Assistant
- Nougrahiya and Nandivada, 2024, Homeostasis Design and Implementation of a Self-Stabilizing Compiler
- Lee and Lee, 2024, IMC-PnG Maximizing runtime performance and timing guarantee for imprecise mixed-criticality real-time scheduling
- 2024, Implementation concept of the IoT platform using C++ programming language
- Siambaton and others, 2024, Implementation Draft Programming Oriented Objects in Parking System Application using Language Programming Java

- Zhang and others, 2024, Introducing Compiler Semantics into Large Language Models as Programming Language Translators A Case Study of C to x86 Assembly
- Schlichtkrull and others, 2024, Isabelle-verified correctness of Datalog programs for program analysis
- Chirila and Sora, 2024, Java Single vs. Platform vs. Virtual Threads Runtime Performance Assessment in the Context of Key Class Detection
- Wang and others, 2024, K-RAPID A Formal Executable Semantics of the RAPID Robot Programming Language
- Gruetter and others, 2024, Live Verification in an Interactive Proof Assistant
- Shoushtary and others, 2024, Memento An Adaptive, Compiler-Assisted Register File Cache for GPUs
- 2024, Online platform learning the Java programming language development, implementation and efficiency
- Donaldson and others, 2024, Randomised Testing of the Compiler for a Verification-Aware Programming Language
- Hu and Tang, 2024, Research on compiler version recognition based on random forest algorithm
- Aneesh and others, 2024, Smart Compiler Assistant An AST based Python Code Analysis
- Xu and others, 2024, SWAT4J Generating System Call Allowlist for Java Container Attack Surface Reduction
- Vraný and Shingarov, 2024, Tinyrossa A Compiler Framework for Vertical, Verified Construction of Smalltalk VMs
- Cunha and others, 2024, Trading Runtime for Energy Efficiency Leveraging Power Caps to Save Energy across Programming Languages
- Schoenberger and others, 2024, Using Compiler Frameworks for the Evaluation of Hardware Design Choices in Trapped-Ion Quantum Computers
- Wang, 2024, Worst-Case Blocking Time Optimization in WCRT Analysis for vMPCP on Multi-Core Virtual Machines
- Makki Mohialden and others, 2023, A Comparative Analysis of Python Code-Line Bug-Finding Methods
- Wang and others, 2023, A General-Purpose Compiler Design for Instruction-Based AI Accelerator Implementation
- Roy, 2023, A Theorem Proving Approach to Programming Language Semantics
- Li and others, 2023, A unified proof technique for verifying program correctness with big-step semantics
- 2023, ADAPTIVE PROGRAMMING LANGUAGE LEARNING SYSTEM BASED ON GENERATIVE AI
- Wang and others, 2023, An Automated Verification Framework for HalideIR-Based Compiler Transformations
- Curry, 2023, An HPC-Oriented Runtime Environment for Enabling Computational Storage
- Drechsler and Schnieber, 2023, Automated Polynomial Formal Verification Human-Readable Proof Generation
- Harshithan, 2023, Batwing Compiler An Artificial Intelligence based Compiler
- Audrito and Haures, 2023, Combining Static and Runtime Verification with AC and Coq
- Zhang and others, 2023, Compiler Technologies in Deep Learning Co-Design A Survey
- Baroffio and Reghenzani, 2023, Compiler-Injected SIHFT for Embedded Operating Systems
- 2023, Development of a mobile robot control system in the Python programming language using Raspberry Pi
- Yu and others, 2023, Efficient Generation of Floating-Point Inputs for Compiler-Induced Variability
- Skarman and others, 2023, Enhancing Compiler-Driven HDL Design with Automatic Waveform Analysis

- Ceng Giap and Erviana, 2023, Implementation of Face Mask Detection Using Phyton Programming Language
- Dange and others, 2023, Implementation on A User Authentication Scheme Using Block Chain-Enabled Fog Nodes
- Abi-Karam and others, 2023, INR-Arch A Dataflow Architecture and Compiler for Arbitrary-Order Gradient Computations in Implicit Neural Representation Processing
- Sadasue and Isshiki, 2023, LLVM-C2RTL C/C++ Based System Level RTL Design Framework Using LLVM Compiler Infrastructure
- Herklotz and others, 2023, Mechanised Semantics for Gated Static Single Assignment
- Strauch, 2023, MRPHS A Verilog RTL to C++ Model Compiler Using Intermediate Representations for Object-oriented Model-driven Prototyping
- Mehdi Pourhashem Kallehbasti and Ghafari, 2023, Naturalistic Static Program Analysis
- Alpay and Heuveline, 2023, One Pass to Bind Them The First Single-Pass SYCL Compiler with Unified Code Representation Across Backends
- Vos and others, 2023, Oraqle A Depth-Aware Secure Computation Compiler
- Hirata and others, 2023, Program logic for higher-order probabilistic programs in Isabelle/HOL
- Cheng, 2023, QA4C An Intelligent Question and Answering System for the C Programming Language Based on Knowledge Graph
- Yu, 2023, Reasoning about MLIR Semantics through Effects and Handlers
- Affeldt and others, 2023, Semantics of Probabilistic Programs using s-Finite Kernels in Coq
- Stolyarov, 2023, STATIC ANALYZER IMPLEMENTATION MODEL FOR THE SOLIDTY PROGRAMMING LANGUAGE
- Denis and others, 2023, Tracing task-based runtime systems Feedbacks from the StarPU case
- Tran and others, 2023, Transport Layer Security 1.0 handshake protocol formal verification case study How to use a proof script generator for existing large proof scores
- DelVado Vírseda, 2023, Visualizing Compiler Design Theory from Implementation Through an Interactive Tutoring Tool Experiences and Results
- Vizcaino and others, 2022, Acceleration with long vector architectures Implementation and evaluation of the FFT kernel on NEC SX-Aurora and RISC-V vector extension
- 2022, Analysis on Strengthening Scheme of Office Building Based on Function Change
- Pant and others, 2022, Automatic Software Engineering Position Resume Screening using Natural Language Processing, Word Matching, Character Positioning, and Regex
- Kobusińska and Wilczynski, 2022, Blocked-based Solidity a Service for Graphically Creating the Smart Contracts in Solidity Programming Language
- Schiewe, 2022, Bridging the gap between source code and high-level concepts in static code analysis
- 2022, Cameleer A deductive verification tool for OCaml
- Zhang and others, 2022, Cape compiler-aided program transformation for HTM-based cache side-channel defense
- Ding and others, 2022, CARL Compiler Assigned Reference Leasing
- Oh and others, 2022, CASH-RF A Compiler-Assisted Hierarchical Register File in GPUs
- Ambal and others, 2022, Certified Derivation of Small-Step From Big-Step Skeletal Semantics
- Ozdemir and others, 2022, CirC Compiler infrastructure for proof systems, software verification, and more
- Sharif and others, 2022, COMPAS Compiler-assisted Software-implemented Hardware Fault Tolerance for RISC-V
- Bik and others, 2022, Compiler Support for Sparse Tensor Computations in MLIR
- Huck and others, 2022, Compiler-Aided Type Correctness of Hybrid MPI-OpenMP Applications

- Agathos and others, 2022, Compiler-assisted, adaptive runtime system for the support of OpenMP in embedded multicores
- Lenkefi and Mezei, 2022, Connections between Language Semantics and the Query-based Compiler Architecture
- Yadav and others, 2022, DISTAL the distributed tensor algebra compiler
- Chaliasos and others, 2022, Finding typing compiler bugs
- Shobaki and others, 2022, Graph transformations for register-pressure-aware instruction scheduling
- Alpay and Heuveline, 2022, How much SYCL does a compiler need? Experiences from the implementation of SYCL as a library for nvc++
- Poletanovic and others, 2022, Implementation of Machine Outliner for nanoMIPS in the LLVM Compiler Infrastructure
- Mosaner and others, 2022, Improving Vectorization Heuristics in a Dynamic Compiler with Machine Learning Models
- Hikmatyarsyah and Rahardjo, 2022, Integration of Downlink Scheme VLC Access Techniques for Low-cost Implementation Indoor Communication System A Survey
- Fayzrakhmanov, 2022, Introducing Programming Language Metrics
- Sammler and others, 2022, Islaris verification of machine code against authoritative ISA semantics
- Gordon and others, 2022, Porting the Kitten Lightweight Kernel Operating System to RISC-V
- Valiron, 2022, Semantics of quantum programming languages Classical control, quantum control
- Cho and others, 2022, Sequential reasoning for optimizing compilers under weak memory concurrency
- Postema and others, 2022, Testing a PL/I Compiler Using Precomputation-based Program Generation
- C K, 2022, Video Calling With Build-In Compiler
- De Blaere and others, 2021, A Compiler Extension to Protect Embedded Systems Against Data Flow Errors
- Myreen, 2021, A minimalistic verified bootstrapped compiler proof pearl
- Jeon and others, 2021, A practical algorithm for learning disjunctive abstraction heuristics in static program analysis
- Yvon and Feeley, 2021, A small scheme VM, compiler, and REPL in 4k
- Benito-Montoro and others, 2021, A Tool to Assist the Compiler Construction Instructor in Checking the Equivalence of Specifications Based on Regular Expressions
- Amaliah and others, 2021, Auto Clustering Source Code To Detect Plagiarism Of Student Programming Assignments in Java Programming Language
- Padmasudha Kannan and others, 2021, Automated high-order curved mesh generator with high-level dynamic programming language julia for photonic applications
- Windsor and others, 2021, C4 the C compiler concurrency checker
- Cruttwell and others, 2021, Categorical semantics of a simple differential programming language
- Hossain and others, 2021, Code Generator based on Voice Command for Multiple Programming Language
- Ploensin and others, 2021, Code Transformation Impact on Compiler-based Optimization A Case Study in the CMSSW
- Jung, 2021, CommitBERT Commit Message Generation Using Pre-Trained Programming Language Model
- Steingartner, 2021, Compiler Module of Abstract Machine Code for Formal Semantics Course
- Bruno and others, 2021, Compiler-assisted object inlining with value fields
- Han and others, 2021, Design and Implementation of a Criticality- and Heterogeneity-Aware Runtime System for Task-Parallel Applications

- Liu and others, 2021, Design and Implementation of Multi-core Parallel Compiler Based on OpenMP
- Mpeis and others, 2021, Developer and user-transparent compiler optimization for interactive applications
- Wang, 2021, Helper function inlining in dynamic binary translation
- Pizzolotto and Inoue, 2021, Identifying Compiler and Optimization Level in Binary Code From Multiple Architectures
- Bekiris, 2021, Implementation of UTASTAR Decision Support System in VBA Programming Language
- del Vado Vírseda, 2021, Learning Compiler Design From the Implementation to Theory
- Lööw, 2021, Lutsig a verified Verilog compiler for verified circuit development
- Antoniadis and others, 2021, Open-Source Memory Compiler for Automatic RRAM Generation and Verification
- Nowicki and others, 2021, Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud extended version
- Chernenko and others, 2021, Proving Reflex Program Verification Conditions in Coq Proof Assistant
- Guria and others, 2021, RbSyn type- and effect-guided program synthesis
- Worthington, 2021, Reflections on a decade of MoarVM, a runtime for the Raku programming language keynote
- Zhao and others, 2021, Similarity-Aware Architecture/Compiler Co-Designed Context-Reduction Framework for Modulo-Scheduled CGRA
- 2021, Socket system in php programming language
- Rao and others, 2021, SODA A Semantics-Aware Optimization Framework for Data-Intensive Applications Using Hybrid Program Analysis
- Bourke, 2021, Specification and end-to-end proof of a reactive language and its compiler invited talk
- Yilmazer-Metin, 2021, sRSP An efficient and scalable implementation of remote scope promotion
- Aldweesh and others, 2021, The OpBench Ethereum opcode benchmark framework Design, implementation, validation and experiments
- Tempel and others, 2021, Towards Reliable Spatial Memory Safety for Embedded Software by Combining Checked C with Concolic Testing
- Xiao, 2021, Transformation System of two Similar Syntax Programs Based on the Compiler Principle
- JEONG and others, 2021, Usage Log-Based Testing of Embedded Software and Identification of Dependencies among Environmental Components
- Geng and others, 2021, Verification of Open-Source Memory Compiler Framework with a Practical PDK
- Sorensen and others, 2020, A simulator and compiler framework for agile hardware-software co-design evaluation and exploration
- Pai T and others, 2020, A Systematic Literature Review of Lexical Analyzer Implementation Techniques in Compiler Design
- Diamantopoulos and others, 2020, Agile Autotuning of a Transprecision Tensor Accelerator Overlay for TVM Compiler Stack
- Barinov and others, 2020, Applying compiler-based binary watermarking technology to ensure binary compatibility in GNU/Linux distribution
- Suchy and others, 2020, CARAT a case for virtual memory through compiler- and runtime-based address translation
- Amarasinghe, 2020, Compiler 2.0
- Flatt and Dybvig, 2020, Compiler and runtime support for continuation marks
- Squar and others, 2020, Compiler Assisted Source Transformation of OpenMP Kernels
- Ghosh and others, 2020, Compiler compatible 5.66 Mb/mm2 8T 1R1W register file in 14 nm FinFET technology

- Kim and others, 2020, Compiler-directed soft error resilience for lightweight GPU register file protection
- Chen and others, 2020, CRAC An automatic assistant compiler of checkpoint/restart for OpenCL program
- Zhang and Meng, 2020, Design and Implementation of Multi-core DSP Parallel Compiler Based on Otsu Method
- Watanabe and others, 2020, Design and Preliminary Evaluation of OpenACC Compiler for FPGA with OpenCL and Stream Processing DSL
- Groenewegen and others, 2020, Evolution of the WebDSL runtime reliability engineering of the WebDSL web programming language
- Queiroz Junior and others, 2020, Finding Effective Compiler Optimization Sequences A Hybrid Approach
- Luo, 2020, Heap Memory Snapshot Assisted Program Analysis for Android Permission Specification
- Ranganathan and others, 2020, Hybrid Scalable Action Rule
- Bezrąk and Przyłucki, 2020, Impact of the cloud application programming language on the performance of its implementation in selected serverless environments
- Georgiou and others, 2020, Lost In Translation Exposing Hidden Compiler Optimization Opportunities
- Zaytsev, 2020, Modelling of Language Syntax and Semantics The Case of the Assembler Compiler
- Andrade Guzmán and Hernández Quiroz, 2020, Natural deduction and semantic models of justification logic in the proof assistant Coq
- Amato and others, 2020, On collecting semantics for program analysis
- Phulia and others, 2020, OOElala order-of-evaluation based alias analysis for compiler optimization
- Vasilyev and Mutilin, 2020, Predicate Extension of Symbolic Memory Graphs for the Analysis of Memory Safety Correctness
- 2020, Proceedings of the 2020 International Conference on Embedded Software EMSOFT
- Sanders and others, 2020, Robustness Analysis of Scaled Resource Allocation Models Using the Imperial PEPA Compiler
- Sulema and Glinskii, 2020, Semantics and pragmatics of programming language ASAMPL
- Cambier and others, 2020, TaskTorrent a Lightweight Distributed Task-Based Runtime System in C++
- Natarajan and Broman, 2020, Temporal Property-Based Testing of a Timed C Compiler using Time-Flow Graph Semantics
- Yin and others, 2020, The Implementation of Simple Smart Contract Language and Its Compiler Based on Ethereum Platform
- Huck and others, 2020, Towards compiler-aided correctness checking of adjoint MPI applications
- Holmes and Groce, 2020, Using mutants to help developers distinguish and debug compiler faults
- Dasgupta and others, 2019, A complete formal semantics of x86-64 user-level instruction set architecture
- Benzaken and Contejean, 2019, A Coq mechanised formal semantics for realistic SQL queries formally reconciling SQL and bag relational algebra
- Lima and others, 2019, A memory-bounded, deterministic and terminating semantics for the synchronous programming language Céu
- Feitosa and others, 2019, A monadic semantics for quantum computing in an object oriented language
- ROMPF and AMIN, 2019, A SQL to C compiler in 500 lines of code
- Ye and Delaware, 2019, A verified protocol buffer compiler

- 2019, Automatic Port to OpenACC/OpenMP for Physical Parameterization in Climate and Weather Code Using the CLAW Compiler
- Bassil, 2019, Compiler Design for Legal Document TranslationIn Digital Government
- Elkhouly and others, 2019, Compiler-support for Critical Data Persistence in NVM
- Kondratyev and Promsky, 2019, Correctness of Proof Strategy for the Sisal Program Verification
- Zwanziger, 2019, Dependently-Typed Montague Semantics in the Proof Assistant Agda-flat
- Salánki and Sarvajcz, 2019, Development of a Gait Recognition System in NI LabVIEW Programming Language
- Melnyk and Kozak, 2019, Easy Universal Translator as an Alternative Compiler-Compiler
- Thiselton and Treude, 2019, Enhancing Python Compiler Error Messages via Stack
- Mao and others, 2019, Exploiting Java Stack Forensics for Runtime Monitoring of IoT Services
- Acun and others, 2019, Fine-Grained Energy Efficiency Using Per-Core DVFS with an Adaptive Runtime System
- Facchinetti and others, 2019, Higher-order Demand-driven Program Analysis
- Duhoux and others, 2019, Implementation of a Feature-Based Context-Oriented Programming Language
- Somani and Srivastava, 2019, Implementation of SAAS Intranet compiler in PHP
- Yim and others, 2019, Implementation of Targeted Advertisement Services on ATSC 3.0 Runtime Environment
- Oshita and others, 2019, Improving User Experience of C Programming Language Learning System for Novices
- Besson and others, 2019, Information-Flow Preservation in Compiler Optimisations
- Nguyen and others, 2019, Integrating Static Program Analysis Tools for Verifying Cautions of Microcontroller
- Zhu and others, 2019, Learning to Restrict Test Range for Compiler Test
- Gorodetskiy, 2019, NEXTGEN PROGRAMMING LANGUAGE WITH PROGRAMMABLE SEMANTICS
- Campbell, 2019, Random Compiler for Fast Hamiltonian Simulation
- Wang and others, 2019, Reg An Ultra-Lightweight Container That Maximizes Memory Sharing and Minimizes the Runtime Environment
- Guerrera and others, 2019, Reproducible stencil compiler benchmarks using prova!
- Heo and others, 2019, Resource-Aware Program Analysis Via Online Abstraction Coarsening
- Choi and others, 2019, Reusable inline caching for JavaScript performance
- Ghorbani and Babamir, 2019, Runtime deadlock tracking and prevention of concurrent multithreaded programs A learning-based approach
- Roy and others, 2019, Security Analysis and Efficient Implementation of Code-based Signature Schemes
- Ivanov and others, 2019, Software Structure, Program Generation and Schedulability Analysis of Extracorporeal Perfusion Pump Embedded Controller
- Kim and Ryou, 2019, Source Code Analysis for Static Prediction of Dynamic Memory Usage
- Arceri and Mastroeni, 2019, Static Program Analysis for String Manipulation Languages
- Delgado-Pérez and Segura, 2019, Study of trivial compiler equivalence on C++ object-oriented mutation operators
- Amarasinghe, 2019, The sparse tensor algebra compiler keynote
- Ryu and others, 2019, Toward Analysis and Bug Finding in JavaScript Web Applications in the Wild
- Wang and others, 2019, Tracking runtime concurrent dependences in java threads using thread control profiling

- Tillet and others, 2019, Triton an intermediate language and compiler for tiled neural network computations
- Guan and others, 2019, Wootz a compiler-based framework for fast CNN pruning via composability
- 2018, 2018 Proceedings of the International Conference on Embedded Software EMSOFT
- Leopoldseder and others, 2018, A cost model for a graph-based intermediate-representation in a dynamic compiler
- Fradet and others, 2018, A Generic Coq Proof of Typical Worst-Case Analysis
- Waites and others, 2018, A Genetic Circuit Compiler Generating Combinatorial Genetic Circuits with Web Semantics and Inference
- Santos and others, 2018, A memory-bounded, deterministic and terminating semantics for the synchronous programming language Céu
- Asadollah and others, 2018, A Runtime Verification Tool for Detecting Concurrency Bugs in FreeRTOS Embedded Software
- Bishnu and Bhatia, 2018, Algorithmic Compiler based FPGA Implementation of Iterative Time-Domain Algorithm for Sparse Channel Estimation
- Yi and Lee, 2018, An Educational System Design to Support Learning Transfer from Block-based Programming Language to Text-based Programming Language
- Sah and others, 2018, An Efficient Hardware-Oriented Runtime Approach for Stack-based Software Buffer Overflow Attacks
- Tohid and others, 2018, Asynchronous Execution of Python Code on Task-Based Runtime Systems
- Ginsbach and others, 2018, CAnDL a domain specific language for compiler analysis
- Belyaev and others, 2018, Comparative Analysis of Two Approaches to Static Taint Analysis
- Besson and others, 2018, CompCertS A Memory-Aware Verified C Compiler Using a Pointer as Integer Semantics
- Heim, 2018, Compiler and language design for quantum computing keynote
- Zhang, 2018, Compiler Practice System Integrated with Real Open Source Compiler
- Huck and others, 2018, Compiler-aided Type Tracking for Correctness Checking of MPI Applications
- Sharrad and others, 2018, Delta Debugging Type Errors with a Blackbox Compiler
- Medeiros and others, 2018, Evaluation of Compiler Optimization Flags Effects on Soft Error Resiliency
- Beaumont and others, 2018, Fast approximation algorithms for task-based runtime systems
- Kusmenko and others, 2018, Highly-Optimizing and Multi-Target Compiler for Embedded System Models
- Terao, 2018, Lazy Abstraction for Higher-Order Program Verification
- Niephaus and others, 2018, Live Multi-language Development and Runtime Environments
- Lyamin, 2018, Method of Formal Program Verification for Post Machine Virtual Laboratory
- Oehlert and others, 2018, Mitigating Data Cache Aging through Compiler-Driven Memory Allocation
- Salama and others, 2018, Online programming language-Learning management system
- Liu and others, 2018, Research of Register Pressure Aware Loop Unrolling Optimizations for Compiler
- Deng and Namjoshi, 2018, Securing a compiler transformation
- Ismael and others, 2018, System on Chip Implementation of Compiler Stack with a Delimiter Matching Application
- Bourke and others, 2018, Towards a verified Lustre compiler with modular reset
- Bowman and Ahmed, 2018, Typed closure conversion for the calculus of constructions

- Fumero and Kotselidis, 2018, Using compiler snippets to exploit parallelism on heterogeneous hardware a Java reduction case study
- Tatsuoka and Kaneko, 2018, Wire congestion aware high level synthesis flow with source code compiler
- Takase and others, 2018, Work-in-Progress Design Concept of a Lightweight Runtime Environment for Robot Software Components Onto Embedded Devices
- Choi and others, 2018, Work-in-Progress Lightweight Deadlock Detection Technique for Embedded Systems via OS-Level Analysis
- Janetschek and Prodan, 2017, A compiler transformation-based approach to scientific workflow enactment
- Imamoglu and Cetinkaya, 2017, A rule based decision support system for programming language selection
- Cambou, 2017, A XOR data compiler Combined with physical unclonable function for true random number generation
- Shen, 2017, Android Security via Static Program Analysis
- 2017, C# Compiler for Blind
- Akdur and others, 2017, Characterizing the Development and Usage of Diagrams in Embedded Software Systems
- Lambert and Saunders, 2017, Compiler auto-vectorization of matrix multiplication modulo small primes
- Proy and others, 2017, Compiler-Assisted Loop Hardening Against Fault Attacks
- Yaneva and others, 2017, Compiler-assisted test acceleration on GPUs for embedded software
- Luo and others, 2017, Compiler-Assisted Threshold Implementation against Power Analysis Attacks
- Maurer and others, 2017, Compiling without continuations
- Akdur and others, 2017, Cross-factor analysis of software modeling practices versus practitioner demographics in the embedded software industry
- Kirichenko and Tarasov, 2017, Development of design flow for multiported register files, which includes a cell library and a compiler for SOI 0.25-µm process
- Prenzel and Provost, 2017, Dynamic Software Updating of IEC 61499 Implementation Using Erlang Runtime System
- Ruberg and others, 2017, Embedded software performance estimations at different compiler optimisation levels
- SulĂr and Poruban, 2017, Exposing Runtime Information through Source Code Annotations
- Ruchkin and others, 2017, Frame model of a compiler of cluster parallelism for embedded computing systems
- Pieters and others, 2017, Handlers for Non-Monadic Computations
- Tian and others, 2017, LLVM Compiler Implementation for Explicit Parallelization and SIMD Vectorization
- Endo and others, 2017, On the Interactions Between Value Prediction and Compiler Optimizations in the Context of EOLE
- Ghica and Alyahya, 2017, On the Learnability of Programming Language Semantics
- Meghzili and others, 2017, On the Verification of UML State Machine Diagrams to Colored Petri Nets Transformation Using Isabelle/HOL
- Wimmer and others, 2017, One compiler deoptimization to optimized code
- Maccabe, 2017, Operating and Runtime Systems Challenges for HPC Systems
- Würthinger and others, 2017, Practical partial evaluation for high-performance dynamic language runtimes
- Chong and others, 2017, Programming languages and compiler design for realistic quantum hardware
- Li and others, 2017, Promotion of Educational Effectiveness by Translation-based Programming Language Learning Using Java and Swift

- Baek and Kim, 2017, Prototype implementation of the OpenGL ES 2.0 shading language offline compiler
- Khatami and others, 2017, Redesigning OP2 Compiler to Use HPX Runtime Asynchronous Techniques
- Lins and others, 2017, Register File Criticality and Compiler Optimization Effects on Embedded Microprocessors Reliability
- Kiefer and others, 2017, Relational Program Reasoning Using Compiler IR
- Wang, 2017, Research on the Execution Time Analysis Technology of the Worst Case System in Real Time System
- Santhiar and Kanade, 2017, Static deadlock detection for asynchronous C# programs
- Wu and others, 2017, Two Schemes to Improve the Implementation of the Aggregation Based Algebraic Multigrid Preconditioner
- Zubarev, 2017, TYPE ANALYSIS FOR THE PREDICATE PROGRAMMING LANGUAGE
- Márton and Porkoláb, 2017, Unit Testing in C++ with Compiler Instrumentation and Friends
- Engelke and Weidendorfer, 2017, Using LLVM for Optimized Lightweight Binary Re-Writing at Runtime
- De Luca and Chen, 2017, Visual IoT/Robotics Programming Language in Pi-Calculus
- Dave and others, 2016, A 1V 800MHz 140Kb register file compiler using variation aware self-timing in 40nm bulk CMOS
- Zhou and Xue, 2016, A Compiler Approach for Exploiting Partial SIMD Parallelism
- Bispo and Cardoso, 2016, A MATLAB subset to C compiler targeting embedded systems
- Chen and others, 2016, A Process-Visible Compiler Aimed for Teaching Assistant
- Wenger and others, 2016, A Programming Language and System for Heterogeneous Cloud of Things
- Kalyur and Nagaraja, 2016, A survey of modeling techniques used in compiler design and implementation
- Hmid and others, 2016, A Transfer-Aware Runtime System for Heterogeneous Asynchronous Parallel Execution
- Horký and others, 2016, Analysis of Overhead in Dynamic Java Performance Monitoring
- Ivey and Riley, 2016, Analysis of Programming Language Overhead in DCE
- Chen and Zong, 2016, Android App Energy Efficiency The Impact of Language, Runtime, Compiler, and Implementation
- Klöckner and others, 2016, Array program transformation with Loo.py by example high-order finite elements
- Jacek and others, 2016, Assessing the limits of program-specific garbage collection performance
- Schäfer and others, 2016, Axiomatic semantics for compiler verification
- Xu and others, 2016, CAOPLE A Programming Language for Microservices SaaS
- Huang and Huang, 2016, Cell-based delay locked loop compiler
- Mäkelä and others, 2016, Compiler assisted dynamic allocation of finite hardware acceleration resources for parallel tasks
- Zhang and others, 2016, Compiler Transformation to Generate Hybrid Sparse Computations
- Tolpin and others, 2016, Design and Implementation of Probabilistic Programming Language Anglican
- Patel and Lee, 2016, Dynamic Analysis of Multi-threaded Embedded Software to Expose Atomicity Violations
- Hariri and others, 2016, Evaluating the Effects of Compiler Optimizations on Mutation Testing at the Compiler IR Level
- Sun and others, 2016, Finding compiler bugs via live code mutation
- Gotti and Mbarki, 2016, Java Swing Modernization Approach - Complete Abstract Representation based on Static and Dynamic Analysis

