Our Approach

Data structure + SMR impl $\models$ Property

SMR impl $\models$ SMR spec

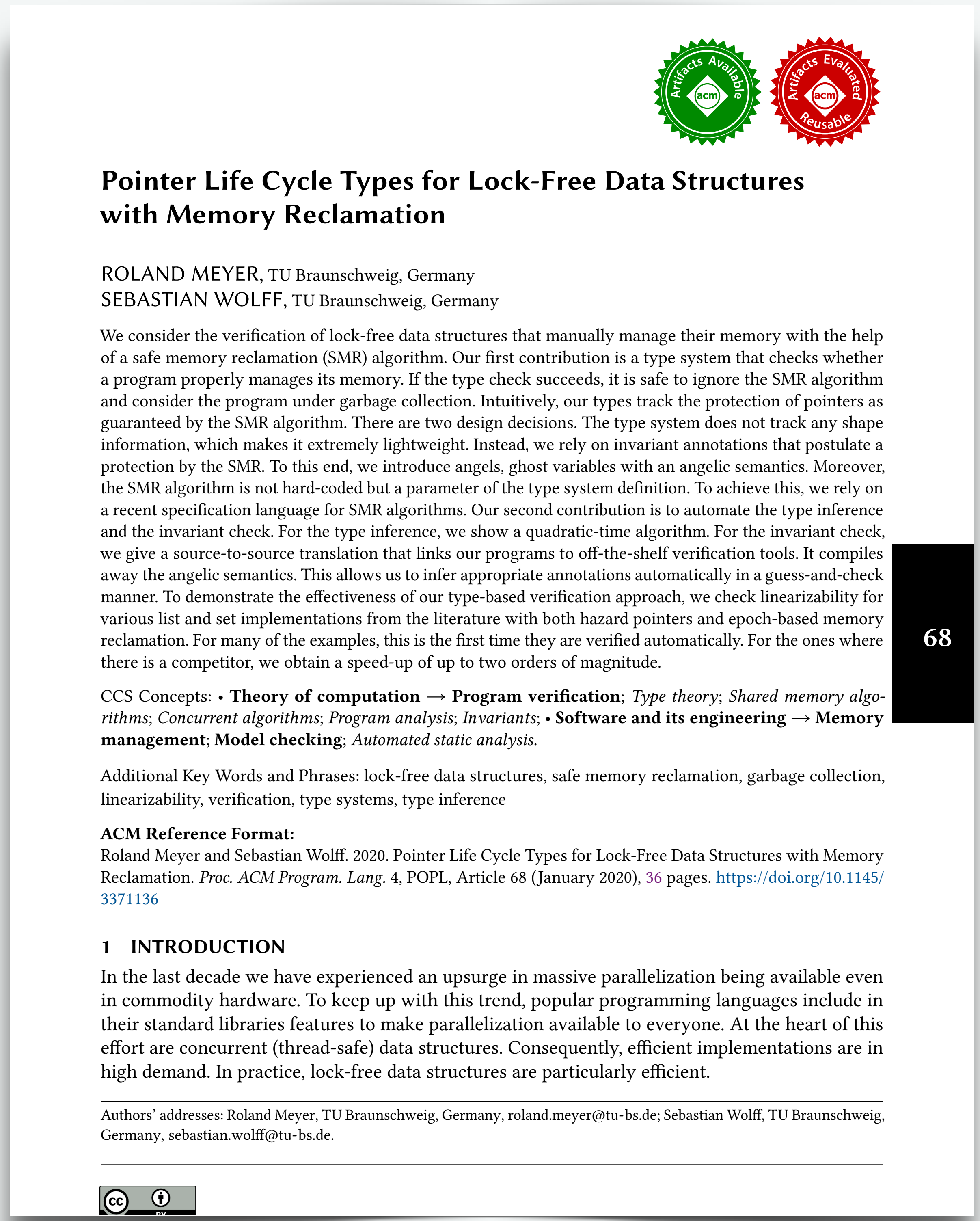
Data structure $+$ **SMR spec** $\models$ **ABA Freedom**

Data structure $+$ **Garbage Collection** $\models$ **Property**

Annot. Data structure $+$ SMR spec $\vdash$ Pointer Race Freedom

[POPL'20]

with Roland Meyer



Pointer Life Cycle Types for Lock-Free Data Structures with Memory Reclamation

ROLAND MEYER, TU Braunschweig, Germany
SEBASTIAN WOLFF, TU Braunschweig, Germany

We consider the verification of lock-free data structures that manually manage their memory with the help of a safe memory reclamation (SMR) algorithm. Our first contribution is a type system that checks whether a program properly manages its memory. If the type check succeeds, it is safe to ignore the SMR algorithm and consider the program under garbage collection. Intuitively, our types track the protection of pointers as guaranteed by the SMR algorithm. There are two design decisions. The type system does not track any shape information, which makes it extremely lightweight. Instead, we rely on invariant annotations that postulate a protection by the SMR. To this end, we introduce angels, ghost variables with an angelic semantics. Moreover, the SMR algorithm is not hard-coded but a parameter of the type system definition. To achieve this, we rely on a recent specification language for SMR algorithms. Our second contribution is to automate the type inference and the invariant check. For the type inference, we show a quadratic-time algorithm. For the invariant check, we give a source-to-source translation that links our programs to off-the-shelf verification tools. It compiles away the angelic semantics. This allows us to infer appropriate annotations automatically in a guess-and-check manner. To demonstrate the effectiveness of our type-based verification approach, we check linearizability for various list and set implementations from the literature with both hazard pointers and epoch-based memory reclamation. For many of the examples, this is the first time they are verified automatically. For the ones where there is a competitor, we obtain a speed-up of up to two orders of magnitude.

