Static Basic Block Versioning

Olivier Melançon
Université de Montréal

Marc Feeley
Université de Montréal

Manuel Serrano
INRIA/UCA

How to safely remove runtime checks in dynamic languages?

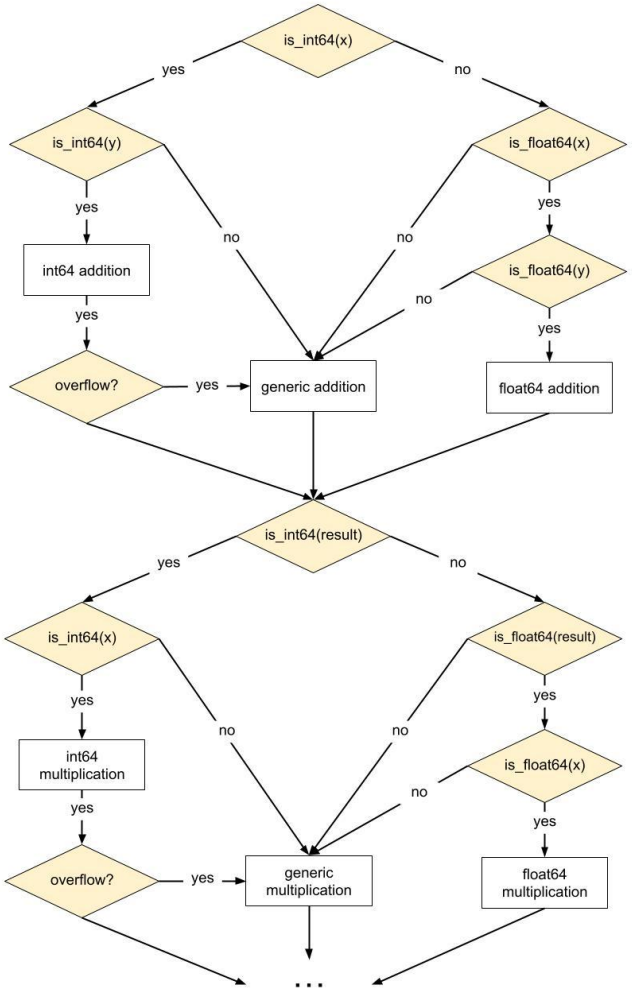
Propagation of Type Information

Example:

```
(x + y) * x
```

Branches: **gain** information