- Na and others, 2016, JavaScript Parallelizing Compiler for Exploiting Parallelism from Data-Parallel HTML5 Applications
- Truong and others, 2016, Latte a language, compiler, and runtime for elegant and efficient deep neural networks
- Yamamoto and others, 2016, Lightweight Ruby Framework for Improving Embedded Software Efficiency
- Guo and Si, 2016, Mechanical hydraulic characteristic analysis scheme based on lightweight crowd data in mobile embedded devices
- Tian and others, 2016, Optimizing GPU Register Usage Extensions to OpenACC and Compiler Optimizations
- Raj, 2016, Performance analysis of different specifications of copy propagation transformation using machine SUIF compiler infrastructure
- Rodríguez and others, 2016, Proving Correctness of a Compiler Using Step-indexed Logical Relations
- Gómez-Déniz and others, 2016, Random Tests Combining Mathematica Package and Latex Compiler
- Gaudet and Stoodley, 2016, Rebuilding an airliner in flight a retrospective on refactoring IBM testarossa production compiler for Eclipse OMR
- Lin and others, 2016, Rust as a language for high performance GC implementation
- Xue and Bogdan, 2016, Scalable and realistic benchmark synthesis for efficient NoC performance evaluation
- Zhang, 2016, Selection and Improvement of Computer Programming Language
- Cho, 2016, Semantics for a Quantum Programming Language by Operator Algebras
- Downen and others, 2016, Sequent calculus as a compiler intermediate language
- Ruchkin and others, 2016, Smart compiler embedded computing systems based on cluster parallelism
- Kroening and others, 2016, Sound static deadlock analysis for C/Pthreads
- Amin and others, 2016, System Verilog Assertions Synthesis Based Compiler
- Campos-López and Wannenwetsch, 2016, The PERICLES Process Compiler Linking BPMN Processes into Complex Workflows for Model-Driven Preservation in Evolving Ecosystems
- Sun and others, 2016, Toward understanding compiler bugs in GCC and LLVM
- Khaldi and Chapman, 2016, Towards Automatic HBM Allocation Using LLVM A Case Study with Knights Landing
- MORIGUCHI and others, 2016, Verification of Content-Centric Networking Using Proof Assistant
- Buchwald and others, 2016, Verified construction of static single assignment form
- Zhou and others, 2016, Worst case response time and schedulability analysis for real-time software transactional memory-lazy conflict detection STM-LCD
- Mhaske and others, 2015, A 2.48Gb/s FPGA-based QC-LDPC decoder An algorithmic compiler implementation
- Kyrtatas and others, 2015, A Basic Linear Algebra Compiler for Embedded Processors
- Murthy and Mellor-Crummey, 2015, A Compiler Transformation to Overlap Communication with Dependent Computation
- Michels and others, 2015, A new probabilistic constraint logic programming language based on a generalised distribution semantics
- Yamato, 2015, Automatic verification for plural virtual machines patches
- Verma and Bakshi, 2015, Chronological Advancement in Compiler Design A Review
- Royuela and others, 2015, Compiler analysis for OpenMP tasks correctness
- Li and others, 2015, Compiler directed automatic stack trimming for efficient non-volatile processors
- Haj-Yihia and others, 2015, Compiler-Directed Power Management for Superscalars
- Basu and others, 2015, Compiler-Directed Transformation for Higher-Order Stencils

- Mosterman and Zander, 2015, Cyber-physical systems challenges a needs analysis for collaborating embedded software systems
- Abramsky and Horsman, 2015, DEMONIC programming a computational language for single-particle equilibrium thermodynamics, and its formal semantics
- Gordon and Scholz, 2015, Dynamic adaptation of functional runtime systems through external control
- 2015, Efficient Implementation of Class Based Decomposition Schemes for Naive Bayes Classifier
- Kim, 2015, Exploiting Window Query Semantics in Scalable Data Stream Processing
- Stanisic and others, 2015, Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures
- Le and others, 2015, Finding deep compiler bugs via guided stochastic program mutation
- Queiroz Junior and da Silva, 2015, Finding Good Compiler Optimization Sets - A Case-based Reasoning Approach
- Breitner, 2015, Formally proving a compiler transformation safe
- Cazzola and Olivares, 2015, Gradually Learning Programming Supported by a Growable Programming Language
- Jiang and others, 2015, Implementation and Comparison of the Way That Office Document Is Converted to PDF Documents in the Java Runtime Environment
- 2015, Implementation of a Motor Imagery based BCI System using Python Programming Language
- Mehta and Yew, 2015, Improving compiler scalability optimizing large programs at small price
- Leopoldseder and others, 2015, Java-to-JavaScript translation via structured control flow reconstruction of compiler IR
- Park and others, 2015, KJS a complete formal semantics of JavaScript
- Ko and others, 2015, LaminarIR compile-time queues for structured streams
- Iskra and Hoefler, 2015, Operating systems and runtime environments on supercomputers
- Ghica and Tapus, 2015, Optimized retargetable compiler for embedded processors - GCC vs LLVM
- Yang and Ruiz Varela, 2015, Qualifying non-volatile register files for embedded systems through compiler-directed write minimization and balancing
- Mosses, 2015, Semantics of programming languages Using Asf+Sdf
- Naish, 2015, Sharing analysis in the Pawns compiler
- Srinivasan and Reps, 2015, Synthesis of machine code from semantics
- D'Silva and others, 2015, The Correctness-Security Gap in Compiler Optimization
- Ricketts and others, 2015, Towards verification of hybrid systems in a foundational proof assistant
- Papadakis and others, 2015, Trivial Compiler Equivalence A Large Scale Empirical Study of a Simple, Fast and Effective Equivalent Mutant Detection Technique
- Lezuo and others, 2015, vanHelsing A Fast Proof Checker for Debuggable Compiler Verification
- Blazy and others, 2015, Verified Validation of Program Slicing

What the survey shows

The representation question is settled in the literature and the selection question is not. Four independent traditions converge on a discriminated return for terminating coroutines, and none of them addresses whether a bounded-memory artefact should pay that discrimination on every call of a construct that never terminates. The present article contributes the observation that the two source forms have different trace alphabets and that the choice can therefore be made per form rather than globally, which the surveyed work does not consider because no surveyed language separates the two forms in its type system.

That separation is Keleusma's own contribution to the problem, and it is the reason the question is even askable here. It is also why the survey cannot answer it.

The scale of the survey should be read for what it is. It lists 1,980 references, of which 35 were selected because the argument depends on them and the rest were harvested by query across thirteen clusters. **The harvested majority establishes coverage and not agreement.** A reader looking for the works that carry the argument should read the 35, which are named in the prose above. A reader checking whether the survey was assembled to flatter its conclusion should note that the queries were fixed before any record was seen, that every admitted record is listed including the 680 that are merely adjacent, and that the selection procedure is reported in Method with the count it discarded.

The Source Base

The generator is a detached package inside the Keleusma repository, consuming the reference compiler's in-memory module representation and emitting LLVM intermediate representation through the inkwell bindings. The bytecode it consumes has already passed a structural verifier, in the tradition the Java Virtual Machine Specification sets out for bytecode verification, so the backend declines what it cannot lower and does not approximate it. The classification instrument is a test in the same package, reporting rather than asserting, with one guard that fails if the corpus walk finds nothing, because a broken path and a real zero look identical in a report.

The differential oracle drives both the virtual machine and the just-in-time compiled native code over identical bytecode and compares the whole sequence of yielded values. The stream drivers are bounded by a caller-supplied count, because a productively divergent program will otherwise run until something kills it, and a hung test reports nothing at all, worse than a failing one.

Epistemic State

Measured, and reproducible from the instrument. The corpus yields 24 stream chunks, of which 22 have a single suspension at the end of the body, 0 have multiple top-level suspensions, 1 has suspensions nested in conditionals, and 1 delegates with no suspension of its own. The nested chunk has 19 suspensions, all inside conditionals and none inside a loop, at nesting depths from 2 to 11. The corpus contains exactly 1 terminating coroutine chunk, in which 9 of 9 suspensions are immediately followed by a return. Ten stream chunks are refused by the current tail-position rule for a tail sequence that is effect-free and stack-neutral.

Derived, and checkable from the definitions. That a terminating coroutine produces two distinguishable observable events while a return-based convention provides one channel, and that the second therefore cannot carry the first, by $|W \times W| = |W|^2 > |W|$ for $|W| \geq 2$. That a divergent coroutine produces one, because its trace alphabet omits the completion letter. That the admissibility rule satisfies $R \subsetneq P$, so its defect is coverage and not soundness. That $\Delta(\pi) = -1$ for both candidate tail segments. That a suspension inside a conditional is a control-flow join while one inside a loop crosses a back edge.

Assumed, and marked as such. That the return convention is cheaper at run time. The structural grounds are stated and the frame-size measurement that would establish it was not performed, because the machine-code section carrying it is not emitted on this host.

Verified, and the verification method itself produced a finding twice. All 35 hand-selected research identifiers were resolved against the registry and compared with the work they are cited as. **Thirty-five of thirty-five resolve to that work, an error rate of zero.** The 1,945 harvested identifiers were **not** audited that way and do not need to be, because they were transcribed from the registry that issued them rather than recalled, so the failure

mode the audit exists to catch cannot arise. What the audit would still catch, and did not run for, is a harvested record cited for a claim it does not support, and no harvested record is cited for any claim.

Sampled, because checking all of them would be checking the registry against itself. A random sample of 250 harvested identifiers was resolved, and **250 of 250 exist**. What the sample does establish is a transcription check on the pipeline in this repository, since a fault there would affect a constant fraction of records and not a rare one, and no such fraction appeared.

Twenty-two of the 250, or 8.8 percent, resolved only through the registry. The identifier resolver did not serve them. Twenty refused the connection outright, of which 14 were Defense Technical Information Center deposits, and 2 returned a not-found response from a publisher that no longer serves the landing page. **In every one of the 22 the identifier is registered and the record is correct.** The defect is therefore in the resolution path and not in the citation. It is reported because a reader checking a sample of these references by clicking them will meet roughly one failure in eleven, and that failure is not evidence of a bad citation.

Two passes were run with two different instruments and they flagged different records, which is itself the lesson. A title-overlap heuristic flagged four. A later check comparing author surname and year flagged five. **The union is eight distinct records and every one of them was vindicated on individual inspection**, for four separate reasons.

Artefact	Records	What the registry does
Title split from subtitle	3	Stores “CakeML”, “Coroutines” or “Capriccio” while the citation carries the full descriptive title
Wrong registration agency	1	Proceedings in the LIPIcs series deposit with DataCite, so a Crossref lookup returns nothing for a valid identifier
Surname particle dropped	1	Stores de Moura as “Moura”, so an anchor built from the full surname fails a substring test
Registry typo	1	Stores “Dyvbig” for Dybvig, so the article is right and the registry is wrong
Identifier resolves to a reprint	2	Reynolds 1972 and Strachey and Wadsworth 1974 carry identifiers for their 1998 and 2000 reprints in Higher-Order and Symbolic Computation, so the registry year is later than the year cited

The last category is the one worth carrying forward, because it is not an instrument error at all. The identifier is correct, the work is correct, and the year genuinely differs, since a foundational paper and its journal reprint are two publications of one text. **A checker comparing years will flag every reprinted classic in any bibliography**, and a bibliography of foundational work is mostly reprinted classics.

Not one of the eight was a citation defect, and reporting any of them as one would have been a measurement mistake of exactly the kind this series keeps documenting. It is recorded because the previous article reported a 5.5 percent rate by the same method and did not separate these artefacts from genuine mismatches, which means **that figure is an upper bound rather than an estimate** and the true rate there may be lower.

Corrected during writing, and left visible. The plan this article set out to support was to unify the two conventions, and it survived until it was traced against a three-instruction example. **The central claim was also stated in a form stronger than the evidence supported**, assuming a one-word return until the governing application binary interface documents were read, and the correction opened an option the draft did not contain. An earlier design in this series specified a reordering transformation for bodies with several suspension points, of which the corpus contains none. A classification that called nineteen suspensions "the general case" was collapsing two structurally different situations, and the per-suspension measurement separated them.

What this article does not establish. Which of the remaining options is correct, since that turns on an ABI decision belonging to a workstream this article does not speak for. Whether the widened-return option is cheaper than two conventions, which is an empirical question and is not measured here. Whether the source-restructuring option is reasonable. Whether the corpus resembles the population the generator will serve.

The literature survey has two layers with different standing and only one of them bears on the argument.

The 35 hand-selected works were chosen by the author's judgement of relevance, were read, and are the ones the prose reasons from. **That layer is a narrative review.** No protocol was registered and no completeness is claimed for it.

The 1,945 harvested works were retrieved by fixed queries against two registries and admitted by a stated filter, which is reproducible in a way the narrative layer is not. **It is still not a systematic review.** The query set was written by the same author whose judgement the narrative layer rests on, only two registries were searched, no grey literature was sought, no inclusion criteria beyond the title filter were applied, and no harvested record was read.

A reader should therefore treat the assertion that four traditions converge on a discriminated return as **an observation over the works cited rather than over the field**. The specific negative claim, that none of the surveyed work addresses per-form selection in a bounded-memory setting, is the one most exposed to an omission. **The harvest makes that claim more exposed rather than less**, because the harvested titles were not read and a paper answering the question could sit in the list unremarked. It is stated as a gap in the survey rather than a gap in the literature.

The strongest claim the evidence supports is that the return convention cannot serve terminating coroutines, on an argument that does not depend on the corpus at all. **The weakest link is the cost comparison** between the two remaining options, which rests on an unmeasured performance claim and on a corpus of twenty-four.

Out of Scope

The design of the lowering for any individual instruction. Register allocation, scheduling and peephole optimisation. The worst-case execution time analysis, which is a separate subject with a separate method. The exact frame-size comparison between the two conventions, which requires a target this host does not emit. Whether the delegating program should be restructured, which is a language question. The Keleusma language itself, which is covered in the getting-started article and the self-hosting strategy.

Conclusion

An apparent design wart turned out to be a semantic boundary, and the evidence that made it look like a wart was accurate. The measurement said nine out of nine, and nine out of nine was true. It was a fact about the shape of instances, offered in answer to a question about the arity of an interface, and the two are not the same question.

The check that settled it cost one traced example and produced an argument that holds independently of the corpus, which the measurement it overturned does not. **Where a claim can be settled by counting the channels an interface provides against the outcomes a behaviour produces, that count dominates any distribution over instances**, and it is available before any code is written.

The article also reports a rule shipped one increment earlier that is stricter than its own stated purpose, refusing ten of twenty-four cases whose tail sequence is provably harmless. It was found by building a table, not by running a test, which is now the second such finding in this series and no longer looks like a coincidence.

References

The references below come from two sources and the distinction matters.

35 research references, together with every specification, application binary interface document and piece of reference documentation, were **selected by hand** because the argument depends on them. Each states something a step of the reasoning uses, and none is included for completeness.

The remaining 1,945 research references were **harvested from bibliographic registries by query** and constitute the survey in The Contemporary Literature. They were not selected for agreement with the article’s conclusion and the selection that produced them is reported in Method.

The two sources have different failure modes and the previous article was caught by one of them. The previous article in this series reports a 5.5 percent error rate on its identifiers, and the cause was that the identifiers were supplied from memory, which is a generative process that produces plausible identifiers for works that do not carry them. **Harvesting removes that failure mode rather than reducing it**, because a harvested identifier is transcribed from the registry that issued it and was never a guess. What harvesting does not remove is the risk that a correctly transcribed record is cited for a claim it does not support, and that risk falls on the 35 hand-selected entries, which is why those and only those are also checked against the work they are cited as. The result of that check is reported in the Epistemic State and not assumed.

Reference

- Arm Architecture Procedure Call Standard for the Arm 64-bit Architecture
- C++ Coroutines
- ECMA-262, ECMAScript Language Specification
- ECMA-334, C# Language Specification
- inkwell, safe Rust bindings to LLVM
- Oracle Java Virtual Machine Specification
- Kotlin Coroutines
- LLVM Coroutines
- LLVM Language Reference
- LLVM Stack Maps and Patch Points
- Lua 5.4 Reference Manual
- PEP 255, Simple Generators

- PEP 342, Coroutines via Enhanced Generators
- PEP 380, Syntax for Delegating to a Subgenerator
- PEP 492, Coroutines with async and await Syntax
- POSIX `makecontext` and `swapcontext`
- The Rust Reference
- System V Application Binary Interface, AMD64 Architecture Processor Supplement
- WebAssembly Stack Switching Proposal

Related Post

- Related Post, Compiler Backend Bring-Up: Blocking Frequency as the Ordering Principle
- Related Post, Getting Started with Keleusma 0.2.2
- Related Post, Keleusma’s Self-Hosting Strategy

Research

- 2018, 2018 Proceedings of the International Conference on Embedded Software EMSOFT
- 2002, 3D Pre-Stack Depth Migration V0 Analysis and Monte Carlo Automatic Velocity Picking in Depth
- 2012, A Compositional Scheme and Framework for Safety Critical Systems Verification
- 2020, A Fibrational Method of Indexed Coinductive Data Types
- A and others, 2024, Design of Mechatronics Based Virtual Telepresence for Robotic System Using Raspberry Python Interpreter
- 2004, A structural approach to operational semantics
- Abdallah, 2018, PRISM revisited Declarative implementation of a probabilistic programming language using multi-prompt delimited control
- Abdelmaksoud and others, 2023, DEL Dynamic Symbolic Execution-based Lifter for Enhanced Low-Level Intermediate Representation
- Abdulaziz and Koller, 2022, Formal Semantics and Formally Verified Validation for Temporal Planning
- Abdulsalam and others, 2014, Program energy efficiency The impact of language, compiler and implementation choices
- Abe, 2019, A type system for data independence of loop iterations in a directive-based PGAS language
- Abella and others, 2017, Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation
- Abi-Karam and others, 2023, INR-Arch A Dataflow Architecture and Compiler for Arbitrary-Order Gradient Computations in Implicit Neural Representation Processing
- Abrahams, 1979, The CIMS PL/I compiler
- Abramsky and Horsman, 2015, DEMONIC programming a computational language for single-particle equilibrium thermodynamics, and its formal semantics
- Abu-Yosef and Kong, 2025, Scalable Data-Flow Modeling and Validation of Distributed-Memory Algorithms
- Aceto and Fokkink, 2004, Guesteditors’introduction Specialissueon Structural Operational Semantics
- Achour and Benattou, 2018, Constraint Based Testing and Verification of Java Bytecode Programs
- Achour and others, 2018, A Constraint-Based Verification Approach for Java Bytecode Programs
- Acun and others, 2019, Fine-Grained Energy Efficiency Using Per-Core DVFS with an Adaptive Runtime System
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Trade Name Compiler Validation Summary Report. Certificate Number 880620W1.09092, Encore Computer Cor-

poration, Encore Verdix Ada Development System, Version 5.5, Encore Multimax 320
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Tradename Compiler Validation Summary Report Certificate Number 880201W1.09019 Verdix Corporation VAda-010-2323, Version 5.5 Sequent Balance 8000
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Tradename Compiler Validation Summary Report Certificate Number 880429W1.09053 Telesoft, Inc. TeleGen2 Ada Compiler for VAX/VMS to 1750A, Version 3.22 MicroVAX 2 to MIL-STD-1750A ECSPO RAID Simulator
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1988, Ada Tradename Compiler Validation Summary Report Certificate Number 880815W1.09143 Rational VAX-VMS, Version 2.0.45 Rational R1000 Series 200 Model 20 and VAX-11/750 Host and Target
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Procedures Version 2.0
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report. CONVEX Computer Corporation, CONVEX Ada, Version 1.1 C210, Host and Target 890508W1.10077
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report. Meridian Software Systems, Inc., AdaVantage, Version 3.0, IBM PS/2 Model 80 with Floating Point Co-Processor Host and Target 890405W1.10049
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report. Verdix Corporation, VAda-110-2323, Version 5.5, Sequent Balance 8000 Host and Target , 890216W1.10029
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1989, Ada Compiler Validation Summary Report Certificate Number 890329I1. 10076 SYSTEAM KG, SYSTEAM Ada Compiler VAX/VMS x MC68020/05-9 Version 1.81
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1990, Ada Compiler Validation Procedures. Version 2.1
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1991, Ada Compiler Validation Summary Report Certificate Number 910612W1. 11168 Telesoft, IBM Ada/370, Version 1.2.0 without Optimization IBM 3080, V / SP HPO Rel 5.0 Unopt Host and Target
- ADA JOINT PROGRAM OFFICE ARLINGTON VA, 1992, Ada Compiler Validation Procedures, Version 3.1
- 2023, ADAPTIVE PROGRAMMING LANGUAGE LEARNING SYSTEM BASED ON GENERATIVE AI
- Affeldt and others, 2023, Semantics of Probabilistic Programs using s-Finite Kernels in Coq
- Agathos and others, 2022, Compiler-assisted, adaptive runtime system for the support of OpenMP in embedded multicores
- Agathos and others, 2011, Design and Implementation of OpenMP Tasks in the OMPi Compiler
- Телегин, 2023, AHEAD-OF-TIME and JUST-IN-TIME technologies
- Ahman and Staton, 2013, Normalization by Evaluation and Algebraic Effects
- Aigrain and others, 1984, Experience with a Graham-Glanville style code generator
- Aissa and others, 2007, Bringing Worst Case Execution Time Awareness to an Open Smart Card OS
- Aisyiyah and Eviyanti, 2026, Android-based Programming Language to Natural Language Translator App
- Akanbi and others, 2022, Code Generation Techniques in Compiler Design Conceptual and Structural Review
- Akdur and others, 2017, Characterizing the Development and Usage of Diagrams in Embedded Software Systems
- Akdur and others, 2017, Cross-factor analysis of software modeling practices versus practitioner demographics in the embedded software industry
- Alatawi and others, 2016, Generating source inputs for metamorphic testing using dy-

namic symbolic execution
- Al-Bataineh and others, 2015, Accelerating worst case execution time analysis of timed automata models with cyclic behaviour
- Albert and others, 2024, Synthesis of Sound and Precise Storage Cost Bounds via Unsound Resource Analysis and Max-SMT
- Aldweesh and others, 2021, The OpBench Ethereum opcode benchmark framework Design, implementation, validation and experiments
- Alexander and Black, 2016, The performance of object encodings in JavaScript
- Ali and others, 2025, Does Coding Style Really Survive Compilation? Stylometry of Executable Code Revisited
- Ali and others, 2025, CrossGuard Runtime-Adaptive LLM Fuzzing for Cross-Contract Vulnerabilities Detection
- Aljaafari and others, 2022, Combining BMC and Fuzzing Techniques for Finding Software Vulnerabilities in Concurrent Programs
- Alkhalid and Labiche, 2018, Towards GUI Functional Verification using Abstract Interpretation
- Alladi and others, 2026, Enabling Automatic Compiler-Driven Vectorization of Transformers
- Allamanis and Sutton, 2013, Why, when, and what Analyzing Stack Overflow questions by topic, type, and code
- Allen and others, 2000, Subsalt Imaging using 3D Pre-Stack Depth Migration in the UK Southern North Sea - a Case History
- Alpay and Heuveline, 2022, How much SYCL does a compiler need? Experiences from the implementation of SYCL as a library for nvc++
- Alpay and Heuveline, 2023, One Pass to Bind Them The First Single-Pass SYCL Compiler with Unified Code Representation Across Backends
- Altan, 2026, Error-Resilient Quantum Compiler Design for Efficient Qubit Mapping, Gate Optimization, and Noise Mitigation in NISQ-Era Devices
- Altassan and Ahmad, 2024, Green threads of change Unravelling the gendered and experienced moderators in the sustainable symphony of green HR practices and environmental responsibility
- Amaliah and others, 2021, Auto Clustering Source Code To Detect Plagiarism Of Student Programming Assignments in Java Programming Language
- Amarasinghe, 2004, Session details Compiler and simulator construction
- Amarasinghe, 2019, The sparse tensor algebra compiler keynote
- Amarasinghe, 2020, Compiler 2.0
- Amato and others, 2020, On collecting semantics for program analysis
- Ambal and others, 2022, Certified Derivation of Small-Step From Big-Step Skeletal Semantics
- Amin and others, 2016, System Verilog Assertions Synthesis Based Compiler
- 2022, Analysis on Strengthening Scheme of Office Building Based on Function Change
- 2016, Analytics of Application Resource Utilization within the Virtual Machine
- Anderson and Loginov, 2013, Static analysis of machine code for supply-chain risk management
- Anderton, 2022, Investigating Sign Language Interpreter Rendering and Guiding Methods in Virtual Reality 360-Degree Content
- Andrade Guzmán and Hernández Quiroz, 2020, Natural deduction and semantic models of justification logic in the proof assistant Coq
- Andrews and others, 1988, Design and implementation of the UW Illustrated compiler
- Aneesh and others, 2024, Smart Compiler Assistant An AST based Python Code Analysis
- Anier, 2008, Motion recognition with abstract interpretation and HMM
- Antoniadis and others, 2021, Open-Source Memory Compiler for Automatic RRAM Generation and Verification
- Aparanji and others, 2017, INDUCTION OF PROPOFOL WITH COINDUCTION OF

PROPOFOL, MIDAZOLAM VERSUS PROPOFOL AUTO COINDUCTION- A COMPARATIVE STUDY
- Compiling with Continuations
- Appel and Jim, 1989, Continuation-passing, closure-passing style
- Appel and Shao, 1992, Callee-save registers in continuation-passing style
- Applis and others, 2025, Suspicious Types and Bad Neighborhoods Filtering Spectra with Compiler Information
- Aquino and others, 2018, Worst-Case Execution Time Testing via Evolutionary Symbolic Execution
- Arakaki and Hirokawa, 2026, Foundational Design of Multi-Modal Typed Programming Language for Quantum-Classical Hybrid Computing System
- Arcaro and others, 2020, Reliability Test based on a Binomial Experiment for Probabilistic Worst-Case Execution Times
- Arceri and Mastroeni, 2019, Static Program Analysis for String Manipulation Languages
- Arendsee, 2026, morloc a workflow language for multi-lingual programming under a common type system
- Aristizábal and others, 2017, Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation
- Arnström and others, 2024, Exact Worst-Case Execution-Time Analysis for Implicit Model Predictive Control
- Arriaga and others, 2026, Tempo An ML-KEM to PAKE Compiler Resilient to Timing Attacks
- 2003, ART An Implementation on the Active_object RunTime Systems Applicable for the Embedded Systems
- Asadollah and others, 2018, A Runtime Verification Tool for Detecting Concurrency Bugs in FreeRTOS Embedded Software
- Asai, 2002, Online partial evaluation for shift and reset
- Asai, 2004, Offline partial evaluation for shift and reset
- Asai and Fujii, 2025, Defining Algebraic Effects and Handlers via Trails and Metacontinuations
- Askim and others, 2004, 4D Seismic Analysis Using Pre Stack Depth Migration
- Astarte, 2025, Conceptualising Programming Language Semantics
- Ataei and Manohar, 2019, AMC An Asynchronous Memory Compiler
- Atapattu, 1994, Design of a Parallel Object Oriented Programming Language
- Attrapadung and others, 2018, Efficient Two-level Homomorphic Encryption in Prime-order Bilinear Groups and A Fast Implementation in WebAssembly
- Attrot and others, 2026, A Pattern Generation Language for MLIR Compiler Matching and Rewriting
- Auchterlonie and others, 2007, Velocity Model Building for Pre-Stack Depth Migration - An Onshore Libya Case Study
- Audrito and Haures, 2023, Combining Static and Runtime Verification with AC and Coq
- Auguston and others, 2001, Visual Meta-Programming Language
- Aung and others, 2011, Compiler-assisted technique for rapid performance estimation of FPGA-based processors
- Auslander and Hopkins, 1982, An overview of the PL.8 compiler
- Austen and others, 2025, Sharing Is Scaring Linking Cloud File-Sharing to Programming Language Semantics
- 2019, Automatic Port to OpenACC/OpenMP for Physical Parameterization in Climate and Weather Code Using the CLAW Compiler
- Avanzini and others, 2021, On continuation-passing transformations and expected cost analysis
- Avigad and others, 2007, A formally verified proof of the prime number theorem
- Avigad and others, 2025, A Proof-Producing Compiler for Blockchain Applications
- Avigad and others, 2017, A Formally Verified Proof of the Central Limit Theorem

- Avvenuti and others, 2003, Java bytecode verification for secure information flow
- Azeemi, 2006, Compiler Directed Battery-Aware Implementation of Mobile Applications
- Aziz and Labiche, 2026, ProtoSYCL A Sample Implementation of a SYCL Compiler for Conformance Test Suite Development
- Azmi, 2025, LLM-Aware Static Analysis Adapting Program Analysis to Mixed Human/AI Codebases at Scale
- Azzahra and others, 2020, PRE STACK DEPTH MIGRATION UNTUK KOREKSI EFEK PULL UP DENGAN MENGGUNAKAN METODE HORIZON BASED DEPTH TOMOGRAPHY PADA LAPANGAN 'A1 DAN A2'
- Badler, 1996, A Task Networking and Visual Programming Language for Jack
- Baek and Kim, 2017, Prototype implementation of the OpenGL ES 2.0 shading language offline compiler
- Baek and Lee, 2024, CaLLi OCaml library for static analysis of LLVM bitcode
- Baev, 2009, Techniques for Region-Based Register Allocation
- Baggett, 2000, An Abstract Interpretation of the Wavelet Dimension Function Using Group Representations
- Baghdadi and others, 2019, Tiramisu A Polyhedral Compiler for Expressing Fast and Portable Code
- Bai and others, 2025, APCer An Agile Physical Compiler for Multi-Port Register File
- Bailin and others, 1993, Model-based reasoning for system and software engineering The Knowledge From Pictures KFP environment
- Baird and Johnson, 1977, COBOL Compiler Validation System, 1974. Version 3.0
- Baird and Oliver, 1977, Programming Language Standards – Who Needs Them
- Baltes and Diehl, 2018, Usage and attribution of Stack Overflow code snippets in GitHub projects
- Baltes and others, 2019, SOTorrent Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets
- Balu and Saraswathi, 2014, Implementation of SAAS Compiler in Intranet
- Banerjee and Karfa, 2018, Compiler-agnostic Translation Validation
- Bansal and Singh, 2011, Dual Stack Implementation of Mobile IPv6 Software Architecture
- Bao and others, 2014, PWCET Power-Aware Worst Case Execution Time Analysis
- Baradaran and others, 2026, Reusing Legacy Code in Wasm Key Challenges of Compilation and Code Semantics Preservation
- Barai and others, 2023, LLVM Static Analysis for Program Characterization and Memory Reuse Profile Estimation
- Baramashetru and others, 2025, A Type System for Data Privacy Compliance in Active Object Languages
- Barany, 2018, Finding missed compiler optimizations by differential testing
- Barash and Shchur, 2013, RNGSSELIB Program library for random number generation. More generators, parallel streams of random numbers and Fortran compatibility
- Barbosa and others, 2025, Formally Verified Correctness Bounds for Lattice-Based Cryptography
- Barbuti and Cataudella, 2004, Java bytecode verification on Java cards
- Barbuti and others, 2002, Fixing the Java bytecode verifier by a suitable type domain
- Bardonek and Zachariasova, 2026, Leveraging design static analysis for vertical reuse Analytical interpretation and scalability behavior
- Barinov and others, 2020, Applying compiler-based binary watermarking technology to ensure binary compatibility in GNU/Linux distribution
- Baroffio and Reghenzani, 2023, Compiler-Injected SIHFT for Embedded Operating Systems
- Barrière and others, 2021, Formally verified speculation and deoptimization in a JIT compiler
- Barrière and others, 2023, Formally Verified Native Code Generation in an Effectful JIT

