Crane Lowers Rocq Safely into C++

Bloomberg

Rocq for Programming Languages 2026 (RocqPL 2026)
January 17, 2026

Matthew Z. Weaver
✉ mweaver89@bloomberg.net

Joomy Korkut
✉ jkorkut@bloomberg.net @joomy @joomy@functional.cafe

TechAtBloomberg.com

Bugs are bad!

Testing is great, but not enough!

11 groups/bsl/bslstl/bslstl_hashtable.h

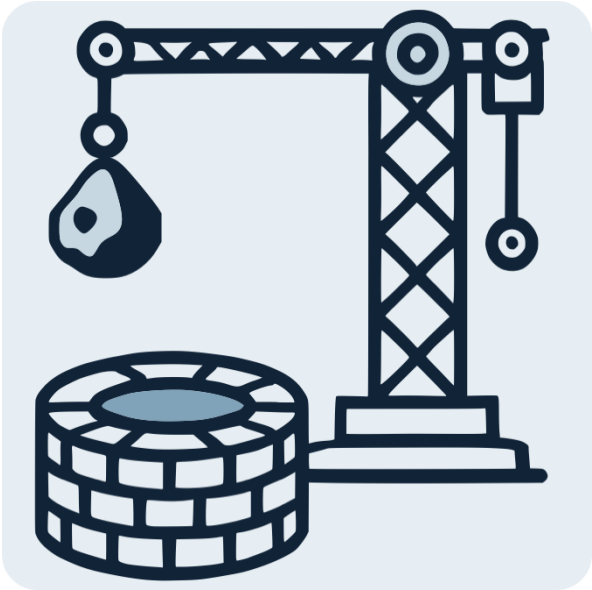
```
4420  4420     template <class KEY_CONFIG, class HASHER, class COMPARATOR, class ALLOCATOR>
4421  4421     void
4422  4422     HashTable<KEY_CONFIG, HASHER, COMPARATOR, ALLOCATOR>::removeAll()
4423  4423     {
4424  4424         this->removeAllImp();
4425        -     native_std::memset(
4426        -                 d_anchor.bucketArrayAddress(),
4427        -                 0,
4428        -                 sizeof(bslalg::HashTableBucket) * d_anchor.bucketArraySize());
      4425  +     if (HashTable_ImpDetails::defaultBucketAddress() !=
      4426  +         d_anchor.bucketArrayAddress()) {
      4427  +         native_std::memset(d_anchor.bucketArrayAddress(),
      4428  +                            0,
      4429  +                            sizeof(bslalg::HashTableBucket) *
      4430  +                                d_anchor.bucketArraySize());
      4431  +     }
4429  4432
4430  4433         d_anchor.setListRootAddress(0);
4431  4434         d_size = 0;
4432  4435     }
```

causes a data race because different threads are writing to the same common address!

Tests are **weak** at **finding** data races and **useless** for proving their **absence;** we need **formal methods** to prove the **absence** of data races!

New Tool: Crane

- A new extraction system from Rocq to C++.
- Generates **functional-style**, **memory-safe**, **thread-safe**, **readable** C++ code.
- Still under active development!



Why C++?

- C++ is the **primary programming language** and the **lingua franca** of our engineering teams, so, we are meeting them where they are!
  - We strive to generate readable, verified C++ library files for engineers to seamlessly integrate into production code.
- C++ has a decent functional subset, but it also provides enough flexibility for low-level optimizations.
  - We can treat it like a portable assembly language for functional language compilers!

What does functional-style C++ look like?

```
Inductive list (A : Type) : Type :=
| nil : list A
| cons : A -> list A -> list A.
```

```
template <typename A> struct nil;
template <typename A> struct cons;
template <typename A> using list =
variant<nil<A>, cons<A>>;

template <typename A> struct nil {
  static shared_ptr<list<A>> make() {
    return make_shared<list<A>>(nil<A>{});
  }
};

template <typename A> struct cons {
  A x;
  shared_ptr<list<A>> xs;
  static shared_ptr<list<A>>
  make(A x, shared_ptr<list<A>> xs) {
    return make_shared<list<A>>(cons<A>{x, xs});
  }
};
```

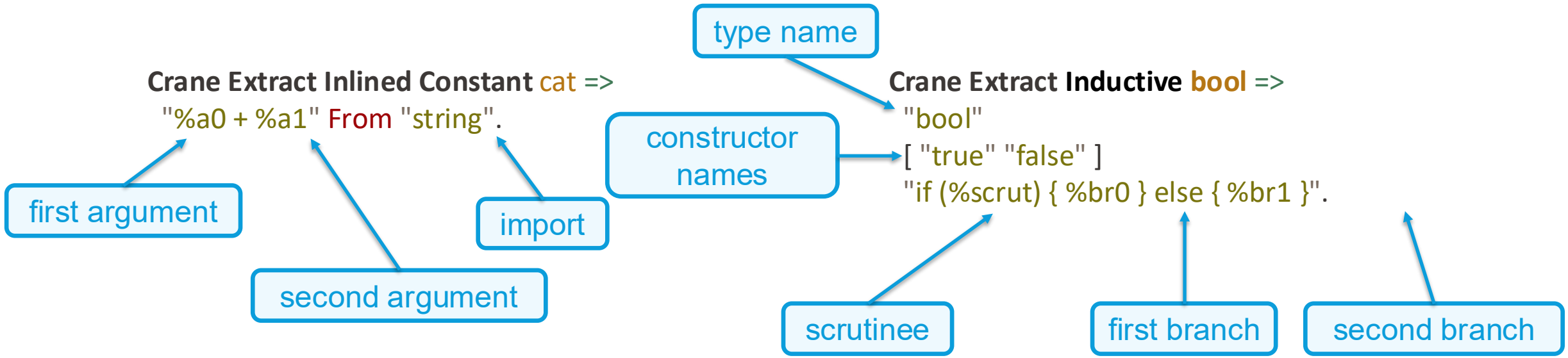
What does functional-style C++ look like?

```
Fixpoint map {A B : Type}
         (f : A -> B)
         (l : list A) : list B :=
 match l with
 | nil => nil
 | cons x l' => cons (f x) (map f l')
 end.
```

```
template <typename A, typename B, MapsTo<B, A> F>
shared_ptr<list<B>>
map(F &&f, const shared_ptr<list<A>> l) {
 return visit(Overloaded {
  [&](const nil<A> _args) -> shared_ptr<list<B>> {
   return nil<B>::make();
  },
  [&](const cons<A> _args) -> shared_ptr<list<B>> {
   return cons<B>::make(
    f(_args.x),
    map<A, B, F>(f, _args.xs));
  }},
  *l);
}
```