CCS Concepts: • **Theory of computation** → **Program verification**; *Type theory*; *Shared memory algorithms*; *Concurrent algorithms*; *Program analysis*; *Invariants*; • **Software and its engineering** → **Memory management**; **Model checking**; *Automated static analysis.*

Additional Key Words and Phrases: lock-free data structures, safe memory reclamation, garbage collection, linearizability, verification, type systems, type inference

ACM Reference Format:
Roland Meyer and Sebastian Wolff. 2020. Pointer Life Cycle Types for Lock-Free Data Structures with Memory Reclamation. *Proc. ACM Program. Lang.* 4, POPL, Article 68 (January 2020), 36 pages. https://doi.org/10.1145/3371136

1 INTRODUCTION

In the last decade we have experienced an upsurge in massive parallelization being available even in commodity hardware. To keep up with this trend, popular programming languages include in their standard libraries features to make parallelization available to everyone. At the heart of this effort are concurrent (thread-safe) data structures. Consequently, efficient implementations are in high demand. In practice, lock-free data structures are particularly efficient.

Authors' addresses: Roland Meyer, TU Braunschweig, Germany, roland.meyer@tu-bs.de; Sebastian Wolff, TU Braunschweig, Germany, sebastian.wolff@tu-bs.de.

Data structure $+$ **Garbage Collection** $\models$ **Annotations**

Our Approach

SMR impl $\models$ SMR spec

Data structure + Garbage Collection $\models$ Property

~~Data structure + SMR spec $\models$ ABA Freedom~~

Annot. Data structure + SMR spec $\vdash$ Pointer Race Freedom

Data structure + Garbage Collection $\models$ Annotations

Data structure + SMR impl $\models$ Property

[POPL’20]

with Roland Meyer

Pointer Life Cycle Types for Lock-Free Data Structures with Memory Reclamation

ROLAND MEYER, TU Braunschweig, Germany
SEBASTIAN WOLFF, TU Braunschweig, Germany

We consider the verification of lock-free data structures that manually manage their memory with the help of a safe memory reclamation (SMR) algorithm. Our first contribution is a type system that checks whether a program properly manages its memory. If the type check succeeds, it is safe to ignore the SMR algorithm and consider the program under garbage collection. Intuitively, our types track the protection of pointers as guaranteed by the SMR algorithm. There are two design decisions. The type system does not track any shape information, which makes it extremely lightweight. Instead, we rely on invariant annotations that postulate a protection by the SMR. To this end, we introduce angels, ghost variables with an angelic semantics. Moreover, the SMR algorithm is not hard-coded but a parameter of the type system definition. To achieve this, we rely on a recent specification language for SMR algorithms. Our second contribution is to automate the type inference and the invariant check. For the type inference, we show a quadratic-time algorithm. For the invariant check, we give a source-to-source translation that links our programs to off-the-shelf verification tools. It compiles away the angelic semantics. This allows us to infer appropriate annotations automatically in a guess-and-check manner. To demonstrate the effectiveness of our type-based verification approach, we check linearizability for various list and set implementations from the literature with both hazard pointers and epoch-based memory reclamation. For many of the examples, this is the first time they are verified automatically. For the ones where there is a competitor, we obtain a speed-up of up to two orders of magnitude.

68

CCS Concepts: • **Theory of computation** → **Program verification**; *Type theory*; *Shared memory algorithms*; *Concurrent algorithms*; *Program analysis*; *Invariants*; • **Software and its engineering** → **Memory management**; **Model checking**; *Automated static analysis.*

Additional Key Words and Phrases: lock-free data structures, safe memory reclamation, garbage collection, linearizability, verification, type systems, type inference

ACM Reference Format:
Roland Meyer and Sebastian Wolff. 2020. Pointer Life Cycle Types for Lock-Free Data Structures with Memory Reclamation. *Proc. ACM Program. Lang.* 4, POPL, Article 68 (January 2020), 36 pages. https://doi.org/10.1145/3371136

1 INTRODUCTION

In the last decade we have experienced an upsurge in massive parallelization being available even in commodity hardware. To keep up with this trend, popular programming languages include in their standard libraries features to make parallelization available to everyone. At the heart of this effort are concurrent (thread-safe) data structures. Consequently, efficient implementations are in high demand. In practice, lock-free data structures are particularly efficient.