Turning the CompCert Backend into a Formally Verified JIT Compiler
- Barthe and others, 2017, Verified Translation Validation of Static Analyses
- Bartlett and others, 2010, Accurate Determination of Loop Iterations for Worst-Case Execution Time Analysis
- Bartsch and others, 2021, Compositional Fault Propagation Analysis in Embedded Systems using Abstract Interpretation
- Basin and others, 2003, Bytecode Verification by Model Checking
- Baskaran and others, 2016, Automatic Code Generation and Data Management for an Asynchronous Task-Based Runtime
- Basold and Geuvers, 2016, Type Theory based on Dependent Inductive and Coinductive Types
- Bassil, 2019, Compiler Design for Legal Document TranslationIn Digital Government
- Basu and others, 2015, Compiler-Directed Transformation for Higher-Order Stencils
- Basu and others, 2017, Compiler-based code generation and autotuning for geometric multigrid on GPU-accelerated supercomputers
- Bate and Kazakov, 2008, New Directions in Worst-Case Execution Time analysis
- Battle and others, 2022, Exploring D3 Implementation Challenges on Stack Overflow
- Bau and others, 2022, Abstract interpretation of Michelson smart-contracts
- Bauckholt and Holz, 2024, WebAssembly as a Fuzzing Compilation Target Registered Report
- Bauer and Pretnar, 2014, An Effect System for Algebraic Effects and Handlers
- Bauer and Pretnar, 2015, Programming with algebraic effects and handlers
- Bauml and Brada, 2011, Reconstruction of Type Information from Java Bytecode for Component Compatibility
- Bavera and Bonelli, 2008, Type-based information flow analysis for bytecode languages with variable object field policies
- Bayer and others, 1968, MPL MATHEMATICAL PROGRAMMING LANGUAGE
- Bayley and Shiel, 2005, JVM Bytecode Verification Without Dataflow Analysis
- Beaumont and others, 2018, Fast approximation algorithms for task-based runtime systems
- Becker and Chakraborty, 2018, Optimizing Worst-Case Execution Times Using Mainstream Compilers
- Becker and others, 2018, Scalable and precise estimation and debugging of the worst-case execution time for analysis-friendly processors a comeback of model checking
- Bee and bernardo, 2018, Predicting Fork Visibility Performance on Programming Language Interoperability in Open Source Projects
- Bekiris, 2021, Implementation of UTASTAR Decision Support System in VBA Programming Language
- Belyaev and others, 2018, Comparative Analysis of Two Approaches to Static Taint Analysis
- Benito-Montoro and others, 2021, A Tool to Assist the Compiler Construction Instructor in Checking the Equivalence of Specifications Based on Regular Expressions
- Benzaken and Contejean, 2019, A Coq mechanised formal semantics for realistic SQL queries formally reconciling SQL and bag relational algebra
- Béra and others, 2016, Practical Validation of Bytecode to Bytecode JIT Compiler Dynamic Deoptimization
- Berdine and others, 2002, Linear Continuation-Passing
- Berg and others, 2025, Manim-DFA Visualising Data Flow Analysis and Abstract Interpretation Algorithms with Automated Video Generation
- Berger and others, 2025, Translation Validation for LLVM's AArch64 Backend
- Berger and Spreen, 2016, A coinductive approach to computing with compact sets
- Berghofer and Strecker, 2004, Extracting a formally verified, fully executable compiler from a proof assistant
- Berlakovich and others, 2026, LOOL Low-Overhead, Optimization-Log-Guided Compiler

Fuzzing
- Berlakovich and others, 2026, LOOL Low-Overhead, Optimization-Log-Guided Compiler Fuzzing - RCR Report
- Bernardeschi and others, 2004, Checking secure information flow in Java bytecode by code transformation and standard bytecode verification
- Bernardeschi and others, 2008, Decomposing bytecode verification by abstract interpretation
- Bernardeschi and others, 2006, Using Control Dependencies for Space-Aware Bytecode Verification
- Bernat and Burns, 2000, An Approach to Symbolic Worst-Case Execution Time Analysis
- Berten and others, 2009, Managing Imprecise Worst Case Execution Times on DVFS Platforms
- Bertholon and others, 2024, Interactive Source-to-Source Optimizations Validated using Static Resource Analysis
- Bertrane and others, 2010, Static Analysis and Verification of Aerospace Software by Abstract Interpretation
- Bertrane and others, 2015, Static Analysis and Verification of Aerospace Software by Abstract Interpretation
- Besson and others, 2018, CompCertS A Memory-Aware Verified C Compiler Using a Pointer as Integer Semantics
- Besson and others, 2019, Information-Flow Preservation in Compiler Optimisations
- Bezrąk and Przyłucki, 2020, Impact of the cloud application programming language on the performance of its implementation in selected serverless environments
- Bhamidipati and Vemuri, 2023, ASPIRE An Intermediate Representation for Abstract Security Policies
- Bhasker, 1988, Implementation of an optimizing compiler for VHDL
- Bhoyar and others, 2025, Sign Language Interpreter using Long Short-Term Memory LSTM
- Bhushan and Yadav, 2020, Verification of Virtual Machine Architecture in a Hypervisor through Model Checking
- Bicarregui and others, 2006, The verified software repository a step towards the verifying compiler
- Pause ‘n’ Play: Formalizing Asynchronous C#
- Biernacki and others, 2005, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations
- Biernacki and others, 2005, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations Preliminary Version
- Biernacki and Danvy, 2005, A Simple Proof of a Folklore Theorem about Delimited Control
- Biernacki and others, 2006, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations
- BIERNACKI and DANVY, 2006, THEORETICAL PEARL A simple proof of a folklore theorem about delimited control
- Biernacki and others, 2015, A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations
- Biernacki and others, 2019, Bisimulations for Delimited-Control Operators
- Biernacki and others, 2017, Handle with care relational interpretation of algebraic effects and handlers
- Bik and others, 2022, Compiler Support for Sparse Tensor Computations in MLIR
- Biradar and others, 2026, Design and Development of an AI-Powered Code Generation System with Persistent Memory and Multi-Agent Architecture
- Bird, 1982, An implementation of a code generator specification language for table driven code generators
- Bishnu and Bhatia, 2018, Algorithmic Compiler based FPGA Implementation of Iterative

Time-Domain Algorithm for Sparse Channel Estimation
- Bispo and Cardoso, 2016, A MATLAB subset to C compiler targeting embedded systems
- Blaak and Van Cutsem, 2026, Towards Least-Privilege WebAssembly Applications Transparent Interposition for WebAssembly Components
- Blanc and Kadobayashi, 2010, Towards revealing JavaScript program intents using abstract interpretation
- Blaß and Philippsen, 2019, GPU-accelerated fixpoint algorithms for faster compiler analyses
- Blazy, 2020, From Verified Compilation to Secure Compilation a Semantic Approach
- Blazy, 2023, CompCert A Journey through the Landscape of Mechanized Semantics for Verified Compilation Keynote
- Blazy and others, 2015, Verified Validation of Program Slicing
- Blieberger, 2002, Data-Flow Frameworks for Worst-Case Execution Time Analysis
- Blower, 1984, An efficient implementation of visibility in Ada
- Bockisch and others, 2020, Java Bytecode Verification with OCL Why, How and Whenc
- Boda and others, 2026, CPerfSmith A Randomized C Program Generator for Performance-Oriented Compiler Testing
- Bodin and others, 2015, Certified Abstract Interpretation with Pretty-Big-Step Semantics
- Boding, 1996, Pre-stack depth migration in three dimensions with the imaging wave machine
- Bodwin and others, 1982, Experience with an experimental compiler generator based on denotational semantics
- Böhme and others, 2009, HOL-Boogie-An Interactive Prover-Backend for the Verifying C Compiler
- Bohnet and Dollner, 2007, CGA Call Graph Analyzer - Locating and Understanding Functionality within the Gnu Compiler Collection's Million Lines of Code
- Boldo and others, 2013, A Formally-Verified C Compiler Supporting Floating-Point Arithmetic
- Bonar and Liffick, 1987, A Visual Programming Language for Novices
- Bonchi and Pous, 2015, Hacking nondeterminism with induction and coinduction
- Bond, 2013, Session details Garbage collection, runtime, and cache management
- Bonkowski and others, 1979, Porting the Zed compiler
- Bonyun, 1979, Euclid Compiler for PDP-11
- Bonyun and Holt, 1978, EUCLID Compiler for PDP-11
- Boo and Lee, 2025, Finding Device Driver Bugs With Fuzzing PCIe Configuration Input
- BOOK and others, 1963, A ONE PASS JOVIAL COMPILER
- Boonriong and others, 2025, Compiler Fuzzing in Continuous Integration A Case Study on Dafny
- Bosse, 2018, A Unified System Modelling and Programming Language based on JavaScript and a Semantic Type System
- - and others, 2013, Bounding the Worst-Case Execution Time for the Fixed-Priority Preemptive Systems Based on the Preemption Points
- bounpaserth, 2026, A Study of the Computer Programming Language Implementation in Computer Engineering Students using the Flowgorithm Platform versus Common Programming
- Bourke, 2021, Specification and end-to-end proof of a reactive language and its compiler invited talk
- Bourke and others, 2017, A formally verified compiler for Lustre
- Bourke and others, 2018, Towards a verified Lustre compiler with modular reset
- Bourke and others, 2019, Mechanized semantics and verified compilation for a dataflow synchronous language with reset
- Bourke and others, 2025, Functional Stream Semantics for a Synchronous Block-Diagram Compiler
- Bowman and Ahmed, 2018, Typed closure conversion for the calculus of constructions

- Boyapati, 2004, Session details Register allocation
- Brachthäuser and others, 2018, Effect handlers for the masses
- Brachthäuser and others, 2020, Effects as capabilities effect handlers and lightweight effect polymorphism
- BRACHTHÄUSER and others, 2020, Effekt Capability-passing style for type- and effect-safe, extensible effect handlers in Scala
- Brady, 2013, Programming and reasoning with algebraic effects and dependent types
- Brady and Hammond, 2006, A verified staged interpreter is a verified compiler
- Brandner and Jordan, 2014, Refinement of worst-case execution time bounds by graph pruning
- Brant and others, 2025, IoT-CODIFT Compiler Optimization DIFT for IoT and Embedded Devices
- Breitner, 2015, Formally proving a compiler transformation safe
- Brennan and others, 2020, JIT Leaks Inducing Timing Side Channels through Just-In-Time Compilation
- Briggs and others, 1994, Improvements to graph coloring register allocation
- Brock and others, 2018, PAYJIT space-optimal JIT compilation and its practical implementation
- Brodersen, 1994, Anatomy of a Silicon Compiler
- Bruno and others, 2021, Compiler-assisted object inlining with value fields
- Bruza, 2016, Syntax and operational semantics of a probabilistic programming language with scopes
- Bryant and others, 2025, Certified Knowledge Compilation with Application to Formally Verified Model Counting
- Buchwald and others, 2016, Verified construction of static single assignment form
- Bulej and others, 2016, Beneath the bytecode
- Bunkenburg and Wu, 2024, Making a Curry Interpreter using Effects and Handlers
- Búr and others, 2021, Worst-case Execution Time Calculation for Query-based Monitors by Witness Generation
- Burdy and Pavlova, 2006, Java bytecode specification and verification
- Burgholzer and others, 2026, Focus Session Paper The MQT Compiler Collection A Blueprint for a Future-Proof Quantum-Classical Compilation Framework
- Burn, 1990, A relationship between abstract interpretation and projection analysis
- Burroughs, 2016, Register allocation and spilling using the expected distance heuristic
- Bushnell and others, 2008, Automatic Testcase Generation for Flight Software
- Butler and Johnson, 1993, Formal Methods for Life-Critical Software
- Butterfield and Woodcock, 2006, A “Hardware Compiler” Semantics for Handel-C
- 2017, C# Compiler for Blind
- Caamaño and Guelton, 2018, Easy Jit compiler assisted library to enable just-in-time compilation in C++ codes
- Cabrera and others, 1992, 3-D Pre-Stack Depth Migration Implementation and Case History
- Cabrera Arteaga and others, 2019, Scalable comparison of JavaScript V8 bytecode traces
- Caldwell and Chiba, 2017, Reducing calling convention overhead in object-oriented programming on embedded ARM thumb-2 platforms
- Caliskan and others, 2018, When Coding Style Survives Compilation De-anonymizing Programmers from Executable Binaries
- Callahan and Koblenz, 1991, Register allocation via hierarchical graph coloring
- Calzarossa and others, 2001, Performance issues of an HPF-like compiler
- Cambier and others, 2020, TaskTorrent a Lightweight Distributed Task-Based Runtime System in C++
- Cambou, 2017, A XOR data compiler Combined with physical unclonable function for true random number generation
- 2022, Cameleer A deductive verification tool for OCaml

- Campbell, 2019, Random Compiler for Fast Hamiltonian Simulation
- Campbell and Beck, 1965, THE FORAST PROGRAMMING LANGUAGE FOR ORDVAC AND BRLESC REVISED
- Campos-López and Wannenwetsch, 2016, The PERICLES Process Compiler Linking BPMN Processes into Complex Workflows for Model-Driven Preservation in Evolving Ecosystems
- Canedo and others, 2009, Design and implementation of a queue compiler
- Carlos Paradis and others, 2025, Towards Streamlining Auditing for Compliance With Requirements in Open-Source Software at NASA
- Carlotto and others, 2016, Interoperability of Annotation Schemes Using the Pepper Framework to Display AWA Documents in the ANNIS Interface
- Carminati and others, 2017, Combining loop unrolling strategies and code predication to reduce the worst-case execution time of real-time software
- Carminati and others, 2018, On the use of static branch prediction to reduce the worst-case execution time of real-time applications
- CARNEY and LABAUGH, 1983, Efficient compiler implementation for a spaceborne image processing demonstration system
- Carter, 1982, Further analysis of code generation for a single register machine
- Carvalho and others, 2019, A dataflow runtime environment and static scheduler for edge, fog and in-situ computing
- Cassola and others, 2020, A Gradual Type System for Elixir
- Castagna and others, 2023, The Design Principles of the Elixir Type System
- Castañeda and Rodríguez, 2023, Asynchronous Wait-Free Runtime Verification and Enforcement of Linearizability
- Castañeda and Rodríguez, 2026, Asynchronous Wait-Free Runtime Verification and Enforcement of Linearizability
- Castro-Lopez and Vega-Lopez, 2019, Multi-target Compiler for the Deployment of Machine Learning Models
- Cattell and others, 1979, Code generation in a machine-independent compiler
- Cazzola and Olivares, 2015, Gradually Learning Programming Supported by a Growable Programming Language
- Ceng Giap and Erviana, 2023, Implementation of Face Mask Detection Using Phyton Programming Language
- Chaitin, 1982, Register allocation and spilling via graph coloring
- Chaitin, 2004, Register allocation and spilling via graph coloring
- Chaliasos and others, 2022, Finding typing compiler bugs
- Chalin, 2007, A Sound Assertion Semantics for the Dependable Systems Evolution Verifying Compiler
- Chalin, 2010, Engineering a Sound Assertion Semantics for the Verifying Compiler
- Chaplygin, 2025, Modeling and implementation of Common LISP functional language compiler
- Charmanas and others, 2026, A topic-oriented trend analysis framework for Stack Exchange questions Case study on ChatGPT related queries on Stack Overflow
- Chase, 2005, Session details Register allocation
- Chatterjee, 2013, Runtime Systems for Extreme Scale Platforms
- Chaumette and others, 2011, Automated extraction of polymorphic virus signatures using abstract interpretation
- Chavanon and others, 2024, PfComp A Verified Compiler for Packet Filtering Leveraging Binary Decision Diagrams
- Chawla and others, 2024, COMPARATIVE STUDY OF PROPOFOL AUTO-COINDUCTION VERSUS KETAMINE PROPOFOL COINDUCTION USING PRIMING PRINCIPLE BY BISPECTRAL INDEX ANALYSIS FOR DAY CARE SURGERY
- Cheatham and Standish, 1970, Optimization aspects of compiler- compilers
- Chen, 2018, Learning to accelerate compiler testing

- A survey of compiler testing
- Chen and others, 2025, De-duplicating Silent Compiler Bugs via Deep Semantic Representation
- Chen and others, 2007, Design of a Certifying Compiler Supporting Proof of Program Safety
- Chen and others, 2016, A Process-Visible Compiler Aimed for Teaching Assistant
- Chen and others, 2022, Register allocation compilation technique for ASIP in 5G micro base stations
- Chen and others, 2020, Enhanced compiler bug isolation via memoized search
- Chen and Suo, 2022, Boosting Compiler Testing via Compiler Optimization Exploration
- Chen and others, 2010, Implementation of Bytecode-based Software Watermarking for Java Programs
- Chen and others, 2009, Bytecode Generation for XQuery Compiler
- Chen and others, 2020, CRAC An automatic assistant compiler of checkpoint/restart for OpenCL program
- Chen and others, 2024, Design and Implementation of an Aspect-Oriented C Programming Language
- Chen and Zong, 2016, Android App Energy Efficiency The Impact of Language, Runtime, Compiler, and Implementation
- Cheng, 2023, QA4C An Intelligent Question and Answering System for the C Programming Language Based on Knowledge Graph
- Cheng and others, 2016, Formalised EMFTVM bytecode language for sound verification of model transformations
- Cheng and Wu, 2026, Toward Unified Chinese Multi-Dialectal Speech Recognition via Pinyin Intermediate Representation
- Cheng and others, 2026, Denotation-based Compositional Compiler Verification
- Chen Zhao and others, 2009, Automated test program generation for an industrial optimizing compiler
- Chernenko and others, 2021, Proving Reflex Program Verification Conditions in Coq Proof Assistant
- Chhak and others, 2021, Towards formally verified compilation of tag-based policy enforcement
- Chi, 1994, Compiler Optimization Technique for Data Cache Prefetching Using a Small CAM Array
- Chi and others, 2009, An Improved Bytecode Verification Algorithm on Java Card
- Chichereau and others, 2024, Fully Integrated Quantum Method for Classical Register Allocation in LLVM
- Ching, 1986, Program analysis and code generation in an APL/370 compiler
- Ching and Katz, 1993, The testing of an APL compiler
- Chirila and Sora, 2024, Java Single vs. Platform vs. Virtual Threads Runtime Performance Assessment in the Context of Key Class Detection
- Chlipala, 2010, A verified compiler for an impure functional language
- Chlipala, 2015, Session details Session 4A Compiler Correctness
- Chmiel and Spinolo, 2022, Testing the impact of remote interpreting settings on interpreter experience and performance
- Cho, 2016, Semantics for a Quantum Programming Language by Operator Algebras
- Cho and others, 2022, Sequential reasoning for optimizing compilers under weak memory concurrency
- Choi and Han, 2006, Optimal register reassignment for register stack overflow minimization
- Choi and Hong, 2019, Design and implementation of virtual machine control and streaming scheme using Linux kernel-based virtual machine hypercall for virtual mobile infrastructure
- Choi and Jeon, 2026, Stack-based static WebAssembly binary slicing and mutation for

generating valid sub-binaries
- Choi and others, 2018, Work-in-Progress Lightweight Deadlock Detection Technique for Embedded Systems via OS-Level Analysis
- Choi and others, 2024, Accelerating Sensor Software Performance on Edge Devices Through Just-in-Time Compilation
- Choi and others, 2019, Reusable inline caching for JavaScript performance
- Chong and others, 2017, Programming languages and compiler design for realistic quantum hardware
- Christodorescu and Jha, 2006, Static Analysis of Executables to Detect Malicious Patterns
- Christopher and others, 1984, Using dynamic programming to generate optimized code in a Graham-Glanville style code generator
- Chudnov and others, 2014, Information Flow Monitoring as Abstract Interpretation for Relational Logic
- Chung Seo and Kim, 2023, Portable and Efficient Implementation of CRYSTALS-Kyber Based on WebAssembly
- Ciaffaglione, 2016, Towards Turing computability via coinduction
- Ciesko and Roussel, 2023, User-Level Threading for HPC Applications
- Cieszewski, 2025, PyHLS Intermediate Representation for Versatile High-Level Synthesis
- C K, 2022, Video Calling With Build-In Compiler
- CLERC, 2016, OCaml-Java The Java Virtual Machine as the target of an OCaml compiler
- Clinger, 1984, The scheme 311 compiler an exercise in denotational semantics
- Clopper and Pearson 1934, The use of confidence or fiducial limits illustrated in the case of the binomial
- Coglio, 2003, Improving the official specification of Java bytecode verification
- Coglio, 2004, Simple verification technique for complex Java bytecode subroutines
- Colin and Puaut, 2000, Worst Case Execution Time Analysis for a Processor with Branch Prediction
- Colombet and others, 2011, Graph-coloring and treescan register allocation using repairing
- Colvin and Hayes, 2011, Structural operational semantics through context-dependent behaviour
- 2016, Comparative Study of Virtual Machine Migration Techniques and Challenges in Post Copy Live Virtual Machine Migration
- 1998, Compiler technology Tools, translators and language implementation
- Cong and Asai, 2023, Towards a Reflection for Effect Handlers
- Consel and Cheng Khoo, 1993, Semantics-directed generation of a prolog compiler
- Design of a separable transition-diagram compiler
- Coppo and Ferrari, 1993, Type inference, abstract interpretation and strictness analysis
- Cordes and others, 2009, A Fast and Precise Static Loop Analysis Based on Abstract Interpretation, Program Slicing and Polytope Models
- Correnson and Finkbeiner, 2025, Coinductive Proofs for Temporal Hyperliveness
- Corrias and others, 2026, An Analysis of Modern Web Security Vulnerabilities Inside WebAssembly Applications
- Corrodi and others, 2018, A semantics comparison workbench for a concurrent, asynchronous, distributed programming language
- Cortesi, 2008, Widening Operators for Abstract Interpretation
- Cortesi and Filé, 1991, Abstract interpretation of logic programs
- Cortesi and others, 1997, Complementation in abstract interpretation
- Corti and Gross, 2004, Approximation of the worst-case execution time using structural analysis
- Costa and others, 2021, Use of Measurements in Worst-Case Execution Time Estimation for Real-Time Systems

- Costagliola and others, 2007, Visual language implementation through standard compiler-compiler techniques
- Couch and Hamm, 1977, Semantic Structures for Efficient Code Generation on a Stack Machine
- Cousot and Cousot 1977, Abstract interpretation
- Cousot, 1996, Program analysis
- Cousot, 1997, Program analysis
- Cousot, 2015, On Various Abstract Understandings of Abstract Interpretation
- Cousot, 2015, Verification by abstract interpretation, soundness and abstract induction
- Cousot and Cousot, 1995, Formal language, grammar and set-constraint-based program analysis by abstract interpretation
- Cousot and Cousot, 2000, Temporal abstract interpretation
- Cousot and Cousot, 2001, A Case Study in Abstract Interpretation Based Program Transformation
- Cousot and Cousot, 2002, Systematic design of program transformation frameworks by abstract interpretation
- Cousot and Cousot, 2011, Grammar semantics, analysis and parsing by abstract interpretation
- Cousot and Cousot, 2012, An abstract interpretation framework for termination
- Cousot and Cousot, 2014, A galois connection calculus for abstract interpretation
- Cowan and Graham, 1970, Design characteristics of the WATFOR compiler
- Coward and others, 2023, Combining E-Graphs with Abstract Interpretation
- C. Robinson and others, 1994, Prospect definition by pre-stack depth migration of a grid of seismic lines - A case history
- Cruttwell and others, 2021, Categorical semantics of a simple differential programming language
- Cummins and others, 2018, Compiler fuzzing through deep learning
- Cummins and others, 2025, LLM Compiler Foundation Language Models for Compiler Optimization
- Cunha and others, 2024, Trading Runtime for Energy Efficiency Leveraging Power Caps to Save Energy across Programming Languages
- Curry, 2023, An HPC-Oriented Runtime Environment for Enabling Computational Storage
- Curtis and others, 2018, A compiler for cyber-physical digital microfluidic biochips
- Czajka, 2020, A new coinductive confluence proof for infinitary lambda calculus
- Czajka, 2020, An operational interpretation of coinductive types
- Dagnino, 2020, Coaxioms flexible coinductive definitions by inference systems
- Dagnino, 2021, Foundations of regular coinduction
- Daiß and others, 2023, Stellar Mergers with HPX-Kokkos and SYCL Methods of using an Asynchronous Many-Task Runtime System with SYCL
- Dakkak and others, 2020, The design and implementation of the wolfram language compiler
- Dalla Preda and Giacobazzi, 2005, Control code obfuscation by abstract interpretation
- Dams and others, 1997, Abstract interpretation of reactive systems
- Dange and others, 2023, Implementation on A User Authentication Scheme Using Block Chain-Enabled Fog Nodes
- Abstracting control
- Defunctionalization at work
- Das and others, 2020, Deep Learning-based Approximate Graph-Coloring Algorithm for Register Allocation
- Dasgupta and others, 2019, A complete formal semantics of x86-64 user-level instruction set architecture
- Dave and others, 2016, A 1V 800MHz 140Kb register file compiler using variation aware self-timing in 40nm bulk CMOS

- Davenport, 1995, Object-Oriented Visual Programming Language. Phase 1
- De and others, 2021, Canonical proof-objects for coinductive programming infinets with infinitely many cuts
- Debenham and Westlake, 2014, Pre-stack depth migration for improved imaging under seafloor canyons 2D case study of Browse Basin, AustraliaFN1
- De Blaere and others, 2021, A Compiler Extension to Protect Embedded Systems Against Data Flow Errors
- Debnath and others, 2024, ARMOR A Formally Verified Implementation of X.509 Certificate Chain Validation
- De Bosschere and Tarau, 1994, High performance continuation passing style Prolog-to-C mapping
- Debray, 1995, Abstract interpretation and low-level code optimization
- DeBuhr and others, 2017, Scalable Hierarchical Multipole Methods Using an Asynchronous Many-Tasking Runtime System
- de Ferrière and others, 2026, SecSwift, a Compiler-Based Framework for Software Countermeasures in Cybersecurity
- De Francesco and others, 2010, Using abstract interpretation to add type checking for interfaces in Java bytecode verification
- Dejtrakulwong and others, 2013, Sensitivity analysis and cascaded interpretation scheme for subtle seismic signatures in thin shaly-sand reservoirs
- Delaët and others, 2025, Abstract machines and small-step semantics a winning ticket for proof automation?
- Delgado-Pérez and Segura, 2019, Study of trivial compiler equivalence on C++ object-oriented mutation operators
- De Luca and Chen, 2017, Visual IoT/Robotics Programming Language in Pi-Calculus
- del Vado Vírseda, 2021, Learning Compiler Design From the Implementation to Theory
- DelVado Vírseda, 2023, Visualizing Compiler Design Theory from Implementation Through an Interactive Tutoring Tool Experiences and Results
- De Macedo and others, 2021, On the Runtime and Energy Performance of WebAssembly Is WebAssembly superior to JavaScript yet?
- De Macedo and others, 2022, WebAssembly versus JavaScript Energy and Runtime Performance
- Demmler and others, 2021, Improved Circuit Compilation for Hybrid MPC via Compiler Intermediate Representation
- Revisiting coroutines
- Deng and others, 2026, Design and Implementation of an OCaml-Based Standalone SystemVerilog Preprocessor Compliant with IEEE 1800-2023
- Deng and Namjoshi, 2018, Securing a compiler transformation
- Denis and others, 2023, Tracing task-based runtime systems Feedbacks from the StarPU case
- de Niz, 2007, Diagrams and Languages for Model-Based Software Engineering of Embedded Systems UML and AADL
- Denney and Fischer, 2005, Formal Safety Certification of Aerospace Software
- Denney and Fischer, 2009, Generating Code Review Documentation for Auto-Generated Mission-Critical Software
- Denzler and others, 2021, Experiences from Adjusting Industrial Software for Worst-Case Execution Time Analysis
- De Paula and Ierusalimschy, 2022, A Foreign Function Interface for Pallene
- Derakhshan and others, 2023, Towards End-to-End Verified TEEs via Verified Interface Conformance and Certified Compilers
- Deransart, 1979, Proof by semantic attributes of a LISP compiler
- Desai, 2009, A Novel Technique for Orchestration of Compiler Optimization Functions Using Branch and Bound Strategy
- Deutsch, 1995, Semantic models and abstract interpretation techniques for inductive

data structures and pointers
- 2023, Development of a mobile robot control system in the Python programming language using Raspberry Pi
- de Vilhena and Pottier, 2021, A separation logic for effect handlers
- DeVries and others, 2009, ActionScript bytecode verification with co-logic programming
- DeVries and others, 2009, ActionScript bytecode verification with co-logic programming abstract only
- de Werra and others, 1999, On a graph-theoretical model for cyclic register allocation
- Diamantopoulos and others, 2020, Agile Autotuning of a Transprecision Tensor Accelerator Overlay for TVM Compiler Stack
- Dickerson and others, 2026, Practical Python FPGA Acceleration with Fast Just-In-Time Compilation and Configuration
- Dico and Tata Sutabri, 2025, Virtual Desktop Infrastructure Sebagai Pendukung Perkuliahan Dengan Algoritma Virtual Machine
- Diehl and others, 2021, Performance Measurements Within Asynchronous Task-Based Runtime Systems A Double White Dwarf Merger as an Application
- Di Lavore and others, 2025, Coinductive Streams in Monoidal Categories
- Dimovski, 2021, Lifted termination analysis by abstract interpretation and its applications
- Dimovski, 2025, Imperative Program Synthesis by Abstract Static Analysis and SMT Mutations
- Ding and others, 2022, CARL Compiler Assigned Reference Leasing
- Ding and others, 2025, HFF-JIT A Hybrid Fuzzing Framework for JIT Compiler Vulnerability Detection in JavaScript
- Ding and Zhang, 2010, Loop-Based Instruction Prefetching to Reduce the Worst-Case Execution Time
- Dinikeev, 2025, Type system for a statically typed concatenative programming language with first class function support
- Di Pierro and others, 2008, Relational Analysis and Precision via Probabilistic Abstract Interpretation
- Dissegna and others, 2016, An Abstract Interpretation-Based Model of Tracing Just-in-Time Compilation
- Ditu, 2015, Model-Based Function Call Code Generation and Stack Management in Retargetable Compilers Application Binary Interface Modeling of Stack Layout and Function Call Sequence
- Dixon, 1982, A Pascal compiler testing facility
- Djalali and others, 2025, The Evolution of Software Usability in Developer Communities An Empirical Study on Stack Overflow
- Dobravec, 2018, JAVA BYTECODE INSTRUCTION USAGE COUNTING WITH ALGATOR
- Dold and others, 2003, A Completely Verified Realistic Bootstrap Compiler
- Domagała and others, 2016, Register allocation and promotion through combined instruction scheduling and loop unrolling
- Donaldson and others, 2023, Industrial Deployment of Compiler Fuzzing Techniques for Two GPU Shading Languages
- Donaldson and others, 2024, Randomised Testing of the Compiler for a Verification-Aware Programming Language
- Donegan and others, 1979, A code generator generator language
- Dong, 2013, Design and Implementation of Compiler Subsystem for Object Oriented Publish/Subscribe Systems
- Dong, 2015, RSTVL A Sound Abstract Memory Model for Program Static Analysis
- Dong, 2018, A sound abstract memory model for static analysis of C programs
- DONG and others, 2011, Logic System for Bytecode Program Modular Certification
- Dong and others, 2026, Presynthesis Towards Scaling Up Program Synthesis with Finer-Grained Abstract Semantics