Customizing Extraction

- As with OCaml extraction, Crane allows users to customize how specific definitions are translated by extraction.



Monadic Effects!

- Rocq is a **pure** functional language:
  no IO, no state, no concurrency, just functions manipulating values.

- But… pure functional languages can represent effectful computation
  using **monads**!

- We created an abstract interface for users to specify both
  - the functional interface for monadic effects,
  - and how they should extract to C++.

- We choose to derive our monads from **interaction trees**.

Monadic Effects!

```
Inductive iIO (A : Type) : Type :=
| iprint : string -> iIO unit
...
```

We define a type family containing the operations of our monad.

```
Definition IO : Type -> Type := itree iIO.
Definition print (s : string) : IO unit := trigger (iprint s).
...
```

We then derive the monadic interface using itrees.

Monadic Effects!

```
Crane Extract Monad IO [bind := bind, ret := Ret].
Crane Extract Inlined Constant print => "std::cout << %a0".
...
```

We declare our type a monad to Crane...

...and specify how each operations should be extracted.

```
Definition io_test (s : string) : IO unit :=
    print (cat "printing " s) ;;
    Ret tt.
```

We now can write effectful programs...

```
void io_test(const std::string s) {
    std::cout << "printing " + s;
    return;
}
```

...and extract them to C++!

Why is concurrency hard?

Uncontrolled:

Multiple threads use the same memory location at the same time ➛ possible **data race**! 👎

Locks:

Multiple threads lock memory locations before use, and unlock after use, so no data race 👍,
but they keep threads waiting ⏳, can deadlock 🔒 (and wait forever), and don't compose!

Software transactional memory:

Multiple threads perform operations optimistically, check if there's a change before committing
No data race, composable, no deadlocks, no waiting 👍
May repeat expensive computation, limits side effects in atomic blocks

Concurrent Hash Table in Rocq with Crane

1. Implemented **STM** as a C++ library.
2. Defined and connected the **STM** interface as a monad in Crane.
3. Implemented **hash table** in Rocq.
4. Extracted to C++ with Crane.

- We are hoping to verify the C++ STM library and provide a proof interface for concurrent programs using STM.

What do we mean by “safely”?

- Our extractor is **not** verified; therefore, our approach can be considered **lightweight formal methods**.

- We are developing a **differential testing** framework to compare outputs of randomly generated Rocq programs compiled with Crane and other compilers and extractors (e.g. CertiCoq, extraction to Malfunction, etc.).

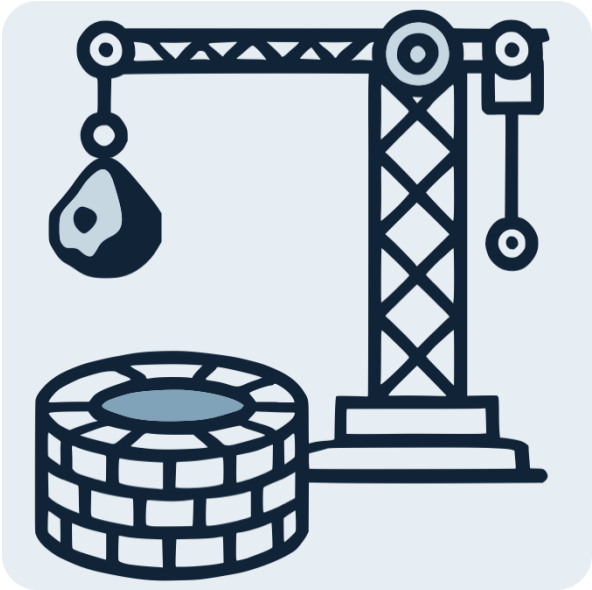
- Generated code will also undergo the same extensive **testing** and **static analysis** all handwritten code at Bloomberg is subject to.

Future Work

- While the basics are (nearly) up and running, we have so much more to do, including (but not limited to):
  - Support **more** of Rocq's language features (more complex inductive types, coinductives, parameterized modules, type classes, etc.)
  - Improve the **efficiency** of generated code without sacrificing readability *too much* (our plan: CPS + defunctionalization)
  - Goal: extract C++ code that runs **faster** than extracted OCaml code!
  - Get Crane-generated code into **production**!

Summary

- Crane is a new extraction system from Rocq to C++, that generates **functional-style**, **memory-safe**, **thread-safe**, **readable** C++ code.
- It is not verified, but we have other lightweight formal methods to ensure the safety of the generated code.
- Using Crane, we are implementing verified C++ libraries such as concurrent hash tables, that will eventually be used by Bloomberg engineers.



Thank you!

Check out our project at:
https://bloomberg.github.io/crane

Bloomberg

TechAtBloomberg.com