Authors’ addresses: Roland Meyer, TU Braunschweig, Germany, roland.meyer@tu-bs.de; Sebastian Wolff, TU Braunschweig, Germany, sebastian.wolff@tu-bs.de.

Experiments

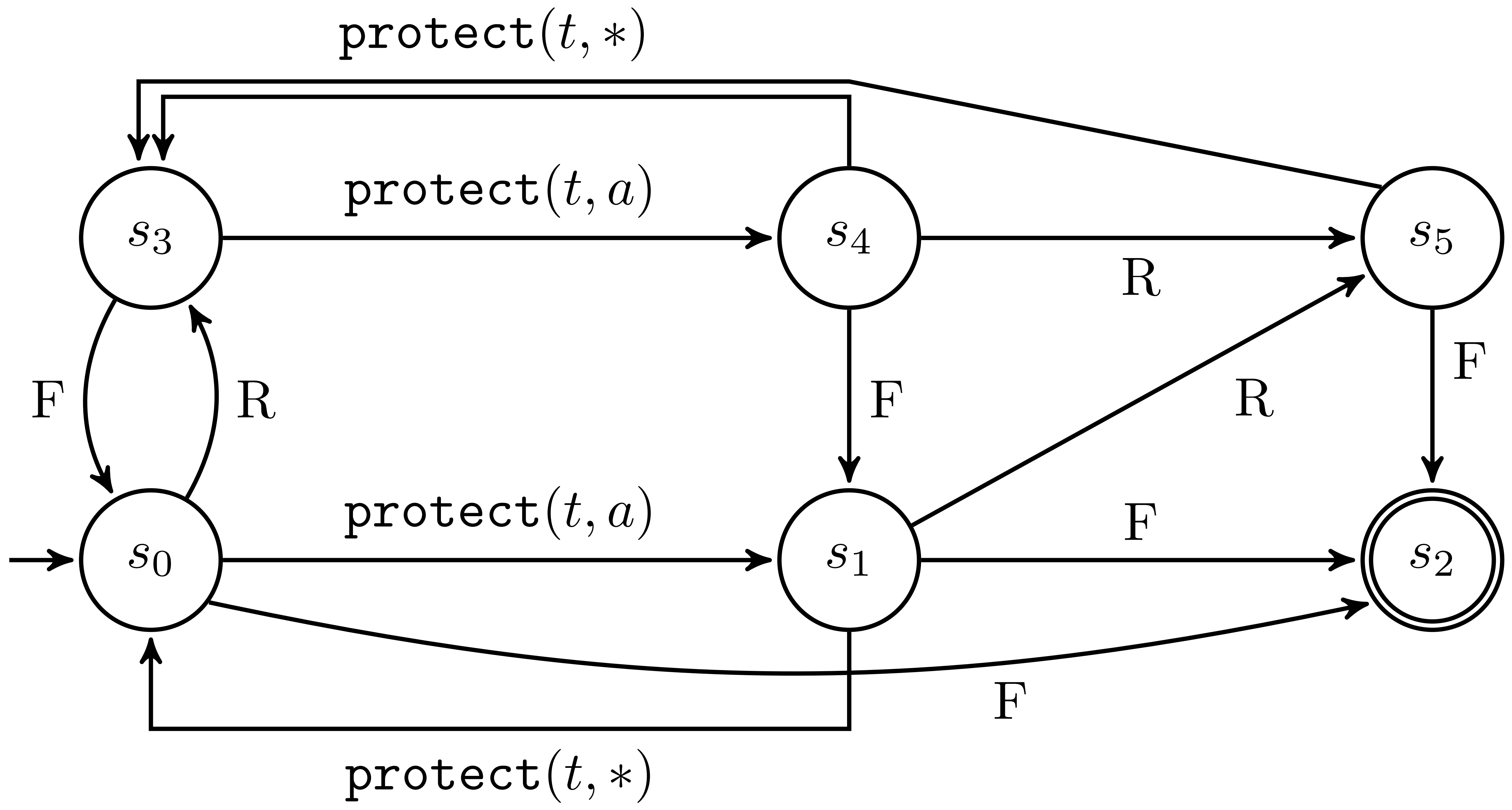
Data Structure with HP	Types	Annot.	Lin.
Treiber's stack	0.7s ✓	12s ✓	1s ✓
Michael&Scott's queue	0.6s ✓	11s ✓	4s ✓
DGLM queue	0.6s ✓	1s ✗*	5s ✓
Vechev&Yahav's 2CAS set	1.2s ✓	13s ✓	98s ✓
Vechev&Yahav's CAS set	1.2s ✓	3.5h ✓	42m ✓
ORVYY set	1.2s ✓	3.2h ✓	47m ✓
Michael's set	1.2s ✓	90s ✗*	t/o 🕒

* imprecision in the back-end verifier

Example

```
Node* head = Head;

protect(head);

atomic {

   @active Head

   assume(head == Head);
```

Type for **head**



$$F \; := \; \texttt{free}(t,a) \vee \texttt{free}(*,a)$$

$$R \; := \; \texttt{retire}(t,a) \vee \texttt{retire}(*,a)$$