- Donohoe, 1987, A Survey of Real-Time Performance Benchmarks for the Ada Programming Language
- Doronin, 2019, PROBLEM RESEARCH AND DEVELOPMENT OF A TOOL FOR CHECKING APPLICATION BINARY INTERFACE COMPATIBILITY OF VIRTUAL METHOD TABLES
- DOWNEN and ARIOLA, 2014, Delimited control and computational effects
- DOWNEN and ARIOLA, 2025, A contextual formalization of structural coinduction
- Downen and others, 2016, Sequent calculus as a compiler intermediate language
- Drechsler and Schnieber, 2023, Automated Polynomial Formal Verification Human-Readable Proof Generation
- Drescher and Engelke, 2024, Fast Template-Based Code Generation for MLIR
- D'Silva and others, 2015, The Correctness-Security Gap in Compiler Optimization
- Du, 2026, Performance Verification of BFS For Unweighted Maze Solving A Comparative Analysis with DFS and A* Via Ocaml Implementation
- Dubach and others, 2007, Fast compiler optimisation evaluation using code-feature based performance prediction
- Duhamel and Pillement, 2024, Runtime Task Scheduling for FPGA-Based Embedded Systems Using Just-in-Time Bitstream Prefetching
- Duhoux and others, 2019, Implementation of a Feature-Based Context-Oriented Programming Language
- Duţu and others, 2026, From Runtime Reflection to Compile-Time Specialization A Template-Based Approach to Runtime Libraries
- A monadic framework for delimited continuations
- E., 2020, Modified Support Vector Machine based Efficient Virtual Machine Consolidation Procedure for Cloud Data Centers
- Eachempati and others, 2010, An open-source compiler and runtime implementation for Coarray Fortran
- Edalat and Pattinson, 2007, Denotational semantics of hybrid automata
- Толстікова and others, 2016, Efficiency asynchronous application programming language Python
- 2015, Efficient Implementation of Class Based Decomposition Schemes for Naive Bayes Classifier
- Efthymiou and others, 2022, Quantum simulation with just-in-time compilation
- Egreteau and Thierry, 2005, Attenuating the Effects of Pre-Stack Depth Migration for AVA Analysis
- Eichenberger and Davidson, 1995, Register allocation for predicated code
- Eisl, 2015, Trace register allocation
- Eisl and others, 2016, Trace-based Register Allocation in a JIT Compiler
- Eisl and others, 2018, Parallel trace register allocation
- ELECTRONIC SYSTEMS DIV HANSCOM AFB MA, 1970, USER'S MANUAL JOVIAL COMPILER VALIDATION SYSTEM
- Kotlin coroutines: design and implementation
- Elkhouly and others, 2019, Compiler-support for Critical Data Persistence in NVM
- Elof Frank, 1995, Constrained Register Allocation in Bus Architectures
- Emerson, 2020, Autoencoding Pixies Amortised Variational Inference with Graph Convolutions for Functional Distributional Semantics
- Endo and others, 2017, On the Interactions Between Value Prediction and Compiler Optimizations in the Context of EOLE
- Engblom and others, 2003, Worst-case execution-time analysis for embedded real-time systems
- Engelke and Schwarz, 2024, Compile-Time Analysis of Compiler Frameworks for Query Compilation
- Engelke and Weidendorfer, 2017, Using LLVM for Optimized Lightweight Binary Re-Writing at Runtime

- Enrici and others, 2019, Efficient Data-Flow Analysis of UML/SysML Diagrams for Optimized Model Compilation of Hardware-software Systems
- Ermedahl and others, 2009, Deriving the Worst-Case Execution Time Input Values
- Ermedahl and Puschner, 2011, Preface to the special issue on worst-case execution-time analysis
- Ermedahl and others, 2003, Clustered calculation of worst-case execution times
- Ertl, 1995, Stack caching for interpreters
- Evans and others, 1978, A Compiler Compiler and Methodology for Problem Oriented Language Compiler Implementors
- Evansi and Sulaiman, 1996, Solving optimisation problems using neucomp-a neural network compiler
- Facchinetti and others, 2019, Higher-order Demand-driven Program Analysis
- Falis, 1982, Design and implementation in Ada of a runtime task supervisor
- Falk, 2009, WCET-aware register allocation based on graph coloring
- Falk and Lokuciejewski, 2010, A compiler framework for the reduction of worst-case execution times
- Falk and Lokuciejewski, 2019, Correction to A compiler framework for the reduction of worst-case execution times
- Fan and others, 2025, Adaptive random compiler for Hamiltonian simulation
- Fan and others, 2024, History-driven Compiler Fuzzing via Assembling and Scheduling Bug-triggering Code Segments
- Fang and others, 2026, LLM-VeriOpt Verification-Guided Reinforcement Learning for LLM-Based Compiler Optimization
- Farkas and others, 2021, Improving productivity in large scale testing at the compiler level by changing the intermediate language from C++ to Java
- Fayzrakhmanov, 2022, Introducing Programming Language Metrics
- Fedasyuk and others, 2017, Architecture of a tool for automated testing the worst-case execution time of real-time embedded systems' firmware
- Feitosa and Ribeiro, 2024, Differential Testing using Random Well-Typed Haskell Programs
- Feitosa and others, 2019, A monadic semantics for quantum computing in an object oriented language
- Feldman, 1966, A formal semantics for computer languages and its application in a compiler-compiler
- Feldman, 1979, Implementation of a portable Fortran 77 compiler using modern tools
- Felker, 2020, Design of Cyanobyte An Intermediate Representation to Standardize Digital Peripheral Datasheets for Automatic Code Generation
- The theory and practice of first-class prompts
- Feng and others, 2025, Finding Compiler Bugs through Cross-Language Code Generator and Differential Testing
- Ferdinand and Heckmann, 2008, Worst-Case Execution Time - A Tool Provider's Perspective
- Feret and others, 2007, Reachability Analysis of Biological Signalling Pathways by Abstract Interpretation
- Ferrante and Allard, 1996, Introducing a CPS style optimizer into an existing compiler
- 2005, Fifth IEEE International Workshop on Source Code Analysis and Manipulation
- Finkel and others, 2019, ClangJIT Enhancing C++ with Just-in-Time Compilation
- Finkelstein, 1968, A compiler optimization technique
- Fisher, 1976, A Common Programming Language for the Department of Defense–Background and Technical Requirements
- Adding delimited and composable control to a production programming environment
- Flatt and Dybvig, 2020, Compiler and runtime support for continuation marks
- Fonseca and others, 2017, An Empirical Study on the Correctness of Formally Verified Distributed Systems

- 1995, Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems A Practitioner's Companion - Volume 2
- 1995, Formal Methods Specification and Verification Guidebook for Software and Computer Systems Planning and Technology Insertion - Volume 1
- FORSTER and others, 2019, On the expressive power of user-defined effects Effect handlers, monadic reflection, delimited control
- 2004, Fourth IEEE International Workshop on Source Code Analysis and Manipulation
- Fox and others, 2017, Verified compilation of CakeML to multiple machine-code targets
- Fradet and others, 2018, A Generic Coq Proof of Typical Worst-Case Analysis
- Francalanza and Tabone, 2023, ElixirST A session-based type system for Elixir modules
- Fraser and Wendt, 1986, Integrating code generation and optimization
- Frąszczak and Frąszczak, 2026, PhishingWebCollector Async python library for automated phishing feed collection
- Freier and Jian-Jia Chen, 2013, Prioritization for real-time embedded systems on dual-core platforms by exploiting the typical- and worst-case execution times
- Frenkel and others, 2011, Towards a Modular and Accessible Modelica Compiler Backend
- Freund and Mitchell, 1998, A type system for object initialization in the Java bytecode language
- Freund and Mitchell, 1998, A Type System for Object Initialization In the Java Bytecode Language summary
- Freund and Mitchell, 1999, A type system for object initialization in the Java bytecode language
- Freund and Mitchell, 2003, A Type System for the Java Bytecode Language and Verifier
- Fried and others, 2023, Register Allocation for Compressed ISAs in LLVM
- Friers and others, 2021, Amortised Encoding for Large High-Resolution Displays
- Friese and others, 2024, Lamellar A Rust-based Asynchronous Tasking and PGAS Runtime for High Performance Computing
- Frigo, 1999, A fast Fourier transform compiler
- Fruehwirth, 2025, Runtime Repeated Recursion Unfolding in CHR A Just-In-Time Online Program Optimization Strategy That Can Achieve Super-Linear Speedup
- Fumero and Kotselidis, 2018, Using compiler snippets to exploit parallelism on heterogeneous hardware a Java reduction case study
- Fumero and others, 2017, Just-In-Time GPU Compilation for Interpreted Languages with Partial Evaluation
- Fusaoka and Hirayama, 1982, Compiler chip
- Fusi and others, 2020, On the Use of Probabilistic Worst-Case Execution Time Estimation for Parallel Applications in High Performance Systems
- Gaál, 1993, Parallel compiler generation
- Gaißert and others, 2025, Tracing Just-in-Time Compilation for Effects and Handlers
- Gal and others, 2005, Integrated Java Bytecode Verification
- Gal and others, 2008, Java bytecode verification via static single assignment form
- Gallaher, 1982, Investigate Capability of Ada Higher Order Programming Language for Developing Machine Independent Software
- Galloway and others, 2015, Performance Metrics of Virtual Machine Live Migration
- Gan and Li, 2015, Coinduction functor in representation stability theory
- Ganzinger and others, 1982, A truly generative semantics-directed compiler generator
- Ganzinger and others, 1976, MUG1 - an incremental compiler-compiler
- Gao and others, 2014, A Method of Binary Code Variable Interval Analysis Based on Abstract Interpretation
- Gao and Parreaux, 2025, A Lightweight Type-and-Effect System for Invalidation Safety Tracking Permanent and Temporary Invalidation with Constraint-Based Subtype Inference
- Gao and Shi, 2005, An improved approach of register allocation via graph coloring

- Gao and others, 2025, Clozemaster Fuzzing Rust Compiler by Harnessing Llms for Infilling Masked Real Programs
- Garcia and others, 2009, Lazy evaluation and delimited control
- Garcia and others, 2010, Lazy Evaluation and Delimited Control
- Gaudet and Stoodley, 2016, Rebuilding an airliner in flight a retrospective on refactoring IBM testarossa production compiler for Eclipse OMR
- Geeson and Smith, 2024, Compiler Testing with Relaxed Memory Models
- Genaim and Zanardini, 2013, Reachability-based acyclicity analysis by Abstract Interpretation
- Geng and others, 2021, Verification of Open-Source Memory Compiler Framework with a Practical PDK
- Georgakoudis and others, 2025, Proteus Portable Runtime Optimization of GPU Kernel Execution with Just-in-Time Compilation
- Georgescu and others, 2024, Evolutionary Generative Fuzzing for Differential Testing of the Kotlin Compiler
- Georgiou and others, 2020, Lost In Translation Exposing Hidden Compiler Optimization Opportunities
- Gerasimov, 2018, Directed Dynamic Symbolic Execution for Static Analysis Warnings Confirmation
- Ghadesi and others, 2024, What causes exceptions in machine learning applications? Mining machine learning-related stack traces on Stack Overflow
- Ghica and Alyahya, 2017, On the Learnability of Programming Language Semantics
- Ghica and others, 2022, High-level effect handlers in C++
- Ghica and Tapus, 2015, Optimized retargetable compiler for embedded processors - GCC vs LLVM
- Ghorbani and Babamir, 2019, Runtime deadlock tracking and prevention of concurrent multithreaded programs A learning-based approach
- Ghosh and others, 2020, Compiler compatible 5.66 Mb/mm2 8T 1R1W register file in 14 nm FinFET technology
- Ghosh and Kulatilake, 1987, A FORTRAN program for generation of multivariate normally distributed random variables
- Ghuzdewan, 2025, Development of a Python-Based Program for Pareto Analysis in Construction Project Cost Management
- Giacobazzi, 2008, Abstract Interpretation in Code Security
- Gibbons, 2021, Continuation-Passing Style, Defunctionalization, Accumulations, and Associativity
- Gifford and others, 1992, Report on the FX-91 Programming Language
- Gillard and others, 2026, Dynamic Wind for OCaml Effect Handlers with Escaping Continuation Support
- Ginsbach and others, 2018, CAnDL a domain specific language for compiler analysis
- Giorgi and Le Métayer, 1990, Continuation-based parallel implementation of functional programming languages
- Girault, 2025, Toward a Formally Verified Compiler for a Synchronous, Functional, Data-Flow Programming Language
- Godboley and others, 2022, AV-AFL A Vulnerability Detection Fuzzing Approach by Proving Non-reachable Vulnerabilities using Sound Static Analyser
- Goldberg, 1998, A specification of Java loading and bytecode verification
- Gómez-Déniz and others, 2016, Random Tests Combining Mathematica Package and Latex Compiler
- Goncharov and others, 2018, Unguarded Recursion on Coinductive Resumptions
- Goodenough and others, 1976, Evaluation of ALGOL 68, JOVIAL J3B, PASCAL, SIMULA 67, and TACPOL vs. TINMAN Requirements for a Common High Order Programming Language
- Goos, 2002, Compiler Verification and Compiler Architecture

- Gordon and others, 2022, Porting the Kitten Lightweight Kernel Operating System to RISC-V
- Gordon and Scholz, 2015, Dynamic adaptation of functional runtime systems through external control
- Gore and others, 2023, Sentence Generator for English Language using Formal Semantics
- Gorelik and Khukhlaev, 1975, Implementation of the incremental fortran compiler
- Gorodetskiy, 2019, NEXTGEN PROGRAMMING LANGUAGE WITH PROGRAMMABLE SEMANTICS
- Gorringe and Jain, 2013, Architectural considerations for implementation of the ATML standards in an open systems architecture runtime environment OSA-RTS using a graphical environment
- Gotti and Mbarki, 2016, Java Swing Modernization Approach - Complete Abstract Representation based on Static and Dynamic Analysis
- Gourdin, 2023, Lazy Code Transformations in a Formally Verified Compiler
- Grabmayer, 2023, A Coinductive Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity
- Grabowski, 2025, Encoding Triangular Norms on Bounded Trellises with Mizar Proof Assistant
- Graham, 1995, Information Technology. Programming Language. The SQL Ada Module Description Language SAMeDL
- Grechanik, 2012, Random benchmark application generation for evaluating program analysis and testing tools
- 2023, Green Supply Chain Management and Competitive Advantage Evidence of Just-in-time Management on Firm Performance SMEs in Indonesia
- GREEN and WOOD, 2024, REASONING ABOUT THE ACOUSTIC REALISATION OF SEMIVOWELS USING AN INTERMEDIATE REPRESENTATION - THE 'SPEECH SKETCH'
- Groce and others, 2022, Making no-fuss compiler fuzzing effective
- Groenewegen and others, 2020, Evolution of the WebDSL runtime reliability engineering of the WebDSL web programming language
- Grolaux and others, 2026, Async/await is an Effective Paradigm for Event Management of User Interfaces
- Groß and others, 2023, FUZZILLI Fuzzing for JavaScript JIT Compiler Vulnerabilities
- Grover, 1985, Guidelines for a Minimal Ada Runtime Environment
- Gruetter and others, 2024, Live Verification in an Interactive Proof Assistant
- Gruner and others, 2025, Finding Information Leaks with Information Flow Fuzzing
- Gruner and others, 2025, Finding Information Leaks with Information Flow Fuzzing-RCR Report
- Gu, 2023, LLM-Based Code Generation Method for Golang Compiler Testing
- Guan and others, 2019, Wootz a compiler-based framework for fast CNN pruning via composability
- Guan and Treude, 2024, Enhancing Source Code Representations for Deep Learning with Static Analysis
- Guarna and Jr, 1987, VPC - A Proposal for a Vector Parallel C Programming Language
- Guerrera and others, 2019, Reproducible stencil compiler benchmarks using prova!
- Guerrini and Masini, 2009, Proofs, tests and continuation passing style
- Guessarian, 1979, Program transformations and algebraic semantics
- Gunadi, 2015, Formal Certification of Non-interferent Android Bytecode DEX Bytecode
- Gundersen and others, 2013, Atomic Lambda Calculus A Typed Lambda-Calculus with Explicit Sharing
- Guo and others, 2007, A Phase-Coupled Compiler Backend for a New VLIW Processor Architecture Using Two-step Register Allocation
- Guo and Si, 2016, Mechanical hydraulic characteristic analysis scheme based on

lightweight crowd data in mobile embedded devices
- Gupta, 1990, An Incremental Type Inference System for the Programming Language Id
- Gupta and Lewis, 2018, Neural Compositional Denotational Semantics for Question Answering
- Guria and others, 2021, RbSyn type- and effect-guided program synthesis
- Gustafsson, 2006, The Worst Case Execution Time Tool Challenge 2006
- Guyer and Lin, 2005, Broadway A Compiler for Exploiting the Domain-Specific Semantics of Software Libraries
- G. Western and Ball, 1991, 3D Pre-stack depth migration in the Gulf of Sueza. A case history
- Haas and others, 2023, Static Analysis of Memory Models for SMT Encodings
- Haas and others, 2017, Bringing the web up to speed with WebAssembly
- Haase, 2016, Abstract Interpretation of Java Bytecode for Immutability Analysis
- Haikun Liu and others, 2011, Live Virtual Machine Migration via Asynchronous Replication and State Synchronization
- Haj-Yihia and others, 2015, Compiler-Directed Power Management for Superscalars
- Hale and others, 2018, Interpreter performance in police interviews. Differences between trained interpreters and untrained bilinguals
- Hamana, 2022, Modular Termination for Second-Order Computation Rules and Application to Algebraic Effect Handlers
- HAMID and ITO, 2017, Image Segmentation to Design Semi-optimized Curve for B-code Generation
- Hammer and others, 2025, Compiler-Like Code Generation for fUML Reducing Overhead in Executable UML
- Hammond and others, 2023, MPI Application Binary Interface Standardization
- Han, 2016, Building the validity foundation for interpreter certification performance testing
- Han and others, 2024, Range Specification Bug Detection in Flight Control System Through Fuzzing
- Han and others, 2021, Design and Implementation of a Criticality- and Heterogeneity-Aware Runtime System for Task-Parallel Applications
- Han and others, 2026, Design and Implementation of Boss AI for 2D Action Games Using Finite State Machines and Coroutine Chaining
- Han and others, 2021, Parallelizing Compiler Translation Validation Using Happens-Before and Task-Set
- Han and others, 2024, Enabling Fine-Grained Incremental Builds by Making Compiler Stateful
- Hanitzsch and others, 2001, Pre-Stack Depth Migration for Time-to-Depth Conversion - the Ultimate Tool?
- Hanley and Lippman-Hand 1983, If nothing goes wrong, is everything all right?
- Hansen and Siveroni, 2005, Towards Verification of Well-Formed Transactions in Java Card Bytecode
- Hanson, 1976, A simple variant of the boundary-tag algorithm for the allocation of coroutine environments
- Hanzich and others, 2009, Subsalt Imaging through Pre-Stack Depth Migration - A Case Study from the North Red Sea
- Hara and others, 2021, Machine-learning Approach using Solidity Bytecode for Smart-contract Honeypot Detection in the Ethereum
- Hardy and Puaut, 2013, Static probabilistic worst case execution time estimation for architectures with faulty instruction caches
- Hardy and Puaut, 2014, Static probabilistic worst case execution time estimation for architectures with faulty instruction caches
- Hariri and others, 2016, Evaluating the Effects of Compiler Optimizations on Mutation Testing at the Compiler IR Level

- Hariri and others, 2019, Comparing Mutation Testing at the Levels of Source Code and Compiler Intermediate Representation
- Harlin and others, 2017, Impact of Using a Static-Type System in Computer Programming
- Harmon, 1988, An Ada implementation of Marsaglia's universal random number generator
- Harmon and Klefstad, 2007, A Survey of Worst-Case Execution Time Analysis for Real-Time Java
- Harmon and Klefstad, 2007, Interactive Back-annotation of Worst-case Execution Time Analysis for Java Microprocessors
- Harmon and Klefstad, 2007, Toward a Unified Standard for Worst-Case Execution Time Annotations in Real-Time Java
- Harmon and others, 2008, A Modular Worst-case Execution Time Analysis Tool for Java Processors
- Harmon and others, 2012, Fast, Interactive Worst-Case Execution Time Analysis With Back-Annotation
- Harnes and Morrison, 2024, SoK Analysis Techniques for WebAssembly
- Harper and Pfenning, 1992, A Module System for a Programming Language Based on the LF Logical Framework
- Harrison, 1989, Research, Development, Training and Education Using the Ada Programming Language
- Harrison and others, 1976, Theoretical results in compiler design and implementation Tutorial Session
- Harshithan, 2023, Batwing Compiler An Artificial Intelligence based Compiler
- Hart and McClanahan, 1981, JOVIAL J73 Compiler Validator
- Hartley and others, 2022, Just-In-Time Compilation on ARM-A Closer Look at Call-Site Code Consistency
- Haspert and Beauregard, 1998, The Commercialization of a Rapid Prototyping Development Tool for Real-Time Embedded Software Intensive, Process, and Resource Management Systems
- HATABA and others, 2019, Generation of Efficient Obfuscated Code through Just-in-Time Compilation
- Hatcliff and Danvy, 1994, A generic account of continuation-passing styles
- Hausladen and others, 2017, Integration of Static Worst-Case Execution Time and Stack Usage Analysis for Embedded Systems Software in a Cloud-Based Development Environment
- Havelund and others, 2000, Formal Analysis of the Remote Agent Before and After Flight
- Hazimeh and others, 2021, Magma A Ground-Truth Fuzzing Benchmark
- Hazott and others, 2025, Using virtual prototypes and metamorphic testing to verify the hardware/software-stack of embedded graphics libraries
- He and others, 2023, Neural-FEBI Accurate function identification in Ethereum Virtual Machine bytecode
- He and others, 2025, Evaluating Program Semantics Reasoning with Type Inference in System F
- He and others, 2023, Profile Guided Optimization Transfer-Learning for OpenCL/SYCL Kernel Compilation and Runtime
- He and Zhong, 2025, From Bug Reports to Workarounds The Real-World Impact of Compiler Bugs
- Heck and Zaidman, 2015, Quality criteria for just-in-time requirements just enough, just-in-time?
- Heim, 2018, Compiler and language design for quantum computing keynote
- Heimbigner, 2006, A Tamper-Resistant Programming Language System
- Heinrich and others, 2024, A Categorical Data Approach for Anomaly Detection in WebAssembly Applications

- Henriksen and Gallagher, 2006, Abstract Interpretation of PIC Programs through Logic Programming
- Henry and others, 2014, How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics
- Hensley and Elgazzar, 2025, DESIGN AND IMPLEMENTATION OF THE MOREHEAD-AZALEA COMPILER MAC
- Heo and others, 2025, Bit-level compiler optimization for ultra low-power embedded systems
- Heo and others, 2019, Resource-Aware Program Analysis Via Online Abstraction Coarsening
- Hepp and Schoeberl, 2012, Worst-Case Execution Time Based Optimization of Real-Time Java Programs
- Herklotz and others, 2023, Mechanised Semantics for Gated Static Single Assignment
- Herring and others, 1993, Research in Presistent Simulation Development of the Persistent ModSim Object-Oriented Programming Language
- Heumann and others, 2015, Scalable Task Scheduling and Synchronization Using Hierarchical Effects
- Heuring and others, 1990, Automatic Compiler Construction
- Hikmatyarsyah and Rahardjo, 2022, Integration of Downlink Scheme VLC Access Techniques for Low-cost Implementation Indoor Communication System A Survey
- Liberating effects with rows and handlers
- HILLERSTRÖM and others, 2020, Effect handlers via generalised continuations
- HILLERSTRÖM and others, 2024, Asymptotic speedup via effect handlers
- Hinkley and others, 1994, Velocity Model Building for 3D Pre-Stack Depth Migration - a Case Study
- Hirata and others, 2023, Program logic for higher-order probabilistic programs in Isabelle/HOL
- Hird, 1991, Formal specification and verification of Ada software
- Hiroyuki, 2009, Idiom Recognition and Program Scheme Recognition Based Program Transformations for Performance Tuning–Beyond Compiler Optimizations
- Hirschowitz, 2019, Familial monads and structural operational semantics
- Hmid and others, 2016, A Transfer-Aware Runtime System for Heterogeneous Asynchronous Parallel Execution
- Hoang and Rabaey, 1992, A compiler for multiprocessor DSP implementation
- Hollingshaus and Daddario, 2015, Performance Philosophy Arrived Just in Time?
- Holmen and others, 2021, A Heterogeneous MPI+PPL Task Scheduling Approach for Asynchronous Many-Task Runtime Systems
- Holmes and Groce, 2018, Causal Distance-Metric-Based Assistance for Debugging after Compiler Fuzzing
- Holmes and Groce, 2020, Using mutants to help developers distinguish and debug compiler faults
- Hong and Ramanujam, 2008, Address Register Allocation in Digital Signal Processors
- Hook, 1987, Export Control of the Ada Trade Name Compiler Validation Capability ACVC
- Hook and Heilbrunner, 1989, Ada Compiler Validation Procedures
- Hook and Lehman, 1992, Ada Compiler Validation Support Fiscal Year 1992
- Horký and others, 2016, Analysis of Overhead in Dynamic Java Performance Monitoring
- Horvat and others, 2026, Comparative Evaluation of Gemini and DeepSeek for LLM-Generated Code Quality and Architectural Robustness in Backend Software Engineering
- Horwat, 1988, A Concurrent Smalltalk Compiler for the Message-Driven Processor
- Hoskins, 1988, The design and implementation of a Karel compiler and interpreter
- Hossain and others, 2021, Code Generator based on Voice Command for Multiple Programming Language
- Howe, 1984, A Study of the Feasibility of Duplicating JAMPS Applications Software in the Ada Programming Language

- Hsu and Kremer, 2003, The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction
- Hu and others, 2025, SSFuzz Synthesizing and scheduling bug-triggering code segments for history-driven compiler testing
- Hu and others, 2026, QJWasm A lightweight runtime system for efficient WebAssembly execution in resource-constrained environments
- Hu and others, 2024, An Abstract Interpretation-Based Framework for WCET Analysis of Parallel Programs
- Hu and Tang, 2024, Research on compiler version recognition based on random forest algorithm
- Hu and others, 2022, Research on global register allocation for code containing array-unit dual-usage register names
- Hu and Zhao, 2014, Analysis on Process Code schedule of Android Dalvik Virtual Machine
- Hua and others, 2010, Model-Based Intrusion Detection by Abstract Interpretation
- Huang and others, 2022, A High-Performance Bidirectional Compiler for Conversion Between SystemC and Verilog
- Huang and Huang, 2016, Cell-based delay locked loop compiler
- Huang and others, 2006, Adaptiveness in well-typed Java bytecode verification
- Huangfu and Zhang, 2018, Estimating the Worst-Case Execution Time of the Shared Data Cache in Integrated CPU-GPU Architectures
- Huber and others, 2011, Worst-case execution time analysis-driven object cache design
- Huch, 1999, Verification of Erlang programs using abstract interpretation and model checking
- Huck and others, 2022, Compiler-Aided Type Correctness of Hybrid MPI-OpenMP Applications
- Huck and others, 2018, Compiler-aided Type Tracking for Correctness Checking of MPI Applications
- Huck and others, 2020, Towards compiler-aided correctness checking of adjoint MPI applications
- Hück and others, 2024, Compiler-Aided Correctness Checking of CUDA-Aware MPI Applications
- Hui Zhao and others, 2015, Virtual machine placement based on the VM performance models in cloud
- Hunt and others, 2008, Using global data flow analysis on bytecode to aid worst case execution time analysis for real-time Java programs
- Hura, 1982, Optimization of assembly code generation in a compiler
- Hussain and others, 2014, RUGRAT Evaluating program analysis and testing tools and compilers with large generated random benchmark applications
- Huynh and Roychoudhury, 2007, Memory model sensitive bytecode verification
- Huynh and Taura, 2017, Delay Spotter A Tool for Spotting Scheduler-Caused Delays in Task Parallel Runtime Systems
- Hyatt and Dewey, 2025, Mutation-Based Fuzzing of the Swift Compiler with Incomplete Type Information
- Hyland and others, 2007, Combining algebraic effects with continuations
- Hyvernat, 2025, The Size-Change Principle for Mixed Inductive and Coinductive types
- Hyvernat, 2025, Totality for Mixed Inductive and Coinductive Types
- Iida and others, 2026, An Efficient Runtime Verification Toolkit for Self-Adaptive Systems Addressing Runtime System Model Changes
- Ikemori and others, 2023, Typed Equivalence of Labeled Effect Handlers and Labeled Delimited Control Operators
- Ilik, 2012, Delimited control operators prove Double-negation Shift
- Ilik, 2013, Continuation-passing style models complete for intuitionistic logic
- Ilik, 2013, Type Directed Partial Evaluation for Level-1 Shift and Reset
- Ilik, 2014, Proofs in continuation-passing style

- Imamoglu and Cetinkaya, 2017, A rule based decision support system for programming language selection
- 2024, Implementation concept of the IoT platform using C++ programming language
- 1973, Implementation of a Pascal compiler for the CII IRIS 80 computer
- 2015, Implementation of a Motor Imagery based BCI System using Python Programming Language
- 2015, Incorporating Near-Surface Velocity Anomalies in Pre-Stack Depth Migration Models
- INFORMATION MANAGEMENT INC SAN FRANCISCO CA, 1970, USER'S MANUAL COBOL COMPILER VALIDATION SYSTEM
- Inoue and others, 2011, A trace-based Java JIT compiler retrofitted from a method-based compiler
- Inoue and Igarashi, 2019, A type system for first-class layers with inheritance, subtyping, and swapping
- Inoue and Kaneko, 2015, Bitwidth-aware register allocation and binding for clock period minimization
- I.Saetchnikov and others, 2026, Microresonator clusters for spectral analysis with machine-learning interpreter
- Ishio and Asai, 2022, Type System for Four Delimited Control Operators
- ISHIURA, 2016, Compiler Fuzzing
- Iskra and Hoefler, 2015, Operating systems and runtime environments on supercomputers
- ISLAM and others, 2022, An Exploration of npm Package Co-Usage Examples from Stack Overflow A Case Study
- Ismael and others, 2018, System on Chip Implementation of Compiler Stack with a Delimiter Matching Application
- Isoda and others, 2024, Type-Safe Code Generation with Algebraic Effects and Handlers
- Israel and others, 2026, Exploring WebAssembly as a Runtime Platform for Safety-Critical Edge Systems
- Ito and others, 2023, Schfuzz Detecting Concurrency Bugs with Feedback-Guided Fuzzing
- Ivanov and others, 2019, Software Structure, Program Generation and Schedulability Analysis of Extracorporeal Perfusion Pump Embedded Controller
- Ivey and Riley, 2016, Analysis of Programming Language Overhead in DCE
- Iwasaki and others, 2018, Lessons Learned from Analyzing Dynamic Promotion for User-Level Threading
- Iyenghar and others, 2026, Incremental Static Analysis for Detecting and Refactoring Data Clumps in TypeScript
- Izawa and others, 2022, Threaded Code Generation with a Meta-Tracing JIT Compiler
- Jaafar and Jaber, 2025, Operational Game Semantics for Generative Algebraic Effects and Handlers
- Jacek and others, 2016, Assessing the limits of program-specific garbage collection performance
- Jackson, 2026, I/O Optimisation at the Compiler Level IOOpt
- Jadhav and others, 2026, Analysis of Compiler-Level Static and Dynamic Features for Automated Bug Prediction Using Transformer Models
- Jadhav and Falk, 2025, Compiler-level DMA-aware multi-objective dynamic SPM allocation
- Jagnik, 2023, Structured Concurrency in Java
- Jain and others, 2022, Coarse Grained FPGA Overlay for Rapid Just-In-Time Accelerator Compilation
- Jamieson and Brown, 2021, Compact native code generation for dynamic languages on micro-core architectures
- Jamil, 2018, Design of a Real-Time Interpreter for Arabic Sign Language

- JANA, 2023, Sensitive Information Leakage Analysis of Database Code by Abstract Interpretation
- Janetschek and Prodan, 2017, A compiler transformation-based approach to scientific workflow enactment
- Janssens and Bruynooghe, 1992, Deriving descriptions of possible values of program variables by means of abstract interpretation
- Jay and Miller, 2018, Structured random differential testing of instruction decoders
- J. Berkhout, 1992, True amplitude aspects of pre-stack depth migration
- Jefferson and others, 1994, Ada Compiler Validation Summary Report Certificate Number 940902S1.11376. UNISYS Corporation IntegrAda for Windows NT, Version 1.0. Intel Deskside Server for Intel Pentium 60 MHz = . Intel Deskside Server with Intel Pentium 60 MHz
- Jefferson and others, 1994, Ada Compiler Validation Summary Report Certificate Number 940902S1.11377 UNISYS Corporation. IntegrAda for Windows NT, Version 1.0. Intel Deskside Server with Intel 80486DX266 = Intel Deskside Server with Intel 80486DX266
- Jeon and others, 2021, A practical algorithm for learning disjunctive abstraction heuristics in static program analysis
- JEONG and others, 2021, Usage Log-Based Testing of Embedded Software and Identification of Dependencies among Environmental Components
- Jeong and others, 2019, Razzer Finding Kernel Race Bugs through Fuzzing
- Jeong and others, 2024, Conflict-aware compiler for hierarchical register file on GPUs
- Jervis, 1963, MOBILE. A MOBIDIC COBOL COMPILER
- Jesus and Weiland, 2024, Evaluating and optimising compiler code generation for NVIDIA Grace
- Jha and others, 2018, Worst Case Execution Time Estimation for Control Code of Automation Systems
- Ji and others, 2007, Automated Worst-Case Execution Time Analysis Based on Program Modes
- Ji and Wang, 2022, Optimizing Aggregate Computation of Graph Neural Networks with on-GPU Interpreter-Style Programming
- Jiang and others, 2024, DCIM Compiler - Physical Design Generator
- Jiang and Li, 2014, Using Contour Marking Bytecode Verification Algorithm on the Java Card
- Jiang and Yan, 2010, Implementation of Static Web-Pages Generator Using JavaScript
- Jiang and others, 2025, Distinguishability-Guided Test Program Generation for WebAssembly Runtime Performance Testing
- Jiang and others, 2010, A Debugging Approach for Java Runtime Exceptions Based on Program Slicing and Stack Traces
- Jiang and others, 2015, Implementation and Comparison of the Way That Office Document Is Converted to PDF Documents in the Java Runtime Environment
- Jimenez Gil and others, 2017, Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time
- Jodogne, 2022, Rendering Medical Images using WebAssembly
- Johann and others, 2010, A Generic Operational Metatheory for Algebraic Effects
- Johnson, 1978, Tools For Automatic Compiler Generation Panel Discussion
- Johnson and others, 2024, Automating Pruning in Top-Down Enumeration for Program Synthesis Problems with Monotonic Semantics
- Johnson and others, 2023, WaVe a verifiably secure WebAssembly sandboxing runtime
- Jones, 1980, Compiler Generation from Denotational Semantics
- Jones and Christiansen, 1981, Control Flow Treatment in a Simple Semantics-Directed Compiler Generator
- Jong-In Lee and others, 2005, A Hybrid Framework of Worst-Case Execution Time Analysis for Real-Time Embedded System Software
- 2010, JSIMIL - A Java Bytecode Clone Detector

- Jui-Ming Chang, 1995, Register Allocation and Binding for Low Power
- Julián-Iranzo and Rubio-Manzano, 2017, A sound and complete semantics for a similarity-based logic programming language
- Jung, 2021, CommitBERT Commit Message Generation Using Pre-Trained Programming Language Model
- Jung, 2024, Miri Practical Undefined Behavior Detection for Rust Keynote
- ., 2018, Just in time and competitive advantage understanding their linkages and impact on operational performance
- Jyh-Charn Liu and Hung-Ju Lee, 1994, Deterministic upperbounds of the worst-case execution times of cached programs
- Kaestner, 2007, Safe worst-case execution time analysis by abstract interpretation of executable code
- Kaestner and others, 2025, Determining Worst-Case Execution Time Bounds for Multi-Core Processors
- Kahn and others, 2026, Big-Stop Semantics Small-Step Semantics in a Big-Step Judgment
- Kakati and Brorsson, 2025, Performance and Usability Implications of Multiplatform and WebAssembly Containers
- Kalebe and others, 2017, A library for scheduling lightweight threads in Internet of Things microcontrollers
- Kalyur and Nagaraja, 2016, A survey of modeling techniques used in compiler design and implementation
- Kameyama and Tanaka, 2010, Equational axiomatization of call-by-name delimited control
- KAMMAR and PRETNAR, 2017, No value restriction is needed for algebraic effects and handlers
- Kanabar and others, 2023, PureCake A Verified Compiler for a Lazy Functional Language
- Kanatov and Zouev, 2022, Unified type system for the modern general-purpose programming language
- Kandemir, 2001, A compiler technique for improving whole-program locality
- Kang and others, 2026, WAMI Compilation to WebAssembly through MLIR without Losing Abstraction
- Kang and others, 2025, HybridServe Adaptive WebAssembly-Container Runtime Selection for Edge Serverless Computing
- Karachalias and others, 2021, Efficient compilation of algebraic effect handlers
- Kargén and Shahmehri, 2018, Speeding Up Bug Finding using Focused Fuzzing
- Karpovich and Gosudarev, 2025, WebAssembly performance in the Node.js environment
- Karr, 1984, Code generation by coagulation
- Karsten and Barghi, 2020, User-level Threading
- Kasampalis and others, 2021, Language-parametric compiler validation with application to LLVM
- Kasaraneni and Nandivada, 2026, Compact Representation and Interleaved Solving for Scalable Constraint-Based Points-to Analysis
- Katel and others, 2022, MLIR-based code generation for GPU tensor cores
- Kawamata and others, 2024, Answer Refinement Modification Refinement Type System for Algebraic Effects and Handlers
- Kaynaroglu and others, 2025, EUTROPY A Python-based software optimized with Just-In-Time compilation for simulating eutrophication dynamics in aquatic systems
- Ke and Chen, 2020, Instruction Verification of Ethereum Virtual Machine by Formal Method
- Kearns and Lou Soffa, 1983, The implementation of retention in a coroutine environment
- Keaton and Seacord, 2014, Performance of Compiler-Assisted Memory Safety Checking
- Keidel and others, 2018, Compositional soundness proofs of abstract interpreters
- KELLY, 1963, ADVANCED MYSTIC- A COMPILER FOR MANAGEMENT CONTROL OF

- Kelly, 1997, Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems Volume II A Practitioner's Companion
- Kelsey, 1995, A correspondence between continuation passing style and static single assignment form
- Kennedy and Syme, 2001, Design and implementation of generics for the .NET Common language runtime
- Kerneis and Chroboczek, 2011, Continuation-Passing C, compiling threads to events through continuations
- Keutzer and Wolf, 1988, Anatomy of a hardware compiler
- Khaldi and Chapman, 2016, Towards Automatic HBM Allocation Using LLVM A Case Study with Knights Landing
- Khaldi and others, 2015, LLVM parallel intermediate representation
- Khatami and others, 2017, Redesigning OP2 Compiler to Use HPX Runtime Asynchronous Techniques
- Khorasani and others, 2015, Scalable SIMD-Efficient Graph Processing on GPUs
- KIAM TAN and others, 2019, The verified CakeML compiler backend
- Kidney and Wu, 2025, Formalising Graph Algorithms with Coinduction
- Kiefer and others, 2017, Relational Program Reasoning Using Compiler IR
- Kim, 2015, Exploiting Window Query Semantics in Scalable Data Stream Processing
- KIM and others, 2000, REGISTER ALLOCATION IN HYPER-BLOCK FOR EPIC PROCESSORS
- Kim and Hong, 2023, Poster BugOss A Regression Bug Benchmark for Empirical Study of Regression Fuzzing Techniques
- Kim and others, 2019, WCET-Aware Stack Frame Management of Embedded Systems Using Scratchpad Memories
- Kim and others, 2012, Generating Verification Conditions from BIRS Code using Basic Paths for Java Bytecode Verification
- Kim and others, 2025, Pre-trained Models for Bytecode Instructions
- Kim and Lee, 2017, A Study on the Light-Weight Virtual Machine Code for IoT Virtual Machine
- Kim and Lee, 2018, A Study on the Code Generator for a Virtual Machine Code based JavaScript Compiler
- Kim and Park, 2023, A Multi-core Based Real-time Scheduler Supporting Periodic and Sporadic Threads and Processes
- Kim and Ryou, 2019, Source Code Analysis for Static Prediction of Dynamic Memory Usage
- Kim and others, 2025, Fuzzing Acceleration for Memory Safety Bug Discovery with Slicer
- Kim and others, 2025, Chimera Fuzzing P4 Network Infrastructure for Multi-Plane Bug Detection and Vulnerability Discovery
- Kim and others, 2020, Finding Bugs in File Systems with an Extensible Fuzzing Framework
- Kim and others, 2020, Compiler-directed soft error resilience for lightweight GPU register file protection
- Kingsley, 1987, The implementation of a state machine compiler
- Kipps, 1982, Experience with porting techniques on a COBOL 74 compiler
- Kirichenko and Tarasov, 2017, Development of design flow for multiported register files, which includes a cell library and a compiler for SOI 0.25-µm process
- Kirner and others, 2009, Precise Worst-Case Execution Time Analysis for Processors with Timing Anomalies
- Kirner and others, 2010, Beyond loop bounds comparing annotation languages for worst-case execution time analysis
- Kirner and Puschner, 2008, Obstacles in Worst-Case Execution Time Analysis
- Kirner and Schoeberl, 2007, Modeling the Function Cache for Worst-Case Execution

Time Analysis
- Kiselyov, 2012, Delimited control in OCaml, abstractly and concretely
- Kishorbhai and Patel, 2026, Online Code Compiler A Modern Web Based Programming Platform
- Kistel and Vandenhouten, 2013, A metamodel-based ASN.1 editor and compiler for the implementation of communication protocols
- Klein, 2005, Verified Java Bytecode Verification Verified Java Bytecode Verification
- Klein and Nipkow, 2001, Verified lightweight bytecode verification
- Klein and Strecker, 2004, Verified bytecode verification and type-certifying compilation
- Kleinsorge and others, 2013, Simple analysis of partial worst-case execution paths on general control flow graphs
- Klimis, 2026, Compilomorphic Fuzzing Turning a Compiler Against Itself
- Klöckner and others, 2016, Array program transformation with Loo.py by example high-order finite elements
- Klohs and Kastens, 2005, Memory Requirements of Java Bytecode Verification on Limited Devices
- Knoblock and Rehof, 2000, Type elaboration and subtype completion for Java bytecode
- Knoblock and Rehof, 2001, Type elaboration and subtype completion for Java bytecode
- Knoop and others, 2017, Replacing conjectures by positive knowledge Inferring proven precise worst-case execution time bounds using symbolic execution
- Ko and others, 2015, LaminarIR compile-time queues for structured streams
- Ko and Heo, 2026, TinyGen Portable and Compact Code Generation for Tiny Machine Learning
- Kobayashi and others, 2017, On the relationship between higher-order recursion schemes and higher-order fixpoint logic
- Kobusińska and Wilczynski, 2022, Blocked-based Solidity a Service for Graphically Creating the Smart Contracts in Solidity Programming Language
- Kocourek and others, 2025, Copy-and-Patch Just-in-Time Compiler for R
- Koehler and Steuwer, 2021, Towards a Domain-Extensible Compiler Optimizing an Image Processing Pipeline on Mobile CPUs
- Kokkonis and others, 2025, ROSA Finding Backdoors with Fuzzing
- Kolek and others, 2013, Adding microMIPS backend to the LLVM compiler infrastructure
- Kolesar and others, 2025, Coinductive Proofs of Regular Expression Equivalence in Zero Knowledge
- Komendantskaya and Li, 2018, Towards Coinductive Theory Exploration in Horn Clause Logic Position Paper
- Komolov and others, 2020, An empirical study of multi-threading paradigms Reactive programming vs continuation-passing style
- Kondratyev and Promsky, 2019, Correctness of Proof Strategy for the Sisal Program Verification
- Kong and Chen, 2013, A Worst-Case Execution Time Analysis Approach Based on AOE Networks
- Kong and Jiang, 2012, A Worst-case execution time analysis approach based on independent paths for ARM programs
- Kong and others, 2014, An Overview of Worst-Case Execution Time Estimation for Embedded Programs
- Koroglu and Wotawa, 2019, Fully Automated Compiler Testing of a Reasoning Engine via Mutated Grammar Fuzzing
- Koskimies and others, 1982, Compiler construction using attribute grammars
- Kossatchev and Posypkin, 2005, Survey of compiler testing methods
- Kot and Kozen, 2005, Kleene Algebra and Bytecode Verification
- Kovačević and others, 2022, Automatic compiler/interpreter generation from programs for Domain-Specific Languages Code bloat problem and performance improvement
- Kowalewski and others, 2013, Model checking and abstract interpretation as building

blocks of advanced program analysis techniques
- KOZEN and SILVA, 2016, Practical coinduction
- Krause, 2015, Bytewise Register Allocation
- Krebs and Schmitz, 2014, Jaccie A Java-based compiler-compiler for generating, visualizing and debugging compiler components
- Krishnamoorthy, 2025, WEBASSEMBLY REVOLUTIONIZING WEB PERFORMANCE AND EXPANDING FRONTIERS OF BROWSER-BASED APPLICATIONS
- Kristensen and others, 1987, Coroutine Sequencing in BETA
- Kroening and others, 2016, Sound static deadlock analysis for C/Pthreads
- Krogstie and others, 2026, PIP Making Andersen's Points-to Analysis Sound and Practical for Incomplete C Programs
- Krzemień and Lukasiewicz, 1976, Automatic generation of lexical analyzers in a compiler-compiler
- Kuang and others, 2018, Enhance virtual-machine-based code obfuscation security through dynamic bytecode scheduling
- CakeML: a verified implementation of ML
- Kumar, 2021, Deep Neural Network Approach to Estimate Early Worst-Case Execution Time
- Kumar and others, 2024, Utilizing Machine Learning Techniques for Worst-Case Execution Time Estimation on GPU Architectures
- Kuperberg and others, 2021, Coinductive Algorithms for Büchi Automata
- Kupke and Rot, 2021, Expressive Logics for Coinductive Predicates
- Kuroda and Yuen, 2026, A Concurrent Extension of Reversible Imperative Programming Language with Runtime
- Kusano and Wang, 2017, Thread-modular static analysis for relaxed memory models
- Kusmenko and others, 2018, Highly-Optimizing and Multi-Target Compiler for Embedded System Models
- Kwon and Bae, 2016, Development of a Code Generation Support System in Integrated Development Environment of an Educational Compiler
- Kwon and others, 2025, Optimization-Directed Compiler Fuzzing for Continuous Translation Validation
- Kwon and others, 2024, Translation Validation for JIT Compiler in the V8 JavaScript Engine
- Kwon and others, 2026, Compiler-Runtime Co-operative Chain of Verification for LLM-Based Code Optimization
- Kyriakou and Tselikas, 2022, Complementing JavaScript in High-Performance Node.js and Web Applications with Rust and WebAssembly
- Kyrtatas and others, 2015, A Basic Linear Algebra Compiler for Embedded Processors
- Lai and others, 2026, Interaction-aware multi-objective optimization method for LLVM compiler option sequences
- Laliotis, 1973, Implementation aspects of the symbol hardware compiler
- Lambert and others, 2022, Leveraging Compiler-Based Translation to Evaluate a Diversity of Exascale Platforms
- Lambert and Saunders, 2017, Compiler auto-vectorization of matrix multiplication modulo small primes
- Lancellotti and others, 2026, Quantum Oracle Synthesis from HDL Designs via Multi Level Intermediate Representation
- Landa and Sorin, 1993, Fast pre-stack depth migration by CRP stacking
- Lane and Poorman, 1991, Preserving software investment using new fortran compiler technology
- Lange and others, 1977, Specification for a STARAN Programming Language
- Larkins and Jones, 2011, Targeting FPGA-based processors for an implementation-driven compiler construction course
- Larus, 2011, Session details Compiler correctness

- Larus and Hilfinger, 1986, Register allocation in the SPUR Lisp compiler
- Lasnier and others, 2026, Brack A Verified Compiler for Scheme via CakeML
- Lawall and Danvy, 1993, Separating stages in the continuation-passing style transformation
- Compiler validation via equivalence modulo inputs
- Le and others, 2015, Finding deep compiler bugs via guided stochastic program mutation
- League, 2002, A Type-Preserving Compiler Infrastructure
- Leavitt and Terrell, 1991, Ada Compiler Evaluation Capability
- Leavitt and Terrell, 1991, ADA Compiler Evaluation Capability User's Guide, Release 2.0
- Leavitt and Terrell, 1991, Ada Compiler Evaluation Capability. Release 2.0
- Le Charlier and Van Hentenryck, 1994, Experimental evaluation of a generic abstract interpretation algorithm for PROLOG
- Lee, 2017, Exploring a relationship between students' interpreting self-efficacy and performance triangulating data on interpreter performance assessment
- Lee and others, 2025, React-tRace A Semantics for Understanding React Hooks An Operational Semantics and a Visualizer for Clarifying React Hooks
- LEE and others, 2009, Visualization and Formalization of User Constraints for Tight Estimation of Worst-Case Execution Time
- Lee and others, 2025, Bit-Level Semantics Scalable RAG Retrieval with Neurosymbolic Hyperdimensional Computing
- Lee and others, 2017, Design and implementation of the secure compiler and virtual machine for developing secure IoT services
- Lee and Lee, 2024, IMC-PnG Maximizing runtime performance and timing guarantee for imprecise mixed-criticality real-time scheduling
- Lee and others, 2015, Flow-sensitive runtime estimation an enhanced hot spot detection heuristics for embedded Java just-in-time compilers
- Lee and Pleban, 1987, A realistic compiler generator based on high-level semantics another progress report
- Lehman, 1992, Ada Compiler Validation Support Fiscal Year 1991
- Lehman and others, 1988, Sources of Compiler Capability Information in Validation Summary Reports
- Lehmann and others, 2025, Hardware/Software Co-Analysis for Worst Case Execution Time Bounds
- Lehmann and Pradel, 2022, Finding the Dwarf Recovering Precise Types from WebAssembly Binaries
- Lei and others, 2025, Furina A Light-weight WebAssembly Runtime for ICS
- Type directed compilation of row-typed algebraic effects
- Leijen, 2017, Structured asynchrony with algebraic effects
- Leijen, 2018, First class dynamic effect handlers or, polymorphic heaps with dynamic effect handlers
- Leinenbach and Petrova, 2008, Pervasive Compiler Verification From Verified Programs to Verified Systems
- Lemaistre and others, 2001, Automatic and Continuous Image Gather Analysis after Pre-Stack Depth Migration
- Lenkefi and Mezei, 2022, Connections between Language Semantics and the Query-based Compiler Architecture
- Leopoldseder and others, 2015, Java-to-JavaScript translation via structured control flow reconstruction of compiler IR
- Leopoldseder and others, 2018, Dominance-based duplication simulation DBDS code duplication to enable compiler optimizations
- Leopoldseder and others, 2018, A cost model for a graph-based intermediate-representation in a dynamic compiler
- Leroy, 2002, Bytecode verification on Java smart cards
- Leroy, 2003, Java Bytecode Verification Algorithms and Formalizations

- Leroy, 2006, Formal certification of a compiler back-end or
- Formal verification of a realistic compiler
- A formally verified compiler back-end
- Coinductive big-step operational semantics
- Leuschel, 2004, A framework for the integration of partial evaluation and abstract interpretation of logic programs
- Leverett and others, 1979, An Overview of the Production Quality Compiler-Compiler Project
- Levy, 1996, Should caches be split or shared? Analysis using the superposition of bursty stack depth processes
- Lezuo and others, 2015, vanHelsing A Fast Proof Checker for Debuggable Compiler Verification
- L. Freitas da Luz and C. Ribeiro Cruz, 2003, The CRS stack as a tool for pre-stack depth migration
- LI, 2008, Program Verification Techniques Based on the Abstract Interpretation Theory
- Li, 2019, Haskell Compiler Testing Automation Based on Equivalence-Modulo-Inputs Method
- Li and others, 2026, Unleashing Triton on CPUs Compilation and Runtime Co-Optimization for Scalable Vector Architectures
- Li and others, 2024, Simulink Compiler Testing via Configuration Diversification With Reinforcement Learning
- Li and others, 2016, Modular SDN Compiler Design with Intermediate Representation
- Li and others, 2022, ALPHAPROG Reinforcement Generation of Valid Programs for Compiler Fuzzing
- Li and others, 2025, Solsmith Solidity Random Program Generator for Compiler Testing
- Li and others, 2022, Unleashing the power of compiler intermediate representation to enhance neural program embeddings
- Li and others, 2017, Promotion of Educational Effectiveness by Translation-based Programming Language Learning Using Java and Swift
- Li and Su, 2023, Finding Unstable Code via Compiler-Driven Differential Testing
- Li and others, 2024, GastCoCo Graph Storage and Coroutine-Based Prefetch Co-Design for Dynamic Graph Processing
- Li and others, 2025, Lightweight and Holistic-Scalable Serverless Secure Container Runtime for High-Density Deployment and High-Concurrency Startup
- Li and others, 2023, A unified proof technique for verifying program correctness with big-step semantics
- Li and others, 2015, Compiler directed automatic stack trimming for efficient nonvolatile processors
- Liang and others, 2017, Improving the precision of static analysis Symbolic execution based on GCC abstract syntax tree
- Liangliang and Yungui, 1990, Clause representations in a compiler-based prolog database
- Lidbury and others, 2015, Many-core compiler fuzzing
- Lidman and Svenningsson, 2017, Bridging Static and Dynamic Program Analysis using Fuzzy Logic
- Lim and Debray, 2023, Automatically Localizing Dynamic Code Generation Bugs in JIT Compiler Back-End
- Lima and others, 2019, A memory-bounded, deterministic and terminating semantics for the synchronous programming language Céu
- Lin and others, 2016, Rust as a language for high performance GC implementation
- Lin and others, 2025, Intermediate Representation-Based Approach for Code Refactoring and Quality Evaluation
- Lin and Padua, 2000, Compiler analysis of irregular memory accesses
- Lin and others, 2023, DeepDiffer Find Deep Learning Compiler Bugs via Priority-guided

Differential Fuzzing
- Lindley, 2014, Algebraic effects and effect handlers for idioms and arrows
- Lins and others, 2017, Register File Criticality and Compiler Optimization Effects on Embedded Microprocessors Reliability
- Lintzmayer and others, 2011, Register Allocation with Graph Coloring by Ant Colony Optimization
- Lion and Broman, 2026, Making Time Observable Compiler Correctness for Real-Time C Programs
- Liu, 2007, Bytecode Verification for Enhanced JVM Access Control
- Liu and others, 2021, Relaxed Peephole Optimization A Novel Compiler Optimization for Quantum Circuits
- Liu and others, 2024, A Verified Compiler for a Functional Tensor Language
- Liu and others, 2018, Research of Register Pressure Aware Loop Unrolling Optimizations for Compiler
- Liu and others, 2025, BoostPolyGlot A Structured IR Generation-Based Fuzz Testing Framework for GCC Compiler Frontend
- Liu and others, 2020, A type-and-effect system for object initialization
- Liu and Li, 2018, A novel virtual machine scheduling policy based on performance prediction model
- Liu and others, 2024, Research on Higher-Order Operator Transformation Algorithms for Syntax Transformation-Based Model Checking
- Liu and Lu, 2025, Bi-directional Taint Flow Analysis A High-precision Static Detection Approach for Java Deserialization Vulnerabilities
- Liu and others, 2021, Design and Implementation of Multi-core Parallel Compiler Based on OpenMP
- Liu and others, 2019, Accelerating sequential consistency for Java with speculative compilation
- Liu and others, 2026, Learning Compiler Fuzzing Mutators from Historical Bugs
- Liu and others, 2025, WebAssembly for Container Runtime Are We There Yet?
- Liu and others, 2022, Coverage-guided tensor compiler fuzzing with joint IR-pass mutation
- Liu and others, 2011, Coroutine-Based Synthesis of Efficient Embedded Software From SystemC Models
- Liu and others, 2024, An efficient schedulability analysis based on worst-case interference time for real-time systems
- Liu and others, 2017, Analyzing divergence in bisimulation semantics
- Liu and others, 2013, Linking Algebraic Semantics and Operational Semantics for Web Services Using Maude
- Lo and Suh, 2012, Worst-case execution time analysis for parallel run-time monitoring
- Lochbihler, 2018, Mechanising a Type-Safe Model of Multithreaded Java with a Verified Compiler
- Lööw, 2021, Lutsig a verified Verilog compiler for verified circuit development
- Lopes, 2023, Torchy A Tracing JIT Compiler for PyTorch
- Lopoukhine and others, 2025, A Multi-level Compiler Backend for Accelerated Microkernels Targeting RISC-V ISA Extensions
- Loring and others, 2019, Sound regular expression semantics for dynamic symbolic execution of JavaScript
- Louise, 2011, Improving Branch Prediction Related WCET Abstract Interpretation
- Loveless and others, 2020, A performance-optimizing compiler for cyber-physical digital microfluidic biochips
- Lowther and others, 2023, CHERI Performance Enhancement for a Bytecode Interpreter
- Loy, 1981, Notes on the Implementation of MUSBOX A Compiler for the Systems Concepts Digital Synthesizer
- Lozano and others, 2016, Register allocation and instruction scheduling in Unison

- Lu and others, 2022, Gradual Soundness Lessons from Static Python
- Lu and others, 2011, A new way about using statistical analysis of worst-case execution times
- Lu and others, 2011, A trace-based statistical worst-case execution time analysis of component-based real-time embedded systems
- Lu and others, 2025, Detecting WebAssembly Runtime Bugs With Grammar-Guided Program Mutation
- Lucanu, 2018, Proving Reachability Properties by Coinduction Extended Abstract
- Lucchi and Mazzara, 2007, A pi-calculus based semantics for WS-BPEL
- Luo, 2020, Heap Memory Snapshot Assisted Program Analysis for Android Permission Specification
- Luo and others, 2017, Compiler-Assisted Threshold Implementation against Power Analysis Attacks
- Luo and others, 2026, GeoJSON agents a multi-agent LLM architecture for geospatial analysis-function calling vs. code generation
- Lv and others, 2010, Static worst-case execution time analysis of the µC/OS-II real-time kernel
- Ly and others, 2009, Reduced Model for a PEMFC Stack Automated Code Generation and Verification
- Lyamin, 2018, Method of Formal Program Verification for Post Machine Virtual Laboratory
- Lynn, 1978, Interactive Compiler Proving Using Hoare Proof Rules
- M, 2026, No-Code Backend Orchestrator with Drag and Drop Architecture and AI-Assisted Contextual Code Generation
- M and Murugesh, 2021, Comparative Study of Binary Classification Algorithms to Analyze the Students Performance on Virtual Machine
- Ma, 2025, Lexical Effect Handlers Fast by Design, Correct by Proof
- Ma and others, 2024, Lexical Effect Handlers, Directly
- Ma and others, 2025, Zero-Overhead Lexical Effect Handlers
- Maccabe, 2017, Operating and Runtime Systems Challenges for HPC Systems
- MacMillen, 2001, Nimble Compiler Environment for Agile Hardware. Volume 1
- Madsen and others, 2018, Tail call elimination and data representation for functional languages on the Java virtual machine
- Magdalenić and others, 2011, Implementation Model of Source Code Generator
- Magesty and Montandon, 2026, PromiseAwait A Dataset of JavaScript Migrations from Promises to Async/Await
- Mahajan and others, 2020, Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching
- Mahajan and Ali, 2008, Hybrid evolutionary algorithm for graph coloring register allocation
- Mahajan and Prasad, 2022, Providing Real-time Assistance for Repairing Runtime Exceptions using Stack Overflow Posts
- Maia and others, 2026, Why Just-In-Time Compilation Matters Evaluating Runtime and Energy Efficiency
- Mainland, 2017, Better living through operational semantics an optimizing compiler for radio protocols
- Maity and Ghose, 2026, SAGE A Compiler-assisted Reinforcement Learning-based Offloading Approach under Near-memory Processing Paradigm
- Mäkelä and others, 2016, Compiler assisted dynamic allocation of finite hardware acceleration resources for parallel tasks
- Makki Mohialden and others, 2023, A Comparative Analysis of Python Code-Line Bug-Finding Methods
- Malcolm, 1971, PL360 Revised . A Programming Language for the IBM360
- Maldonado and Carrasco-Sáez, 2026, A Reusable J48-Targeting Python Backend for

Scikit-Learn Workflows Differential Validation Against WEKA J48
- Male and others, 2011, Formalisation and implementation of an algorithm for bytecode verification of @NonNull types
- Mallawarachchi and Jayaweera, 2025, Implementation of Wyltl An Imperative Language with a Dual Interpreter Compiler Architecture
- Mamdouh and others, 2014, On-demand distributed on-card bytecode verification
- Mane Ritesh Pratap and Alpona Das, 2023, Designing a Random Password Generator Using Python Programming Language
- Manjunath, 2010, Colored Steganography Implementation Scheme of Images using Transform Technique
- Mann, 2003, The Compiler Design Handbook Optimisation and Machine Code Generation
- Manna, 1988, TABLOG The Deductive Tableau Programming Language
- Mao and others, 2025, Research on a Lightweight Full-Stack Edge Execution Optimization Framework Based on Serverless and WebAssembly
- Mao and others, 2019, Exploiting Java Stack Forensics for Runtime Monitoring of IoT Services
- Marcelino and others, 2025, Lumos Performance Characterization of WebAssembly as a Serverless Runtime in the Edge-Cloud Continuum
- Marcelino and Nastic, 2023, CWASI A WebAssembly Runtime Shim for Inter-function Communication in the Serverless Edge-Cloud Continuum
- Marcozzi and others, 2019, Compiler fuzzing how much does it matter?
- Coroutines: A Programming Methodology, a Language Design and an Implementation
- Marref, 2011, Fully-automatic derivation of exact program-flow constraints for a tighter worst-case execution-time analysis
- Marriott and others, 1994, Denotational abstract interpretation of logic programs
- Marshall, 1982, The linear graph package, a compiler building environment
- Martinsen and others, 2016, Combining thread-level speculation and just-in-time compilation in Google’s V8 JavaScript engine
- Márton and Porkoláb, 2017, Unit Testing in C++ with Compiler Instrumentation and Friends
- Mary-Anne K Posenau, 1993, Unstructured Grid Generation Techniques and Software
- Massey and Olivier, 2026, WASP Stack protection for WebAssembly
- Massidda and others, 2024, Bringing Binary Exploitation at Port 80 Understanding C Vulnerabilities in WebAssembly
- Mastorou and others, 2022, Coinduction inductively mechanizing coinductive proofs in Liquid Haskell
- Masuko and Asai, 2009, Direct implementation of shift and reset in the MinCaml compiler
- Matsikoudis and Stergiou, 2014, First Draft of the act Programming Language
- Matsubara and others, 2025, Seamless Self-Healing in WebAssembly Container Orchestration with Runtime-Neutral Checkpointing
- Matteo R. Donelli, 2025, Compiler-Assisted Optimization Using Neural Code Embeddings for Heterogeneous Architectures
- Maurer and others, 2017, Compiling without continuations
- Mazaher and Berry, 1985, Deriving a compiler from an operational semantics written in VDL
- McNerney, 1991, Verifying the correctness of compiler transformations on basic blocks using abstract interpretation
- Medeiros and others, 2018, Evaluation of Compiler Optimization Flags Effects on Soft Error Resiliency
- Meghzili and others, 2017, On the Verification of UML State Machine Diagrams to Colored Petri Nets Transformation Using Isabelle/HOL
- Mehdi Pourhashem Kallehbasti and Ghafari, 2023, Naturalistic Static Program Analysis
- Mehta and Yew, 2015, Improving compiler scalability optimizing large programs at small

price
- Meier and others, 2025, CertiCoq-Wasm A Verified WebAssembly Backend for CertiCoq
- Melnyk and Kozak, 2019, Easy Universal Translator as an Alternative Compiler-Compiler
- Melo Alves and others, 2018, An Interference-Aware Virtual Machine Placement Strategy for High Performance Computing Applications in Clouds
- Melquiond and Moreau, 2024, A Safe Low-Level Language for Computer Algebra and Its Formally Verified Compiler
- Melrose and others, 2015, Just in Time
- Menetrey and others, 2021, Twine An Embedded Trusted Runtime for WebAssembly
- Menetrey and others, 2022, WaTZ A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone
- Ménétrey and others, 2024, A Comprehensive Trusted Runtime for WebAssembly With Intel SGX
- Meng and others, 2020, Survey on Estimation and Optimization of Worst-case Execution Time with Energy Consumption Constraint
- Merazga and others, 2025, Worst-Case Execution Time Analysis of a Real-Time System based on Arduino in CAN Network
- Meybodi, 2015, The links between just-in-time practices and alignment of benchmarking performance measures
- Meyer and Wolff, 2019, Decoupling lock-free data structures from memory reclamation for static analysis
- Mguidich and others, 2016, Time-Accurate ASM as a Refinement Scheme for Worst-Case Execution Time Estimation in Hard Real-Time Systems
- Mhaske and others, 2015, A 2.48Gb/s FPGA-based QC-LDPC decoder An algorithmic compiler implementation
- Michaud and others, 2024, Robust Stack Smashing Protection for WebAssembly
- Michels and others, 2015, A new probabilistic constraint logic programming language based on a generalised distribution semantics
- Midtgaard, 2025, Property-Based Testing of OCaml 5's Runtime System
- MIDZIC and NOVAK, 2025, Performance Comparison of WebAssembly and Phaser in Procedural Maze Generation
- Mihelic and others, 2021, A denotational semantics of a concatenative/compositional programming language
- Milano and others, 2022, A flexible type system for fearless concurrency
- Milos and others, 1984, Direct implementation of compiler specifications or the pascal p-code compiler revisited
- Minato and others, 2025, Implementation and Evaluation of a System Call Moving Target Defense Applied Multiple Times at Runtime for Binary Injections
- Miné, 2017, Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation
- Mitra and Givargis, 2009, Session details Microfluidics, worst-case execution time, and cache optimization
- Mittal and others, 2021, Towards an Approach for Translation Validation of Thread-level Parallelizing Transformations using Colored Petri Nets
- Mizikovskiy, 2024, Methodology of management cost accounting and calculation of the cost of not amortised organizational and technological equipment for the production of industrial enterprise products
- Mizobuchi and Takayama, 2017, Two improvements to detect duplicates in Stack Overflow
- 1997, Modern compiler implementation in C Basic techniques
- 1997, Modern compiler implementation in Java Basic techniques
- 1997, Modern compiler implementation in ML Basic techniques
- 1998, Modern compiler implementation in Java Revised and expanded edition
- Mohan, 2008, Worst-case execution time analysis of security policies for deeply embed-

ded real-time systems
- Mohnen, 1997, A COMPILER CORRECTNESS PROOF FOR THE STATIC LINK TECHNIQUE BY MEANS OF EVOLVING ALGEBRAS
- Moliavko and others, 2019, uJVM Lightweight Java Virtual Machine for embedded systems
- MolinaFraticelli, 2012, Auto Code Generation for Simulink-Based Attitude Determination Control System
- Mondal and others, 2023, Investigating Technology Usage Span by Analyzing Users' QandA Traces in Stack Overflow
- Mondal and others, 2016, Embedded Emotion-based Classification of Stack Overflow Questions Towards the Question Quality Prediction
- Mondshein, 1967, VITAL COMPILER SYSTEM REFERENCE MANUAL
- Monniaux, 2024, Memory Simulations, Security and Optimization in a Verified Compiler
- Monsuez, 1995, Using abstract interpretation to define a strictness type inference system
- Montenegro and others, 2015, Space consumption analysis by abstract interpretation Reductivity properties
- Moody and Richards, 1980, A coroutine mechanism for BCPL
- Moor, 1982, An applicative compiler for a parallel machine
- Moore, 1968, Data Processing with the Compiler Compiler
- Moret and others, 2011, Polymorphic bytecode instrumentation
- Morford, 1999, A Case Study Using 3D Pre-Stack Depth Migration To Improve The Sub-Salt Image In Marbella, Mexico, Gulf Of Mexico
- Morford, 2000, A case study Using 3d pre-stack depth migration to image sub-salt sediments and fault zones in Marbella, Mexico
- Morford, 2001, Analysis Of The Effects Of Varying The Anti - Alias Filter In Kirchoff Pre-Stack Depth Migration
- MORIGUCHI and others, 2016, Verification of Content-Centric Networking Using Proof Assistant
- Moron and Wallentowitz, 2023, Support for Just-in-Time Compilation of WebAssembly for Embedded Systems
- Mosaner and others, 2022, Improving Vectorization Heuristics in a Dynamic Compiler with Machine Learning Models
- Moser and Schneckenreither, 2020, Automated amortised resource analysis for term rewrite systems
- Moskalenko, 2025, Application of WebAssembly for High-Performance Client-Side Media Content Analysis
- Moss and others, 2016, The ARES High-Level Intermediate Representation
- Mössenböck, 1984, Ein einfacher Compiler-Compiler für Mikrocomputer / A simple compiler-compiler for microcomputer
- Mosses, 2004, Modular structural operational semantics
- Mosses, 2015, Semantics of programming languages Using Asf+Sdf
- Mosterman and Zander, 2015, Cyber-physical systems challenges a needs analysis for collaborating embedded software systems
- Mpeis and others, 2021, Developer and user-transparent compiler optimization for interactive applications
- Mückenschnabel, 2024, Algebraic Effect Handlers with Bidirectional Type-Checking
- Mukherjee and others, 2024, HLS-IRT Hardware Trojan Insertion through Modification of Intermediate Representation During High-Level Synthesis
- Müller and others, 2026, WasmWeaver A Framework for Runtime-Aware WebAssembly Program Generation with Reinforcement Learning
- Müller and others, 2023, From Capabilities to Regions Enabling Efficient Compilation of Lexical Effect Handlers
- Muller and Zhou, 1992, Abstract interpretation in weak powerdomains

- Munley and others, 2024, LLM4VV Developing LLM-driven testsuite for compiler validation
- Murthy and Mellor-Crummey, 2015, A Compiler Transformation to Overlap Communication with Dependent Computation
- Mushtaq and others, 2015, Calculation of worst-case execution time for multicore processors using deterministic execution
- Müssig, 2019, Just enough, just in time, just for me
- Musumbu, 2008, Static Checking by Means of Abstract Interpretation
- Muts and Falk, 2020, Compiler-based WCET prediction performing function specialization
- Mycroft, 1993, Completeness and predicate-based abstract interpretation
- Mykola, 2024, USING ASYNCHRONOUS PROGRAMMING IN PYTHON TO IMPROVE APPLICATION PERFORMANCE
- Myreen, 2010, Verified just-in-time compiler on x86
- Myreen, 2021, A minimalistic verified bootstrapped compiler proof pearl
- Mzid and others, 2022, Use of Compiler Intermediate Representation for Reverse Engineering A Case Study for GCC Compiler and UML Activity Diagram
- Na and others, 2016, JavaScript Parallelizing Compiler for Exploiting Parallelism from Data-Parallel HTML5 Applications
- Nadi and Treude, 2020, Essential Sentences for Navigating Stack Overflow Answers
- Nagata and others, 2024, Evaluation of Interoperability of CNN Models between MATLAB and Python Environments Using ONNX Runtime Model
- Naik, 2013, Session details Compiler validation
- Nair, 2012, A Formal Semantics for Ciset and Ciset Relation Operators
- Nair and Sarasamma, 2006, Formal Semantics of Ciset Relational Operators
- Naish, 2015, Sharing analysis in the Pawns compiler
- Nakata and Matsubara, 2025, Self-Hosted WebAssembly Runtime for Runtime-Neutral Checkpoint/Restore in Edge-Cloud Continuum
- Nakata and Uustalu, 2015, A Hoare logic for the coinductive trace-based big-step semantics of While
- Namakonov and Podkopaev, 2019, Compilation of OCaml memory model into Power
- Nanthaamornphong and others, 2015, Bytecode-based class dependency extraction tool Bytecode-CDET
- Nappa and others, 2019, Fast Parallel Equivalence Relations in a Datalog Compiler
- Narkthong and others, 2024, ALLI/O Diagram An Action-based Visual Programming Language for Embedded System
- Natarajan and Broman, 2020, Temporal Property-Based Testing of a Timed C Compiler using Time-Flow Graph Semantics
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1988, Ada Compiler Validation Summary Report Certificate Number 880708S1. 09151, SoftTech, Inc., Ada 86, Version 3.21 VAX 11/780-11/785 Host and Intel IAPX 80386R Target
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1988, Ada Compiler Validation Summary Report Certificate Number 880719S1. 09154, Naval Underwater Systems Command, ADAUYK43 ALS/N Ada/L , Version 1.0, VAX 11/785 Host and AN/UYK-43 Target
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1988, Ada Compiler Validation Summary Report Compiler Name ADE/32 Revision 3.00. Certificate Number 880527S1.09113. Host MV/20000 under AOS/VS, Revision 7.56. Target ROLM HAWK/32 under AOS/VS, Revision 7.56
- NATIONAL BUREAU OF STANDARDS GAITHERSBURG MD, 1989, Ada Compiler Validation Summary Report Certificate Number 880624S1. 09132, Control Data Corporation CYBER 180 Ada Compiler, Version 1.1 HOST and TARGET COMPUTER CYBER 180-930-31
- Nayvelt and Bear, 2008, Case study Anisotropic Pre-Stack Depth Migration on the

Louisiana Shelf
- Proof-carrying code
- Necula, 2000, Translation validation for an optimizing compiler
- Necula and Lee, 1998, The design and implementation of a certifying compiler
- Necula and Lee, 2004, The design and implementation of a certifying compiler
- Nedoria, 2023, Programming Language for Teaching Compilation and Transformation Technologies
- Negrini, 2026, Whole-value analysis by abstract interpretation
- Neis and others, 2015, Pilsner a compositionally verified compiler for a higher-order imperative language
- Nejjar and others, 2024, LLMs for science Usage for code generation and data analysis
- Nemer and others, 2007, Improving the Worst-Case Execution Time Accuracy by Inter-Task Instruction Cache Analysis
- 2016, Network-Aware Virtual Machine Placement in the Cloud
- New and others, 2023, Gradual Typing for Effect Handlers
- Newey, 1975, Formal Sematics of LISP with Applications to Program Correctness
- Nezamabadi and others, 2026, Verified VCG and Verified Compiler for Dafny
- Ngo and others, 2015, Modular translation validation of a full-sized synchronous compiler using off-the-shelf verification tools
- Nguyen and others, 2019, Integrating Static Program Analysis Tools for Verifying Cautions of Microcontroller
- Nguyen and McCaskey, 2022, Extending Python for Quantum-classical Computing via Quantum Just-in-time Compilation
- Nguyen and others, 2023, Effect Handlers for Programmable Inference
- Ni and Li, 2025, Interleaving Large Language Models for Compiler Testing
- Nickerson, 1990, Graph coloring register allocation for processors with multi-register operands
- Nielsen, 1997, Compiler Support for Message Passing Systems
- Nielsen, 1998, Compiler Support for Message Passing Systems
- Nielson, 1987, Strictness analysis and denotational abstract interpretation
- Nielson, 1988, Strictness analysis and denotational abstract interpretation
- Niephaus and others, 2018, Live Multi-language Development and Runtime Environments
- Nikhil Sripathi Rao, 2024, WebAssembly Revolutionizing Web User Interface Development through Performance and Cross-Language Integration
- Nilsen and Rygg, 1995, Worst-case execution time analysis on modern processors
- Nipkow, 2003, Java Bytecode Verification
- Nishizaki, 2017, Linear lambda calculus with non-linear first-class continuations
- Nishizaki, 2017, Type Inference of Linear Lambda Calculus with First-Class Continuations
- Nishizaki, 2019, ML Polymorphism of Linear Lambda Calculus with First-class Continuations
- Niu and others, 2024, FAIR Flow Type-Aware Pre-Training of Compiler Intermediate Representations
- 2026, NOCI-COMPILER A THEORETICAL ARCHITECTURE FOR THE SEMANTIC TRANSLATION OF NOCICEPTIVE SIGNALS INTO A DIGITAL PAIN ALPHABET
- Noguchi and others, 2021, Implementing Algebraic Effects and Handlers in Non-functional Programming Languages
- Norris and Pollock, 1994, Register allocation over the program dependence graph
- Nougrahiya and Nandivada, 2024, Homeostasis Design and Implementation of a Self-Stabilizing Compiler
- Nowicki and others, 2021, Performance evaluation of Java/PCJ implementation of parallel algorithms on the cloud extended version
- Nyangaresi and Ma, 2022, A Formally Verified Message Validation Protocol for Intelli-

gent IoT E-Health Systems
- Oates and others, 1998, The Application of Pre-Stack Depth Migration using Topographic Analysis to Aid in the
- O'Callahan, 1999, A simple, comprehensive type system for Java bytecode subroutines
- Odaira and others, 2010, Coloring-based coalescing for graph coloring register allocation
- Odegard and others, 2014, Model-Based GN and C Simulation and Flight Software Development for Orion Missions beyond LEO
- Oehlert and others, 2018, Mitigating Data Cache Aging through Compiler-Driven Memory Allocation
- Oh and others, 2022, CASH-RF A Compiler-Assisted Hierarchical Register File in GPUs
- Oh and Kim, 2026, Detecting Compiler-Introduced Security Bugs via IR Mutation and Coverage-Guided Fuzzing
- Oh and others, 2015, Bytecode-to-C Ahead-of-Time Compilation for Android Dalvik Virtual Machine
- Oiwa, 2009, Implementation of the memory-safe full ANSI-C compiler
- Okasaki and others, 1994, Call-by-need and continuation-passing style
- O Kim and Lee, 2026, Amortised neural acoustic tomography domain-specific encoder design and topology-dependent Eikonal analysis
- Oliva and others, 1995, The VLISP verified PreScheme compiler
- O'Loughlin and Gillam, 2015, Addressing Issues of Cloud Resilience, Security and Performance through Simple Detection of Co-locating Sibling Virtual Machine Instances
- Olteanu and Oprişa, 2025, LambdaGo A Functional Extension of the Go Programming Language
- 2024, Online platform learning the Java programming language development, implementation and efficiency
- Ooashi and others, 1992, ASL program written in abstract sequential machine style and its compiler
- Orlic, 2010, Implementation by capture with executable UML
- Orlitsky, 2002, Scalar versus vector quantization worst case analysis
- Orlov, 2017, Program Source Code Static Analysis for Memory Access Error Detection Using Backwards Execution
- Orr and Henderson, 2000, Space Shuttle Software Development and Certification
- Ortiz, 2022, Using WebAssembly to Teach Code Generation in a Compiler Design Course
- Osborne and others, 2024, Awkward Just-In-Time JIT Compilation A Developer's Experience
- Oshita and others, 2019, Improving User Experience of C Programming Language Learning System for Novices
- Othman and El Ghoul, 2022, BuHamad - The first Qatari virtual interpreter for Qatari Sign Language
- Ottoni, 2018, HHVM JIT a profile-guided, region-based compiler for PHP and Hack
- Ouadjaout and others, 2016, Static analysis by abstract interpretation of functional properties of device drivers in TinyOS
- Ozdemir and others, 2022, CirC Compiler infrastructure for proof systems, software verification, and more
- Padmasudha Kannan and others, 2021, Automated high-order curved mesh generator with high-level dynamic programming language julia for photonic applications
- Painter, 1970, Effectiveness of an optimizing compiler for arithmetic expressions
- Pai T and others, 2020, A Systematic Literature Review of Lexical Analyzer Implementation Techniques in Compiler Design
- Pałka and others, 2011, Testing an optimising compiler by generating random lambda terms
- Palsberg, 1992, Provably Correct Compiler Generation
- Palsberg and Schwartzbach, 1992, Binding Time Analysis Abstract Interpretation

vs. Type Inference
- Pan and others, 2026, A Backend-Agnostic Compiler for Approximate Query Processing with Probabilistic Tensor Algebra
- Pan and others, 2015, Nonvolatile main memory aware garbage collection in high-level language virtual machine
- Pandey, 2025, Compiler Design and Its Construction
- Panigrahi and Karfa, 2023, Translation Validation of Information Leakage of Compiler Optimizations
- Panigrahi and Karfa, 2023, An Investigation into the Security of Register Allocation with Spilling and Splitting
- Pankhurst, 1968, GULP—A compiler-compiler for verbal and graphic languages
- Pant and others, 2022, Automatic Software Engineering Position Resume Screening using Natural Language Processing, Word Matching, Character Positioning, and Regex
- Papadakis and others, 2015, Trivial Compiler Equivalence A Large Scale Empirical Study of a Simple, Fast and Effective Equivalent Mutant Detection Technique
- Papadimitriou and others, 2021, Automatically exploiting the memory hierarchy of GPUs through just-in-time compilation
- Pape and others, 2017, Adaptive just-in-time value class optimization for lowering memory consumption and improving execution time performance
- Paraman and Murthy, 2023, Analysis of benchmark program results of worst case execution time for multithreaded programs
- Parashar and others, 2026, A Comparative Architectural and Performance Analysis of WebAssembly and JavaScript for Computationally Intensive Web Applications
- Park and others, 2005, Implementation of Worst Case Execution Time Analysis Tool For Embedded Software based on XScale Processor
- Park and others, 2023, Bespoke Virtual Machine Orchestrator An Approach for Constructing and Reconfiguring Bespoke Virtual Machine in Private Cloud Environment
- Park and others, 2001, Register Allocation for Banked Register File
- Park and others, 2015, KJS a complete formal semantics of JavaScript
- Park and others, 2018, A formal verification tool for Ethereum VM bytecode
- Parra, 2026, SurtGIS A high-performance raster geospatial analysis library in Rust with WebAssembly and Python support
- Parwez and others, 2025, Signease Machine Learning Based Sign Language Interpreter
- Pasareanu and others, 2009, Model Based Analysis and Test Generation for Flight Software
- Pasupuleti, 2025, AI-Guided Quantum Compiler Design Using Superalgebraic Symmetries
- Patel and others, 2025, SySTeC A Symmetric Sparse Tensor Compiler
- Patel and Lee, 2016, Dynamic Analysis of Multi-threaded Embedded Software to Expose Atomicity Violations
- Pathiran and Prakash, 2014, Design and implementation of a model-based PI-like control scheme in a reset configuration for stable single-loop systems
- Patterson and Ahmed, 2019, The next 700 compiler correctness theorems functional pearl
- Pattinson and Schröder, 2016, Program Equivalence is Coinductive
- Paul and others, 2020, Improving execution efficiency of just-in-time compilation based query processing on GPUs
- Pauli and Soffa, 1980, Coroutine behaviour and implementation
- Paulson, 1982, A semantics-directed compiler generator
- Pavlovskiy and Platov, 2025, RESEARCH ON CODE GENERATION OF C/C++ COMPILER FOR HIGH-PERFORMANCE COMPUTING IN MICROPROCESSOR SYSTEMS
- Payne, 1978, A formalised technique for expressing compiler exercisers
- Pečimúth, 2023, Remote Just-in-Time Compilation for Dynamic Languages
- Pelsmaeker and others, 2019, Towards language-parametric semantic editor services

based on declarative type system specifications
- Penev and Dimitrov, 2021, Design of a Virtual Machine for Training Compilers
- Peng and others, 2004, Code sharing among states for stack-caching interpreter
- Petchartee, 2025, A Universal Quantum Compiler GPT Multi-Framework Optimization and Translation Using Large Language Models
- Peterson and others, 2017, Addressing Global Data Dependencies in Heterogeneous Asynchronous Runtime Systems on GPUs
- Petkovski and Bradey, 2001, The success of pre-stack depth migration over the Anama structure in the Papuan Foreland Basin, PNG a case history
- Petrosino and others, 2023, Cross-Paradigm Interoperability Between Jadescript and Java
- Pham and Odersky, 2024, Stack-Copying Delimited Continuations for Scala Native
- Phulia and others, 2020, OOElala order-of-evaluation based alias analysis for compiler optimization
- Pichler and others, 2023, Hybrid Execution Combining Ahead-of-Time and Just-in-Time Compilation
- Pieters and others, 2017, Handlers for Non-Monadic Computations
- PIETERS and SCHRIJVERS, 2020, Faster coroutine pipelines A reconstruction
- Pike and others, 2012, Experience Report A Do-It-Yourself High-Assurance Compiler
- Pinckney and others, 2020, Wasm/k delimited continuations for WebAssembly
- Pizzolotto and Inoue, 2021, Identifying Compiler and Optimization Level in Binary Code From Multiple Architectures
- P. Jeannot, 1994, Robust Kirchhoff pre-stack depth imaging with semi-gridded rays
- P. Jeannot and Berranger, 1994, Ray-mapped focusing - A migration velocity analysis for Kirchoff pre-stack depth imaging
- Pla, 2004, Weapon System Software Technology Support WSSTS . Delivery Order 0008 Real-Time Java for Embedded Systems RTJES
- Plangger and Krall, 2016, Vectorization in PyPy's Tracing Just-In-Time Compiler
- Plasterie and Chagalov, 2006, Wave Equation versus Kirchhoff pre-stack depth migration algorithms ? An Australian case study
- Pleban, 1979, The use of transition matrices in a recursive-descent compiler
- Pleban, 1984, Compiler prototyping using formal semantics
- Plishka and Ifarraguerri, 1993, Evaluation of Alsys 037 Ada Compiler
- Ploensin and others, 2021, Code Transformation Impact on Compiler-based Optimization A Case Study in the CMSSW
- PLOTKIN, 2004, The origins of structural operational semantics
- Plotkin and Pretnar, 2008, A Logic for Algebraic Effects
- Handlers of algebraic effects
- Plotkin and Pretnar, 2013, Handling Algebraic Effects
- Pockstaller and others, 2023, Comparing the Energy Consumption of WebAssembly and JavaScript in Mobile Browsers
- Poletanovic and others, 2022, Implementation of Machine Outliner for nanoMIPS in the LLVM Compiler Infrastructure
- Polito and others, 2022, Interpreter-guided differential JIT compiler unit testing
- Poonguzhali and Vinodha, 2014, IMPLEMENTATION OF ANTI-RESET WINDUP SCHEME IN PI CONTROLLER FOR SPHERICAL TANK PROCESS
- Popeea and Chin, 2004, A type system for resource protocol verification and its correctness proof
- Porcher, 1995, Benchmarking the POMPC compiler on the Connection Machine CM-2
- Posner and others, 2025, Toward Dynamic Resource Management An Asynchronous Many-Task AMT Runtime System leveraging Dynamic Processes with PSets DPP
- Postema and others, 2022, Testing a PL/I Compiler Using Precomputation-based Program Generation
- Pous, 2015, Coinductive techniques, from automata to coalgebra

- Pous, 2016, Coinduction All the Way Up
- Powell, 1984, A portable optimizing compiler for Modula-2
- Power, 2006, The Universal Algebra of Computational Effects Lawvere Theories and Monads
- Pratt, 1997, Second Calculus of Binary Relations as a Concurrent Programming Language
- Prenzel and Provost, 2017, Dynamic Software Updating of IEC 61499 Implementation Using Erlang Runtime System
- Pretnar, 2014, Inferring Algebraic Effects
- Pretnar, 2015, An Introduction to Algebraic Effects and Handlers. Invited tutorial paper
- Prinz, 2023, Compilation of Distributed Programs to Services Using Multiple Programming Languages
- Pritchard, 1976, A proof rule for multiple coroutine systems
- 2020, Proceedings of the 2020 International Conference on Embedded Software EMSOFT
- 2002, Proceedings Second IEEE International Workshop on Source Code Analysis and Manipulation
- 2003, Proceedings Third IEEE International Workshop on Source Code Analysis and Manipulation
- Theory and practice of coroutines with snapshots
- Proy and others, 2017, Compiler-Assisted Loop Hardening Against Fault Attacks
- Puchol and others, 1998, An Operational Semantics and Compiler for Real-Time Specifications1
- Puffitsch, 2016, Efficient Worst-Case Execution Time Analysis of Dynamic Branch Prediction
- Punchihewa and Wu, 2021, Safe mutation with algebraic effects
- Punnoose, 2020, Ensuring completeness of formal verification with Gap Free Verification
- Puranik, 2025, Bridging Formal Methods and Software Engineering Through a Tagless-Final Embedded DSL for Program Semantics
- Puschner, 1997, Worst-Case Execution Time Analysis at Low Cost
- Puschner, 1998, Worst-case execution-time analysis at low cost
- Pyzik, 2023, Call-By-Name Is Just Call-By-Value with Delimited Control
- Qassir, 2025, MyDSL Front-End Compiler Design for a User-Friendly Language Supporting Hybrid Meta-Heuristics
- Qian, 2000, Standard fixpoint iteration for Java bytecode verification
- Qian and others, 2026, Thinking Fast and Correct Automated Rewriting of Numerical Code through Compiler Augmentation
- Qian and others, 2025, FreeWavm Enhanced WebAssembly Runtime Fuzzing Guided by Parse Tree Mutation and Snapshot
- Qin and others, 2026, Augmenting LLM Code Translation with Compiler Analysis for C to Triton Kernel Generation
- Qu and others, 2023, Scope-based Compiler Differential Testing
- 2025, Quantum Software Engineering Algorithm Design, Error Mitigation, and Compiler Optimization for Fault-Tolerant Quantum Computing
- Queiroz Junior and da Silva, 2015, Finding Good Compiler Optimization Sets - A Case-based Reasoning Approach
- Queiroz Junior and others, 2020, Finding Effective Compiler Optimization Sequences A Hybrid Approach
- Quiring and others, 2021, 3CPS The Design of an Environment-Focussed Intermediate Representation
- Radonić and others, 2014, One solution of loop invariant code motion compiler optimisation
- Radooevii and Magdalenii, 2014, Python Implementation of Source Code Generator Based on Dynamic Frames

- Raj, 2016, Performance analysis of different specifications of copy propagation transformation using machine SUIF compiler infrastructure
- Raju Cherukuri, 2024, Building Scalable Web Applications Best Practices for Backend Architecture
- Ramdani and others, 2025, Pengembangan Backend Aplikasi Geoproperty dengan Golang di PT. Nerdvana Solusi Teknologi
- Ramesh and others, 2024, ThriveJIT Dynamic Just-In-Time Compilation for Efficient Execution of Arithmetic Expressions
- Ramkumar and Kale, 1990, A Chare kernel implementation of a parallel Prolog compiler
- Rand, 2022, Writing and verifying a Quantum optimizing compiler keynote
- Ranganathan and others, 2020, Hybrid Scalable Action Rule
- Ranjan and others, 2025, ClosureX Compiler Support for Correct Persistent Fuzzing
- Ranzato, 2013, Session details Abstract interpretation
- Rao and others, 2021, SODA A Semantics-Aware Optimization Framework for Data-Intensive Applications Using Hybrid Program Analysis
- Raskovsky, 1982, Denotational semantics as a specification of code generators
- Rath and others, 2018, Interoperability-Guided Testing of QUIC Implementations using Symbolic Execution
- Ravanbakhsh and Sankaranarayanan, 2014, Infinite horizon safety controller synthesis through disjunctive polyhedral abstract interpretation
- Ravedutti Lucio Machado and others, 2025, P4IRS An intermediate representation and compiler for parallel and performance-portable particle simulations
- Reb and others, 2008, A JML Compiler Based on AspectJ
- Recharla, 2025, Parallel Sparse Matrix Algorithms in OCaml v5 Implementation, Performance, and Case Studies
- Reddy and others, 2026, Compiler-Assisted Instruction Fusion
- Refaie and Thyabat, 2015, Effect of just-in-time selling strategy on firms' performance in Jordan
- Eliminating stack overflow by abstract interpretation
- Reinhardt and others, 2018, Augmenting stack overflow with API usage patterns mined from GitHub
- Reis and others, 2017, Compiler Techniques for Efficient MATLAB to OpenCL Code Generation
- Reiss, 1983, Generation of Compiler Symbol Processing Mechanisms from Specifications
- Reitz and Posner, 2025, Stackless vs. Stackful Coroutines A Comparative Study for RDMA-based Asynchronous Many-Task AMT Runtimes
- Ren and others, 2026, SpiceFuzz LLM-Based Fuzzing for Spice Circuit Simulator Tools Bug Detection
- Reshef and Roth, 2003, Anisotropy Corrections after Pre-Stack Depth Migration
- Definitional interpreters for higher-order programming languages
- R. Granli, 1993, Imaging salt with pre-stack depth migration
- Ricardo and others, 2025, On the Practicality of LLM-Based Compiler Fuzzing
- Ricketts and others, 2015, Towards verification of hybrid systems in a foundational proof assistant
- Rinard, 2026, Testing, Credible Compilation, and Verification in the Axon Verified Compiler in Lean and Claude Code
- Ritchie and others, 2005, Challenges and opportunities in pre-stack depth imaging of legacy seismic data an overthrust belt case study
- Rivera and others, 2021, An Interval Compiler for Sound Floating-Point Computations
- Rivera and others, 2022, A Compiler for Sound Floating-Point Computations using Affine Arithmetic
- Robbins, 1984, Engineering a high-capacity Pascal compiler for high performance
- Roberts, 1995, 3D Post Stack Depth Migration in the UK Southern North Sea - a Case Study

- Robin and Khan, 2024, An open-source P416 compiler backend for reconfigurable match-action table switches Making networking innovation accessible
- Rodríguez and others, 2009, CPPC a compiler-assisted tool for portable checkpointing of message-passing applications
- Rodríguez and others, 2016, Proving Correctness of a Compiler Using Step-indexed Logical Relations
- Rodriguez Ferrandez and others, 2023, Worst Case Execution Time and Power Estimation of Multicore and GPU Software A Pedestrian Detection Use Case
- Roessle and others, 2019, Formally verified big step semantics out of x86-64 binaries
- Rogers, 1993, Ada Embedded Computer Software Support AECSS
- Rohr and Lindenstruth, 2017, Fast Failure Erasure Encoding Using Just in Time Compilation for CPUs, GPUs, and FPGAs
- Rokotyanskaya and Abramov, 2023, Studying WebAssembly and comparison of its performance with JavaScript
- Romano and Wang, 2023, When Function Inlining Meets WebAssembly Counterintuitive Impacts on Runtime Performance
- ROMPF and AMIN, 2019, A SQL to C compiler in 500 lines of code
- Rong, 2009, Tree register allocation
- Rong and others, 2025, IRFuzzer Specialized Fuzzing for LLVM Backend Code Generation
- Rosà and others, 2023, Automated Runtime Transition between Virtual and Platform Threads in the Java Virtual Machine
- Rose, 2003, Lightweight Bytecode Verification
- Rosenblum and others, 2010, Extracting compiler provenance from program binaries
- Rosing and others, 1990, The DINO Parallel Programming Language
- Roşu, 2018, Finite-trace linear temporal logic coinductive completeness
- Rot and others, 2016, Proving language inclusion and equivalence by coinduction
- Rowland and Perugini, 2025, The Formal Semantics and Implementation of a Domain-Specific Language for Mixed-Initiative Dialogs
- Roy, 2023, A Theorem Proving Approach to Programming Language Semantics
- Roy and others, 2019, Security Analysis and Efficient Implementation of Code-based Signature Schemes
- Royer, 1986, Transformations of denotational semantics in semantics directed compiler generation
- Royuela and others, 2015, Compiler analysis for OpenMP tasks correctness
- Ruan and Chen, 2015, Performance-to-Power Ratio Aware Virtual Machine VM Allocation in Energy-Efficient Clouds
- Ruberg and others, 2017, Embedded software performance estimations at different compiler optimisation levels
- Ruchkin and others, 2016, Smart compiler embedded computing systems based on cluster parallelism
- Ruchkin and others, 2017, Frame model of a compiler of cluster parallelism for embedded computing systems
- Rudmik and Lee, 1979, Compiler design for efficient code generation and program optimization
- Rudolph and Thiemann, 2010, Mnemonics type-safe bytecode generation at run time
- 1981, Ruggedized minicomputer hardware and software topics, 1981 Proceedings of the 4th ROLM MIL-SPEC Computer User's Group Conference
- Rushby, 2002, Formally Verified Hardware Encapsulation Mechanism for Security, Integrity, and Safety
- Russinoff, 1992, A verified prolog compiler for the Warren Abstract Machine
- Ryu and others, 2019, Toward Analysis and Bug Finding in JavaScript Web Applications in the Wild
- S and others, 2025, Impact of Peer-Based Feedback Mechanisms on Understanding Pro-

gramming Language
- Sabry and Felleisen, 1992, Reasoning about programs in continuation-passing style
- Sabry and Felleisen, 1993, Reasoning about programs in continuation-passing style
- Sabry and Felleisen, 1994, Is continuation-passing useful for data flow analysis?
- Sack, 2025, Interfacing Programming Language Semantics and Pragmatics What Does “Hello, World” Mean?
- Sadanand Giri and others, 2024, Green threads of progress Natural fibers reshaping wastewater cleanup strategies, a review
- Sadasue and Isshiki, 2023, LLVM-C2RTL C/C++ Based System Level RTL Design Framework Using LLVM Compiler Infrastructure
- Sadat, 2005, A Compiler Driven Simulation Technique for the Analysis of Digital Logic Circuit
- Sager, 1985, A technique for creating small fast compiler frontends
- Sah and others, 2018, An Efficient Hardware-Oriented Runtime Approach for Stack-based Software Buffer Overflow Attacks
- Sahkhar and others, 2022, Efficient Cloudlet Allocation to Virtual Machine to Impact Cloud System Performance
- Sai and others, 2022, Machine learning-based malware detection using stacking of opcodes and bytecode sequences
- Saieva and Kaiser, 2020, Binary Quilting to Generate Patched Executables without Compilation
- Salama and others, 2018, Online programming language-Learning management system
- Salánki and Sarvajcz, 2019, Development of a Gait Recognition System in NI LabVIEW Programming Language
- Salapura and Harper, 2015, High Performance Virtual Machine Recovery in the Cloud
- Salapura and Harper, 2015, Remote Restart for a High Performance Virtual Machine Recovery in a Cloud
- SALEH and SCHRIJVERS, 2016, Efficient algebraic effect handlers for Prolog
- Salim and others, 2019, Towards a WebAssembly standalone runtime on GraalVM
- Sambasivam and others, 2021, Writing P4 compiler backend for packet processing engines
- Samet, 1976, Compiler testing via symbolic interpretation
- Samet, 1977, A normal form for compiler testing
- Samet, 1980, A Coroutine Approach to Parsing
- Samiei and others, 2024, Worst-Case Execution Time Analysis of Real-Time Robotic Algorithms Using Reinforcement Learning
- Sammler and others, 2022, Islaris verification of machine code against authoritative ISA semantics
- Sanada, 2023, Category-Graded Algebraic Theories and Effect Handlers
- SANADA, 2024, Algebraic effects and handlers for arrows
- Sanders and others, 2020, Robustness Analysis of Scaled Resource Allocation Models Using the Imperial PEPA Compiler
- Sangiorgi, 2022, From enhanced coinduction towards enhanced induction
- Sanmorino, 2012, Development of computer assisted instruction CAI for compiler model The simulation of stack on code generation
- Santhi and others, 2015, The Simian concept Parallel Discrete Event Simulation with interpreted languages and just-in-time compilation
- Santhiar and Kanade, 2017, Static deadlock detection for asynchronous C# programs
- SanthiKumar and others, 2025, Enhancing Machine Learning Performance with AI-Based Virtual Machine Load Balancing
- Santo and others, 2009, Continuation-Passing Style and Strong Normalisation for Intuitionistic Sequent Calculi
- Santone, 2011, Clone detection through process algebras and Java bytecode
- Santos and others, 2024, Assessing the Impact of Compiler Optimizations on GPUs Reli-

ability
- Santos and others, 2018, A memory-bounded, deterministic and terminating semantics for the synchronous programming language Céu
- Sanusi and others, 2024, Assuring Correctness, Testing, and Verification of X-Compiler by Integrating Communicating Stream X-Machine
- Saraf and Dashora, 2013, An Optimal Code Heuristic Approach for Compiler Optimization using Graph Coloring Technique
- Sato, 2021, Proof Assistant and Type Theory
- Sato and others, 2019, Combining higher-order model checking with refinement type inference
- Satya Teja Muddada, 2025, Serverless 2.0 Unlocking Performance and Portability with WebAssembly
- Schäfer and others, 2016, Axiomatic semantics for compiler verification
- Scheidl, 2020, Valent-Blocks Scalable High-Performance Compilation of WebAssembly Bytecode For Embedded Systems
- Schiewe, 2022, Bridging the gap between source code and high-level concepts in static code analysis
- Schkufza and others, 2019, Just-In-Time Compilation for Verilog
- Schlägl and Groβe, 2025, Fast Interpreter-Based Instruction Set Simulation for Virtual Prototypes
- Schlichtkrull and others, 2024, Isabelle-verified correctness of Datalog programs for program analysis
- Schliephake and others, 2011, Design and Implementation of a Runtime System for Parallel Numerical Simulations on Large-Scale Clusters
- Schmale and others, 2022, Backend compiler phases for trapped-ion quantum computers
- Schmeck, 1983, Algebraic semantics of recursive flowchart schemes
- Schmidt, 2000, Abstract interpretation and program modelling
- Schmidt and Völler, 1984, A multi-language compiler system with automatically generated codegenerators
- Schmidt-Schauß and Sabel, 2015, Improvements in a functional core language with call-by-need operational semantics
- Schmitz, 1992, The visual compiler-compiler SIC abstract
- Schnakenbeck and others, 2023, A Control Flow based Static Analysis of GRAFCET using Abstract Interpretation
- Schneider and others, 2006, A Verified Compiler for Synchronous Programs with Local Declarations
- Schoeberl and others, 2010, Worst-case execution time analysis for a Java processor
- Schoenberger and others, 2024, Using Compiler Frameworks for the Evaluation of Hardware Design Choices in Trapped-Ion Quantum Computers
- Schöpp, 2017, Defunctionalisation as modular closure conversion
- Schuele and Schneider, 2004, Abstraction of assembler programs for symbolic worst case execution time analysis
- Schuiki and others, 2020, LLHD a multi-level intermediate representation for hardware description languages
- Schuster and others, 2020, Compiling effect handlers in capability-passing style
- Schuster and others, 2022, A typed continuation-passing translation for lexical effect handlers
- Schwaab and Siek, 2013, Modular type-safety proofs in Agda
- Schwarcz and others, 2024, LOOL Low-Overhead, Optimization-Log-Guided Compiler Fuzzing Registered Report
- Schwarz and others, 2026, TPDE A Fast Adaptable Compiler Back-End Framework
- Scott, 1986, The Interface Between Distributed Operating System and High-Level Programming Language. Revision
- Sculthorpe and others, 2016, A Modular Structural Operational Semantics for Delimited

Continuations
- Seassau and others, 2025, Formal Semantics and Program Logics for a Fragment of OCaml
- Segura, 2026, Algebraic effects for bounded prompt pipelines Lawvere theories, handlers, and outcome traces
- Seidler and others, 2026, Wasm-WCET Worst-Case Execution-Time Analysis of WebAssembly Modules on Updatable Resource-Constrained Embedded Devices
- Sekiyama and Unno, 2023, Temporal Verification with Answer-Effect Modification Dependent Temporal Type-and-Effect System with Delimited Continuations
- Sen and others, 2000, Velocity analysis using pre-stack depth migration and 3D tomogrphy A case study over steeply dipping salt
- Seo and Kim, 2016, Measuring Method of Worst-case Execution Time by Analyzing Relation between Source Code and Executable Code
- Seo and Kim, 2016, Operator-data type pair based execution environments independent worst-case execution time measuring method
- Seo and others, 2007, Goal-directed weakening of abstract interpretation results
- Serafin and others, 2023, Pipestitch An energy-minimal dataflow architecture with lightweight threads
- Serrano, 2021, Of JavaScript AOT compilation performance
- Serrano, 2022, On JavaScript Ahead-of-Time Compilation Performance Keynote
- Seshia and Rakhlin, 2009, Quantitative Analysis of Embedded Software Using Game-Theoretic Learning
- Sethi, 1982, Control flow aspects of semantics directed compiling Summary
- Translation validation for a verified OS kernel
- Seyfer, 1982, Tailoring testing to a specific compiler—experiences
- Shahrokhi and others, 2023, Efficient Query Processing in Python Using Compilation
- Shaikhha and others, 2024, A Tensor Algebra Compiler for Sparse Differentiation
- Shan, 2007, A static simulation of dynamic delimited control
- Shannon 1948, A mathematical theory of communication
- Sharif and others, 2022, COMPAS Compiler-assisted Software-implemented Hardware Fault Tolerance for RISC-V
- Sharma, 2025, Green Threads Human Connection with Nature in Richard Powers' The Overstory
- Sharma and Reddy, 2025, Analysis of FreeRTOS and Contiki Scheduler Performance with Scalable Task Loads on a Uniform Platform
- Sharma and Sharma, 2024, Parameterized Static Analysis for Weak Memory Models
- Sharma and others, 2023, RustSmith Random Differential Compiler Testing for Rust
- Sharp, 1990, Pythia A Parallel Compiler for Delirium
- Sharrad and others, 2018, Delta Debugging Type Errors with a Blackbox Compiler
- Sharygin and Buchatskiy, 2017, Survey of Just-in-Time Query Compilation Methods
- Shcherbakov and Shcherbakova, 2024, Comparative analysis of coroutine functionality in modern programming languages
- Sheinidashtegol and Galloway, 2017, Performance Impact of DDoS Attacks on Three Virtual Machine Hypervisors
- Shen, 2017, Android Security via Static Program Analysis
- Sherman, 2018, Redesigning Soot's data-flow analysis framework for abstract interpretation
- Sheth and Damevski, 2022, Grouping related stack overflow comments for software developer recommendation
- Shi and others, 2008, Virtual machine showdown
- Shih, 1998, An operational semantic approach to continuation style interpreter of logic programs
- Shih and Chen, 1996, Iterative Pre-Stack Depth Migration With Velocity Analysis
- Shiina and others, 2021, Lightweight preemptive user-level threads

- Shimchik and others, 2021, Improving Accuracy and Completeness of Source Code Static Taint Analysis
- Shin and others, 2004, AIRES Automatic Integration of Reusable Embedded Software, Methodologies, Toolkit, and Experiments
- Shirai and others, 2026, Does Programming Language Matter? An Empirical Study of Fuzzing Bug Detection
- Shivers, 1991, The semantics of Scheme control-flow analysis
- Shobaki and others, 2022, Graph transformations for register-pressure-aware instruction scheduling
- Sholihin and Hidayati, 2024, A Forward Chaining Expert System for Personalized Programming Language Selection
- Shoushtary and others, 2024, Memento An Adaptive, Compiler-Assisted Register File Cache for GPUs
- Shukla and others, 2014, A Formal Approach to the Provably Correct Synthesis of Mission Critical Embedded Software for Multi Core Embedded Platforms
- Siambaton and others, 2024, Implementation Draft Programming Oriented Objects in Parking System Application using Language Programming Java
- Sianipar and others, 2018, Virtual Machine Integrity Verification in Crowd-Resourcing Virtual Laboratory
- Silva and others, 2024, Efficient Data Exchange between WebAssembly Modules
- Simon and Kowalewski, 2016, Static analysis of Sequential Function Charts using abstract interpretation
- Simonnet and others, 2024, A Dependent Nominal Physical Type System for Static Analysis of Memory in Low Level Code
- Sinharoy, 1988, EPL - Equational Programming Language Parsing and Dimension Propagation
- Sites, 1979, Machine-independent register allocation
- Retrofitting effect handlers onto OCaml
- Siveroni, 2004, Operational semantics of the Java Card Virtual Machine
- Skarman and others, 2023, Enhancing Compiler-Driven HDL Design with Automatic Waveform Analysis
- Smith and others, 2004, A generalized algorithm for graph-coloring register allocation
- Smith and Zhang, 2024, A Pure Demand Operational Semantics with Applications to Program Analysis
- Snavely, 2011, Test and Evaluation of Architecture-Aware Compiler Environment
- Snyder, 1975, A Portable Compiler for the Language C
- 2021, Socket system in php programming language
- Sohrabi and others, 2018, A Novel Virtual Machine Selection Policy for Virtual Machine Consolidation
- Sokhatskyi and Maslianko, 2018, The systems engineering of consistent pure language with effect type system for certified applications and higher languages
- Soleimani and Balarostaghi, 2016, Seismic image enhancement in post stack depth migration by finite offset CDS stack method
- Soluian and Liushenko, 2025, Research of WebAssembly usage for high-performance code development in web applications
- Somani and Srivastava, 2019, Implementation of SAAS Intranet compiler in PHP
- Son and Lee, 2016, A Study on the Interpreter for the Light-Weighted Virtual Machine on IoT Environments
- Son and others, 2014, A Reversing Technique for Symbol Table Verification on Compiler Constructions
- Son and others, 2017, Design and Implementation of the RSIL to LLVM IR Translator for Verification of the Intermediate Code on IoT Virtual Machine
- Song and Wang, 2021, Monadic Programming Featured Teaching Innovation of Compilation Experiments

- Sorensen and others, 2020, A simulator and compiler framework for agile hardware-software co-design evaluation and exploration
- Souha and others, 2025, Modeling and Automated Code Generation of the Backend of Tourism Applications
- Spampinato and Püschel, 2016, A basic linear algebra compiler for structured matrices
- Spivey, 2017, Faster coroutine pipelines
- Squar and others, 2020, Compiler Assisted Source Transformation of OpenMP Kernels
- Squire and Funkhouser, 2014, “A Bit of Code” How the Stack Overflow Community Creates Quality Postings
- Srinivasan and Reps, 2015, Synthesis of machine code from semantics
- Stanisic and others, 2015, Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures
- Stappert and Altenbernd, 2000, Complete worst-case execution time analysis of straight-line hard real-time programs
- Stata and Abadi, 1998, A type system for Java bytecode subroutines
- Stata and Abadi, 1999, A type system for Java bytecode subroutines
- Steingartner, 2021, Compiler Module of Abstract Machine Code for Formal Semantics Course
- Stepanov and others, 2021, Type-Centric Kotlin Compiler Fuzzing Preserving Test Program Correctness by Preserving Types
- Stepanov and Itsykson, 2022, Backend Bug Finder a platform for effective compiler fuzzing
- Stepanov and Klym, 2025, A QUANTITATIVE ANALYSIS OF WEBASSEMBLY INTEGRATION ARCHITECTURAL PATTERNS, TOOLING, AND PERFORMANCE EVALUATION
- Stiévenart and others, 2022, Static stack-preserving intra-procedural slicing of webassembly binaries
- Stöhr and O’Boyle, 1998, First Fast Sink A compiler algorithm for barrier placement optimisation
- Stolyarov, 2023, STATIC ANALYZER IMPLEMENTATION MODEL FOR THE SOLIDTY PROGRAMMING LANGUAGE
- Stoonkisto and Subhlok, 1997, Coordinating Foreign Modules with a Parallelizing Compiler
- Continuations: A mathematical semantics for handling full jumps
- Strauch, 2023, MRPHS A Verilog RTL to C++ Model Compiler Using Intermediate Representations for Object-oriented Model-driven Prototyping
- Su and others, 2017, Automatic generation of fast BLAS3-GEMM A portable compiler approach
- Subha, 2009, A Modified Linear Scan Register Allocation Algorithm
- Subha, 2010, A register allocation algorithm
- Subramanyan, 2025, CWAMR REIMAGINING A CAPABILITYBASED WEBASSEMBLY RUNTIME VIA CHERI-BASED COMPARTMENTALIZATION
- Suchy and others, 2020, CARAT a case for virtual memory through compiler- and runtime-based address translation
- Suetterlein and others, 2022, Extending an asynchronous runtime system for high throughput applications A case study
- Suganuma and others, 2003, A region-based compilation technique for a Java just-in-time compiler
- Suhendra and Bachtiar, 2016, MIGRATION CODE PADA BACKEND CRIMEZONE DARI PHP KE SCALA
- Sukha, 2015, A Compiler-Runtime Application Binary Interface for Pipe-While Loops
- SulĂr and Poruban, 2017, Exposing Runtime Information through Source Code Annotations
- Sulema and Glinskii, 2020, Semantics and pragmatics of programming language ASAMPL

- Sun and others, 2016, Finding compiler bugs via live code mutation
- Sun and others, 2016, Toward understanding compiler bugs in GCC and LLVM
- Sun and Staron, 2026, Agentic Pipelines in Embedded Software Engineering Emerging Practices and Challenges
- Sun and others, 2025, Archs A WebAssembly Runtime for Cross-host Heterogeneous Computing in Serverless
- Sun and others, 2025, Automating Target Description Processing for Efficient Compiler Backend Development
- Suo and others, 2024, Fuzzing MLIR Compiler Infrastructure via Operation Dependency Analysis
- 2022, Supplemental Material for A Just-in-Time Adaptive Intervention to Enhance Physical Activity in the SMARTFAMILY2.0 Trial
- Surakka and others, 2005, Towards compiler backend optimization for low energy consumption at instruction level
- Surati, 1993, A Parallelizing Compiler Based on Partial Evaluation
- Susca and others, 2022, Worst-Case Execution Time Estimation for Numerical Controllers
- Suwa and others, 2017, Verification of code generators via higher-order model checking
- S.Venkatesan, 2025, TOWARDS AUTONOMOUS CODE OPTIMIZATION A REINFORCEMENT LEARNING FRAMEWORK FOR COMPILER DESIGN
- Swierstra and others, 2016, A Lazy Language Needs a Lazy Type System
- Swillus and Zaidman, 2023, Sentiment overflow in the testing stack Analyzing software testing posts on Stack Overflow
- The F# asynchronous programming model
- 2026, Syntax-Directed Semantics in Programming Language Design
- Szydełko, 2018, BONDS BALANCE SHEET VALUATION IN AMORTISED COST CHOSEN ASPECTS
- Tabassam and Obermaisser, 2017, Class-based query-optimization for minimizing worst-case execution times of diagnostic queries in embedded real-time systems
- Tadepalli and others, 2003, 3D pre-stack depth imaging and well ties A case history of depth imaging in Gulf of Mexico
- Taft, 2024, Sound and precise static analysis using a generalization of static single assignment and value numbering
- Tahat and others, 2019, Scalable Translation Validation of Unverified Legacy OS Code
- Takase and others, 2018, Work-in-Progress Design Concept of a Lightweight Runtime Environment for Robot Software Components Onto Embedded Devices
- Talaat and others, 2025, GrammLLM Grammar-Guided LLM Test Generation for Compiler Validation
- Talamali and others, 2024, Formal Specification and Verification of MQTT Protocol Using CoQ Proof Assistant
- Ţălu, 2025, A Comparative Study of WebAssembly Runtimes Performance Metrics, Integration Challenges, Application Domains, and Security Features
- Tamura and others, 2025, Bringing Together Cross-ISA Checkpoint/Restoration and AOT Compilation of WebAssembly Programs
- Tan, 2009, The worst-case execution time tool challenge 2006
- Tan and others, 2025, The Burden of Proof Automated Tooling for Rapid Iteration on Large Mechanised Proofs
- Tan and others, 2016, A new verified compiler backend for CakeML
- Tan and others, 2015, A verified type system for CakeML
- Tang and others, 2019, Compiler testing a systematic literature analysis
- Tang and others, 2010, Balanced Bipartite Graph Based Register Allocation for Network Processors in Mobile and Wireless Networks
- Tang and others, 2025, CTDip a diversity-guided test program synthesis approach for boosting compiler bug detection

- Tant and others, 2023, Software Compilation Using FPGA Hardware Register Allocation
- Tao and others, 2010, An Automatic Testing Approach for Compiler Based on Metamorphic Testing Technique
- TARAU, 2011, The BinProlog experience Architecture and implementation choices for continuation passing Prolog and first-class logic engines
- Tardieu, 2014, Session details Compiler optimizations
- Tarjan, 1985, Amortized Computational Complexity
- Tatsuoka and Kaneko, 2018, Wire congestion aware high level synthesis flow with source code compiler
- Tavares and others, 2011, Decoupled graph-coloring register allocation with hierarchical aliasing
- Tempel and others, 2021, Towards Reliable Spatial Memory Safety for Embedded Software by Combining Checked C with Concolic Testing
- ten Hagen and others, 1996, Codesign of a parallel architecture and an optimizing compiler backend SIN rete processing as a case study
- Terao, 2018, Lazy Abstraction for Higher-Order Program Verification
- Terci, 2023, A Learning-Based Coloring Algorithm for Register Allocation Problem
- Thangamani and others, 2023, Lifting Code Generation of Cardiac Physiology Simulation to Novel Compiler Technology
- 2015, The Deasibility and Properties of Dividing Virtual Machine Resources using the Virtual Machine Cluster as the Unit in Cloud Computing
- 2007, The Role of Abstract Interpretation in Formal Methods
- 2017, The Utilization of Cloud Computing as Virtual Machine
- Thielecke, 2003, From control effects to typed continuation passing
- Thielecke, 2012, Functional semantics of parsing actions, and left recursion elimination as continuation passing
- Thiselton and Treude, 2019, Enhancing Python Compiler Error Messages via Stack
- Thomas and others, 2024, Analyzing Data Flow and Control Flow of Multicore Software A Solution for Efficient Worst-Case Execution Time Analysis
- Thorpe and others, 2022, Verification of Cyber Emulation Experiments Through Virtual Machine and Host Metrics
- Thorpe and others, 2022, WiP Verification of Cyber Emulation Experiments Through Virtual Machine and Host Metrics
- Tian and others, 2016, Optimizing GPU Register Usage Extensions to OpenACC and Compiler Optimizations
- Tian and others, 2017, LLVM Compiler Implementation for Explicit Parallelization and SIMD Vectorization
- Tian and others, 2024, Differential testing solidity compiler through deep contract manipulation and mutation
- Tillet and others, 2019, Triton an intermediate language and compiler for tiled neural network computations
- Tine and others, 2019, POSTER Tango An Optimizing Compiler for Just-In-Time RTL Simulation
- Tinnerholm and others, 2019, Towards introducing just-in-time compilation in a Modelica compiler
- Tirichine and others, 2026, A Reinforcement Learning Environment for Automatic Code Optimization in the MLIR Compiler
- Titzer, 2024, Whose Baseline Compiler is it Anyway?
- Titzer, 2025, WebAssembly How Low Can a Bytecode Go?
- Todoran, 2020, Metric Semantics for Concurrent Languages Designed in Continuation-Passing Style
- Todoran and Ciobanu, 2025, Metric Continuation-Passing Semantics for Multiparty Interactions
- Todoran and Papaspyrou, 2017, Concurrency Semantics in Continuation-Passing Style

- Tohid and others, 2018, Asynchronous Execution of Python Code on Task-Based Runtime Systems
- TOKUMORI and others, 2007, Development of Metadata Generation Support System Using Compiler Compiler
- Tokumoto and others, 2016, MuVM Higher Order Mutation Analysis Virtual Machine for C
- Tolpin and others, 2016, Design and Implementation of Probabilistic Programming Language Anglican
- Touzeau, 1984, A Fortran compiler for the FPS-164 scientific computer
- Tran and others, 2023, Transport Layer Security 1.0 handshake protocol formal verification case study How to use a proof script generator for existing large proof scores
- Trifanov and Schrijvers, 2026, Staging Effect Handlers for Modular Search
- Trout, 1967, A compiler—compiler system
- Truong and others, 2016, Latte a language, compiler, and runtime for elegant and efficient deep neural networks
- Tsuyama and others, 2024, An Intrinsically Typed Compiler for Algebraic Effect Handlers
- Tu and others, 2022, Remgen Remanufacturing a Random Program Generator for Compiler Testing
- Tu and others, 2023, Detecting C++ Compiler Front-End Bugs via Grammar Mutation and Differential Testing
- Tuck, 1988, An Optimally Portable SIMD Single-Instruction Multiple-Data Programming Language
- Turcotte and Vitek, 2019, Towards a Type System for R
- 2009, Tutorial on SSA-Based Register Allocation
- Uddin and others, 2020, Mining API usage scenarios from stack overflow
- Ueno and Ohori, 2022, Concurrent and parallel garbage collection for lightweight threads on multicore processors
- Uh, 2003, Session details Compiler optimizations
- Upadhyaya and Rajan, 2015, Effectively mapping linguistic abstractions for message-passing concurrency to threads on the Java virtual machine
- Urban and others, 2025, Static analysis by abstract interpretation against data leakage in machine learning
- Utting and others, 2023, Differential Testing of a Verification Framework for Compiler Optimizations Case Study
- V and others, 2019, A WEIGHTED ENSEMBLE OF AUTOMATIC ALGORITHMS FOR VIRTUAL MACHINE PERFORMANCE PREDICTION IN CLOUD
- Valiron, 2022, Semantics of quantum programming languages Classical control, quantum control
- VANDENBROUCKE and SCHRIJVERS, 2023, Disjunctive Delimited Control
- Vandercammen and others, 2018, A flexible framework for studying trace-based just-in-time compilation
- van der Hoeven and Lecerf, 2021, Amortized Bivariate Multi-point Evaluation
- van Deursen, 1999, Modern Compiler Implementation in Java
- van Rooij and Krebbers, 2025, Affect An Affine Type and Effect System
- van Tonder and Le Goues, 2020, Tailoring programs for static analysis via program transformation
- Vasilyev and Mutilin, 2020, Predicate Extension of Symbolic Memory Graphs for the Analysis of Memory Safety Correctness
- Vegdahl and Pleban, 1989, The runtime environment for Scheme, a Scheme implementation on the 88000
- Veit and Böcskei, 2024, IFRS 9 Classification Aspects Measurement of Sustainability-Linked Loans at Amortised Cost or Fair Value
- Velazquez-Rodriguez and others, 2022, Uncovering Library Features from API Usage on Stack Overflow

- Venkanna and others, 2018, PSO based optimization of worst-case execution time for ASIP application
- Venkanna and Rao, 2018, Static Worst-Case Execution Time Optimization using DPSO for ASIP Architecture
- VenkataKeerthy and others, 2023, RL4ReAl Reinforcement Learning for Register Allocation
- Ventovaara and others, 2020, Worst-case Execution Time Estimation of Legacy Vehicular Embedded Functions An Industrial Case Study
- Vepuri and Jiang, 2023, Performance Analysis of Virtual Machine Monitoring System
- Verdejo and Martí-Oliet, 2006, Executable structural operational semantics in Maude
- Verma and Bakshi, 2015, Chronological Advancement in Compiler Design A Review
- Verma and others, 2023, Array Bytecode Support in MicroJIT
- 2017, Virtual Machine Consolidation using Load Balancing algorithm in Cloud Data Center
- VISTA RESEARCH CORP TUCSON AZ, 1994, Ada Compiler Validation Summary Report Certificate Number 940223W1. 11338 Green Hills Software, Inc. Green Hills Optimizing Ada Compiler, 1.8.7 SPARCstation 10 under SunOS, Release 4.1.3
- VISTA RESEARCH CORP TUCSON AZ, 1994, Ada Compiler Validation Summary Report Certificate Number 940305W1. 11335 TLD Systems, Ltd. TLD Comanche VAX/1960 Ada Compiler System, Version 4.1.1 VAX Cluster under VMS 5.5 = Tronix JIAWG Execution Vehicle i960MX under TLD Real Time Executive, Version 4.1.1
- Vizcaino and others, 2022, Acceleration with long vector architectures Implementation and evaluation of the FFT kernel on NEC SX-Aurora and RISC-V vector extension
- Voigt and others, 2025, Dynamic Wind for Effect Handlers
- Capriccio: scalable threads for internet services
- Vos and others, 2023, Oraqle A Depth-Aware Secure Computation Compiler
- Vraný and Shingarov, 2024, Tinyrossa A Compiler Framework for Vertical, Verified Construction of Smalltalk VMs
- V. Samuel Blessed Nayagam and Shajin Nargunam, 2018, Secure Data Verification and Virtual Machine Monitoring
- Vu, 2008, Denotational semantics for thread algebra
- Wagle and others, 2019, Runtime Adaptive Task Inlining on Asynchronous Multitasking Runtime Systems
- Wagstaff and others, 2008, Automatic Code Generation for Instrument Flight Software
- Wahyudi and Miswanto, 2020, VIRTUAL MACHINE FORENSIC ANALYSIS and RECOVERY VMFAR SEBAGAI FRAMEWORK UNTUK ANALISIS BUKTI DIGITAL PADA VIRTUAL MACHINE
- Waites and others, 2018, A Genetic Circuit Compiler Generating Combinatorial Genetic Circuits with Web Semantics and Inference
- Wall, 1986, Global register allocation at link time
- Wall, 1988, Register windows vs. register allocation
- Wall, 2004, Register windows vs. register allocation
- Wambua, 2025, Topics, Trends, and Sentiments in Software Testing An Analysis of Developers' Engagement on Stack Overflow
- Wang, 2017, Research on the Execution Time Analysis Technology of the Worst Case System in Real Time System
- Wang, 2019, Web Crawler Scheduler Based on Coroutine
- Wang, 2021, Can "micro VM" become the next generation computing platform? Performance comparison between light weight Virtual Machine, container, and traditional Virtual Machine
- Wang, 2021, Helper function inlining in dynamic binary translation
- Wang, 2024, Worst-Case Blocking Time Optimization in WCRT Analysis for vMPCP on Multi-Core Virtual Machines
- Wang and others, 2023, MLIRSmith Random Program Generation for Fuzzing MLIR Com-

piler Infrastructure
- Wang and Dahl, 1971, Coroutine sequencing in a block structured environment
- Wang and others, 2015, WCET-Aware Energy-Efficient Data Allocation on Scratchpad Memory for Real-Time Embedded Systems
- Wang and others, 2025, Research on Compiler Fuzzing Based on Syntax-semantics Dual-Dimensional Classification
- Wang and Huang, 2022, SGPM A coroutine framework for transaction processing
- Wang and huang, 2022, Sgpm A Coroutine Scheduling Model for Wound-Wait Concurrency Control
- Wang and others, 2014, Register allocation for hybrid register architecture in nonvolatile processors
- Wang and Jung, 2024, Rustlantis Randomized Differential Testing of the Rust Compiler
- Wang and others, 2019, Tracking runtime concurrent dependences in java threads using thread control profiling
- Wang and others, 2023, A General-Purpose Compiler Design for Instruction-Based AI Accelerator Implementation
- Wang and others, 2022, Unleashing Coveraged-Based Fuzzing Through Comprehensive, Efficient, and Faithful Exploitable-Bug Exposing
- Wang and others, 2026, Random test generators demystified Differences and potential for compiler reliability
- Wang and others, 2010, Stack Bound Inference for Abstract Java Bytecode
- Wang and others, 2020, RSDS Getting System Call Whitelist for Container Through Dynamic and Static Analysis
- Wang and others, 2024, K-RAPID A Formal Executable Semantics of the RAPID Robot Programming Language
- Wang and others, 2009, Reducing Code Size by Graph Coloring Register Allocation and Assignment Algorithm for Mixed-Width ISA Processor
- Wang and others, 2023, An Automated Verification Framework for HalideIR-Based Compiler Transformations
- Wang and Xie, 2024, Enhancing Translation Validation of Compiler Transformations with Large Language Models
- Wang and others, 2017, Faster mutation analysis via equivalence modulo states
- Wang and Yang, 2014, Applied Technology in Front-End Implementation of Tiger Compiler Using Hacs Language
- Wang and others, 2019, Reg An Ultra-Lightweight Container That Maximizes Memory Sharing and Minimizes the Runtime Environment
- Wang and others, 2023, A Comprehensive Study of WebAssembly Runtime Bugs
- Wang and Zhu, 2011, Animating the Approach of Deriving Operational Semantics from Algebraic Semantics for Web Services
- WanXin and others, 2022, CUDA Acceleration of Worst-Case Execution Time Analysis Based On Model Checking
- Watanabe and others, 2020, Design and Preliminary Evaluation of OpenACC Compiler for FPGA with OpenCL and Stream Processing DSL
- Watt, 2025, Concurrency in WebAssembly
- -, 2021, WebAssembly for High-Performance Web Applications A Study on Execution Speed and Efficiency
- Weber and Fischer, 2020, Process-Based Simulation with Stackless Coroutines
- Weber and others, 2022, A closer look at process-based simulation with stackless coroutines
- Wei and others, 2023, Compiling Parallel Symbolic Execution with Continuations
- Weiner and Ramakrishman, 1988, A piggy-back compiler for Prolog
- Welch and others, 2017, Formalization IDEs Integrated with a Verifying Compiler
- Wenes and others, 1994, A Practical implementation of a 3D pre-stack depth migration algorithm

- Wenger and others, 2016, A Programming Language and System for Heterogeneous Cloud of Things
- Wheeler and others, 2011, Video subset selection for measurement based Worst Case Execution Time analysis
- Whiteside and others, 2012, Directional Imaging Stack DIS for Shot Based Pre-stack Depth Migrations
- Wibowo and others, 2015, Unit test code generator for lua programming language
- The worst-case execution-time problem: overview of methods and survey of tools
- Williams and Bulmer, 1978, Use of a formal notation for static semantics in compiler design
- Williams and Elliott, 2025, Libfork Portable Continuation-Stealing With Stackless Coroutines
- Williams and Roger, 2009, Test generation strategies to measure worst-case execution time
- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . ANSI
- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . ANSI, BACKUP, TAR
- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . BACKUP
- Wilson, 1989, Ada Compiler Validation Capability ACVC Version 1.11 Field-Test Release . TAR
- Wilson, 1990, Ada/Ed Compiler, Version 1.10 UNIX
- Wilson, 1990, Ada/Ed Compiler, Version 1.10 VAX
- Wimmer and others, 2017, One compiler deoptimization to optimized code
- Windsor and others, 2021, C4 the C compiler concurrency checker
- Woronow, 1989, Correction for a “FORTRAN program for generation of multivariate normally distributed random variables”
- 2016, WORST CASE EXECUTION TIME CALCULATION OF PARALLEL EMBEDDED REAL-TIME SOFTWARE
- Worthington, 2021, Reflections on a decade of MoarVM, a runtime for the Raku programming language keynote
- Wu, 2023, PyTorch 2.0 The Journey to Bringing Compiler Technologies to the Core of PyTorch Keynote
- Wu and others, 2021, Effective Register Allocation for Configurable VLIW Crypto-Processor
- Wu and others, 2026, Fluctuation-guided adaptive random compiler for Hamiltonian simulation
- Wu and others, 2025, WBSan WebAssembly Bug Detection for Sanitization and Binary-Only Fuzzing
- Wu and Li, 2007, Extending Traditional Graph-Coloring Register Allocation Exploiting Meta-heuristics for Embedded Systems
- Wu and others, 2026, Reconfigurable Computing Challenge FPGA-Based WebAssembly Stack Co-Processor
- Wu and others, 2014, Effect handlers in scope
- Wu and others, 2026, Designing quantum chemistry algorithms with just-in-time compilation
- Wu and others, 2023, Boosting Compiler Testing via Eliminating Test Programs with Long-Execution-Time
- Wu and others, 2017, Two Schemes to Improve the Implementation of the Aggregation Based Algebraic Multigrid Preconditioner
- Wu and Zhang, 2012, A Model Checking Based Approach to Bounding Worst-Case Execution Time for Multicore Processors
- Wu and Zhang, 2013, Reducing worst-case execution time of hybrid SPM-caches

- Wu and Zhang, 2018, Cache-Aware SPM Allocation to Reduce Worst-Case Execution Time for Hybrid SPM-Caches
- Wu and others, 2025, Compiler Optimization Testing Based on Optimization-Guided Equivalence Transformations
- Würthinger and others, 2017, Practical partial evaluation for high-performance dynamic language runtimes
- Xiang and others, 2026, LoopHint A Compiler-Assisted Loop Branch Predictor for Embedded DSPs
- Xiao, 2021, Transformation System of two Similar Syntax Programs Based on the Compiler Principle
- Xiao and others, 2023, Read-Write Dependency Aware Register Allocation
- Xia and DiVito, 2005, Software Certification for Temporal Properties With Affordable Tool Qualification
- Xie and others, 2020, Effect handlers, evidently
- Xie and others, 2022, First-class names for effect handlers
- Xie and others, 2024, Parallel Algebraic Effect Handlers
- Xie and Leijen, 2020, Effect handlers in Haskell, evidently
- Xie and Leijen, 2021, Generalized evidence passing for effect handlers efficient compilation of effect handlers to C
- Xie and others, 2025, Kitten A Simple Yet Effective Baseline for Evaluating LLM-Based Compiler Testing Techniques
- Xu and Gregg, 2015, An Efficient Vectorization Approach to Nested Thread-level Parallelism for CUDA GPUs
- Xu and Kjolstad, 2021, Copy-and-patch compilation a fast compilation algorithm for high-level languages and bytecode
- Xu and others, 2020, DSmith Compiler Fuzzing through Generative Deep Learning Model with Attention
- Xu and Zhang, 2014, Formal verification of software safety criteria using Event-B
- Xu and others, 2024, SWAT4J Generating System Call Allowlist for Java Container Attack Surface Reduction
- Xu and others, 2016, CAOPLE A Programming Language for Microservices SaaS
- Xue and Bogdan, 2016, Scalable and realistic benchmark synthesis for efficient NoC performance evaluation
- Yadav and others, 2022, DISTAL the distributed tensor algebra compiler
- Yallop and others, 2018, A modular foreign function interface
- Yamamoto and others, 2016, Lightweight Ruby Framework for Improving Embedded Software Efficiency
- Yamane and others, 2020, Verification Method of Safety Properties of Embedded Assembly Program by Combining SMT-Based Bounded Model Checking and Reduction of Interrupt Handler Executions
- Yamato, 2015, Automatic verification for plural virtual machines patches
- Yan and Zhang, 2008, Analyzing the worst-case execution time for instruction caches with prefetching
- Yaneva and others, 2017, Compiler-assisted test acceleration on GPUs for embedded software
- Finding and understanding bugs in C compilers
- Yang, 2018, Formal Process Virtual Machine for Smart Contracts Verification
- Yang and others, 2025, Formal Verification of a Custom Compiler for a Fully Homomorphic Encryption Accelerator
- Yang and others, 2015, Towards a verified compiler prototype for the synchronous language SIGNAL
- Yang and others, 2024, WhiteFox White-Box Compiler Fuzzing Empowered by Large Language Models
- Yang and Duan, 2008, Operational semantics of Framed Tempura

- Yang and Ruiz Varela, 2015, Qualifying non-volatile register files for embedded systems through compiler-directed write minimization and balancing
- Yang and others, 2022, Simulink Model Static Analysis Results based on Abstract Interpretation
- Ye and Delaware, 2019, A verified protocol buffer compiler
- Ye and others, 2023, A Generative and Mutational Approach for Synthesizing Bug-Exposing Test Cases to Guide Compiler Fuzzing
- Ye and others, 2025, BazzAFL Moving Fuzzing Campaigns Towards Bugs via Grouping Bug-Oriented Seeds
- Yesua and others, 2019, Keamanan Penggunaan Propofol Auto-Coinduction Dibandingkan Dengan Midazolam Coinduction Berdasarkan Perubahan Hemodinamik Pada Induksi Anestesi Pasien Yang Dilakukan General Anestesi
- Yi, 2011, Automated programmable control and parameterization of compiler optimizations
- Yi and others, 2026, Understanding and Finding JIT Compiler Performance Bugs
- Yi and Lee, 2018, An Educational System Design to Support Learning Transfer from Block-based Programming Language to Text-based Programming Language
- Yildiz and others, 2019, Software UART A Use Case for VSCPU Worst-Case Execution Time Analyzer
- Yilmazer-Metin, 2021, sRSP An efficient and scalable implementation of remote scope promotion
- Yim and others, 2019, Implementation of Targeted Advertisement Services on ATSC 3.0 Runtime Environment
- Yin and others, 2020, The Implementation of Simple Smart Contract Language and Its Compiler Based on Ethereum Platform
- Yoo and Kim, 2017, Coroutine based Algorithms for reducing Memory overhead in Virtual Reality
- Yoshikawa and others, 2003, Random program generator for Java JIT compiler test system
- Yoshioka and others, 2024, Abstracting Effect Systems for Algebraic Effect Handlers
- You and Chen, 2015, Vector-aware register allocation for GPU shader processors
- You and Lu, 2012, A markup language for java bytecode
- YU, 2008, Design and implementation of NC code compiler based on ANTLR
- Yu, 2009, Scalable Services Orchestration with Continuation-Passing Messaging
- Yu, 2023, Reasoning about MLIR Semantics through Effects and Handlers
- Yu and Cohen, 2015, Guided Test Generation for Finding Worst-Case Stack Usage in Embedded Systems
- Yu and Haque, 2011, Decentralised web-services orchestration with continuation-passing messaging
- Yu and Yang, 2007, Continuation-passing enactment of distributed recoverable workflows
- Yu and others, 2023, Efficient Generation of Floating-Point Inputs for Compiler-Induced Variability
- Yvon and Feeley, 2021, A small scheme VM, compiler, and REPL in 4k
- Zaitsev and Guliaiev, 2011, Stack E6 and Its Implementation within Linux Kernel
- Zakowski and others, 2020, An equational theory for weak bisimulation via generalized parameterized coinduction
- Zaks and Pnueli, 2008, Program analysis for compiler validation
- Zanardini, 2008, The Semantics of Abstract Program Slicing
- Zapanov, 2025, Foreign Function Interface for Managed Runtime Systems with Lightweight Threading
- Zaytsev, 2018, An industrial case study in compiler testing tool demo
- Zaytsev, 2020, Modelling of Language Syntax and Semantics The Case of the Assembler Compiler

- Zelenova, 2025, Static Memory Layout for Real-Time Operating Systems
- Zelkowitz, 1975, Third generation compiler design
- Zendra and Colnet, 2001, Coping with aliasing in the GNU Eiffel Compiler implementation
- Zeng and others, 2026, TypeNFuzz Dynamic Type-aware Object Dependence Graph-Guided Fuzzing for JavaScript Library Bug Discovery
- Zhang, 2016, Selection and Improvement of Computer Programming Language
- Zhang, 2018, Compiler Practice System Integrated with Real Open Source Compiler
- Zhang, 2025, Comparative Implementation of Binary Tree and Recursive Backtracking Maze Generation Algorithms in OCaml
- Zhang and others, 2022, Cape compiler-aided program transformation for HTM-based cache side-channel defense
- Zhang and others, 1993, Pipelined processors and worst case execution times
- Zhang and others, 2023, Characterizing and Detecting WebAssembly Runtime Bugs
- Zhang and Deng, 2018, A Depth Variant Seismic Wavelets Extraction Method for Inversion of Post-Stack Depth Domain Seismic Data
- Zhang and others, 2026, GeoIR-Compiler A Geospatial Intermediate Representation and Compilation Framework for Chinese Urban Spatial Question Answering
- Zhang and others, 2026, Transformation-Recipe-Based FPGA Synthesis Compiler Testing
- Zhang and Koutsoukos, 2015, Improving the Precision of Abstract Interpretation Based Cache Persistence Analysis
- Zhang and others, 2019, SNC A Cloud Service Platform for Symbolic-Numeric Computation Using Just-In-Time Compilation
- Zhang and Meng, 2020, Design and Implementation of Multi-core DSP Parallel Compiler Based on Otsu Method
- Zhang and Myers, 2019, Abstraction-safe effect handlers via tunneling
- Zhang and others, 2011, A Case Study of Pre-stack Depth Migration Application over a Salt Dome Area
- Zhang and others, 2025, Refactoring for Java-Structured Concurrency
- Zhang and others, 2017, Skeletal program enumeration for rigorous compiler testing
- Zhang and others, 2016, Compiler Transformation to Generate Hybrid Sparse Computations
- Zhang and others, 2017, Weak Memory Models Balancing Definitional Simplicity and Implementation Flexibility
- Zhang and others, 2013, Register Allocation by Incremental Graph Colouring for Clustered VLIW Processors
- Zhang and others, 2026, LLM-Powered Silent Bug Fuzzing in Deep Learning Libraries via Versatile and Controlled Bug Transfer
- Zhang and others, 2023, Compiler Technologies in Deep Learning Co-Design A Survey
- Zhang and Yan, 2009, Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches
- Zhang and Yin, 2021, A Virtual Machine Placement Strategy Based on Virtual Machine Selection and Integration
- Zhang and others, 2024, Introducing Compiler Semantics into Large Language Models as Programming Language Translators A Case Study of C to x86 Assembly
- Zhang and others, 2026, LEGO-compiler enhancing neural compilation through translation composability
- Zhang and others, 2020, A Dynamic Instruction Cache Locking Approach for Minimizing Worst Case Execution Time of a Single Task
- Zhao and others, 2025, Thread-sensitive fuzzing for concurrency bug detection
- Zhao and others, 2024, Design and Implementation of the MTP Compiler
- Zhao and others, 2026, QuaMap A Multi-Backend Benchmark Dataset for Quantum Circuit Mapping and Learning-Based Compiler Evaluation
- Zhao and others, 2024, COPS A Coroutine-Based Priority Scheduling Framework Per-

ceived by the Operating System
- Zhao and others, 2021, Similarity-Aware Architecture/Compiler Co-Designed Context-Reduction Framework for Modulo-Scheduled CGRA
- Zhao and others, 2024, Wapplique Testing WebAssembly Runtime via Execution Context-Aware Bytecode Mutation
- Zhao and others, 2021, Hot question prediction in Stack Overflow
- Zheng and Xia, 2019, Exploring mixed integer programming reformulations for virtual machine placement with disk anti-colocation constraints
- Zhong, 2022, Enriching Compiler Testing with Real Program from Bug Report
- Zhong and others, 2023, Neural Network Guided Evolutionary Fuzzing for Finding Traffic Violations of Autonomous Vehicles
- Zhong and others, 2026, A Multi-Modal Retrieval-Augmented Framework for Compiler Backend Generation with LLMs
- Zhong and others, 2024, ComBack A Versatile Dataset for Enhancing Compiler Backend Development Efficiency
- Zhong and others, 2026, Towards Fully Automated Compiler Backend Generation with Multi-Agent Systems How Far Are We?
- Zhong and others, 2026, BePilot An AI Programming Assistant for Compiler Backend Development
- Zhou, 1996, Parameter passing and control stack management in Prolog implementation revisited
- Zhou and others, 2024, C-CORE Clustering by Code Representation to Prioritize Test Cases in Compiler Testing
- Zhou and others, 2016, Worst case response time and schedulability analysis for real-time software transactional memory-lazy conflict detection STM-LCD
- Zhou and Mu, 2016, Representative Virtual Machine Templates An optimized virtual machine templates management mechanism for an Cloud system based on K-medoids Clustering
- Zhou and others, 2020, Design and Implementation of Coroutine Scheduling System on SW26010
- Zhou and Xue, 2016, A Compiler Approach for Exploiting Partial SIMD Parallelism
- Zhu, 2001, Denotational semantics of programming languages and compiler generation in PowerEpsilon
- Zhu and others, 2007, Algebraic Approach to Operational Semantics and Observation-Oriented Semantics for a Timed Shared-Variable Language with Probability
- Zhu and others, 2008, From algebraic semantics to denotational semantics for Verilog
- Zhu and others, 2019, Learning to Restrict Test Range for Compiler Test
- Zhu and others, 2025, Emerging Compiler Testing Based on Test Case Reuse
- Zhu and Xie, 2025, A Domain-Specific Compiler for Embedded DSP Development Based on Component Code Generation Architecture and Correctness
- Zhu and others, 2009, Animating the Link Between Operational Semantics and Algebraic Semantics for a Probabilistic Timed Shared-Variable Language
- Zhu and others, 2012, Linking operational semantics and algebraic semantics for a probabilistic timed shared-variable language
- Zhu and others, 2009, Locality-Based Normal Form Approach to Linking Algebraic Semantics and Operational Semantics for an Event-Driven System-Level Language
- Zolda and others, 2011, Context-Sensitive Measurement-Based Worst-Case Execution Time Estimation
- Zolduoarrati and others, 2025, A cross-continental analysis of how regional cues shape top stack overflow contributors
- Zou and others, 2022, Buddy Stacks Protecting Return Addresses with Efficient Thread-Local Storage and Runtime Re-Randomization
- Zou and others, 2017, Towards comprehending the non-functional requirements through Developers' eyes An exploration of Stack Overflow using topic analysis

- Zubarev, 2017, TYPE ANALYSIS FOR THE PREDICATE PROGRAMMING LANGUAGE
- Zuepke and Kaiser, 2019, Deterministic Futexes Addressing WCET and Bounded Interference Concerns
- Zúñiga and Bel-Enguix, 2020, Coinductive Natural Semantics for Compiler Verification in Coq
- Zwanziger, 2019, Dependently-Typed Montague Semantics in the Proof Assistant Agda-flat
- Zyuzin and Nanevski, 2021, Contextual modal types for algebraic effects and handlers