Join point: **lose** information

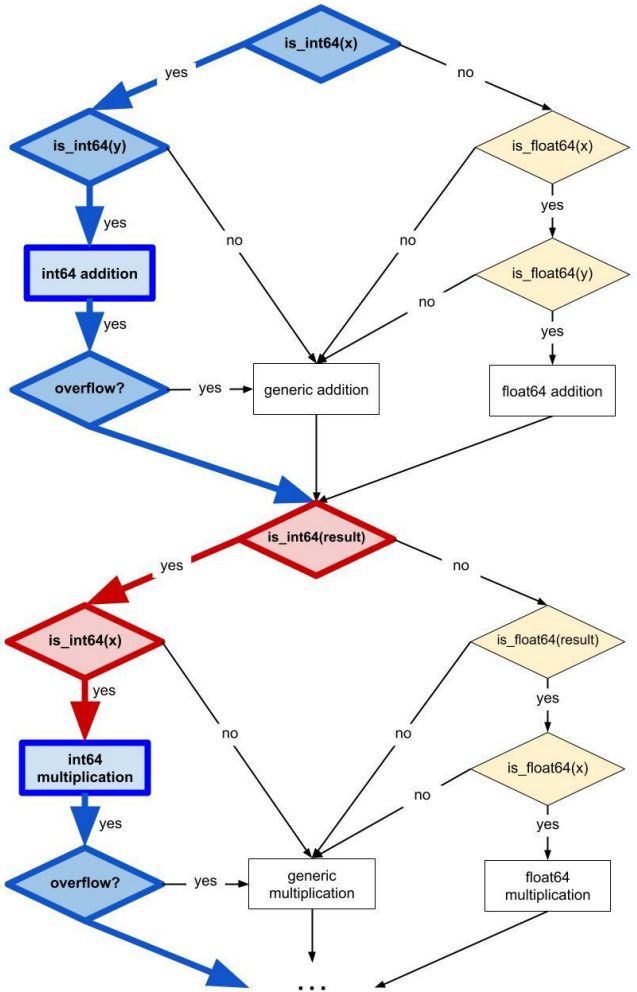


Propagation of Type Information

Example:

```
(x + y) * x
```

Integer addition fast path
Redundant tests in red



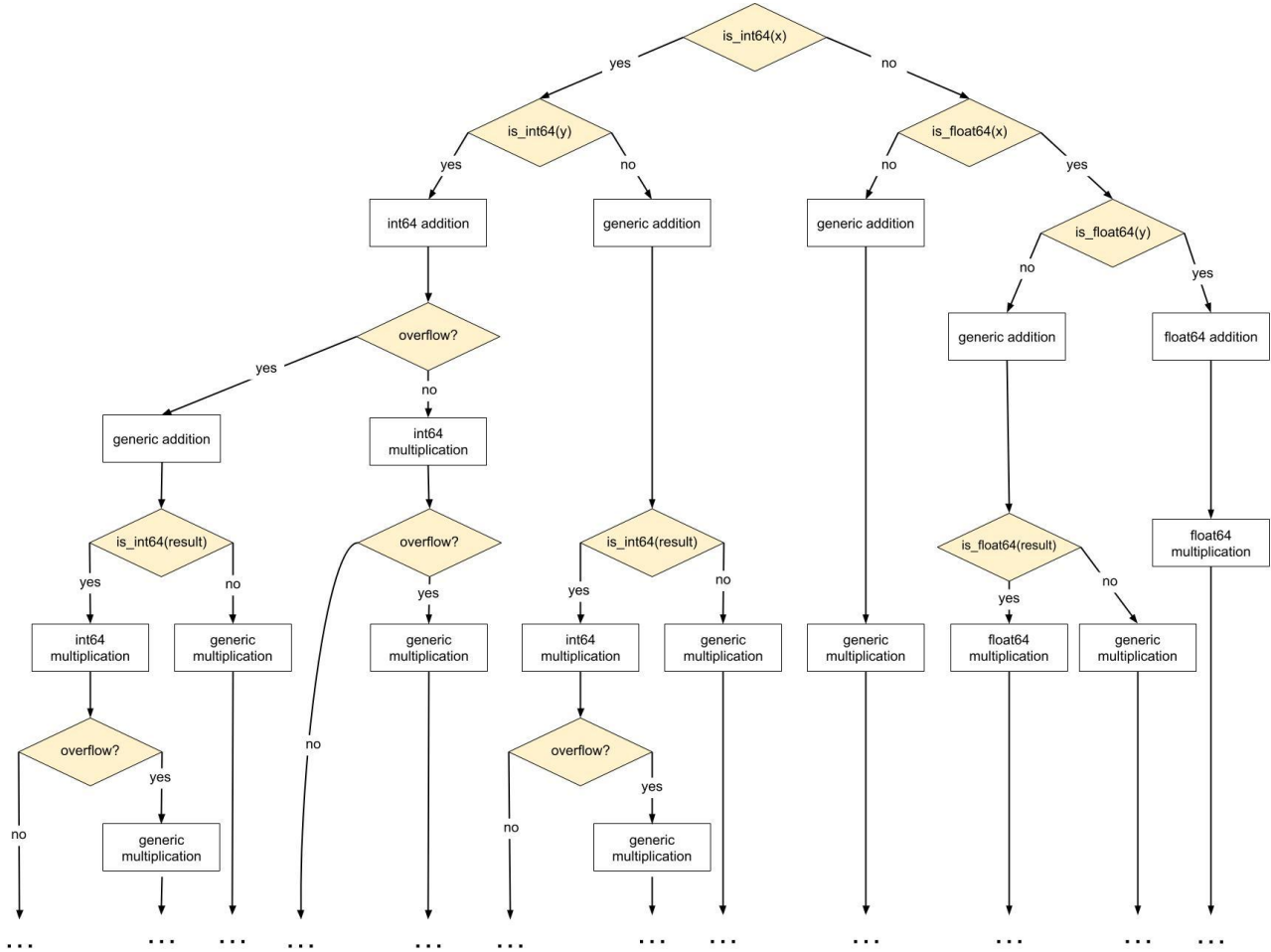
Branches: **gain** information

Join point: **lose** information

Duplicate!

Duplicating join points

prevents loss…

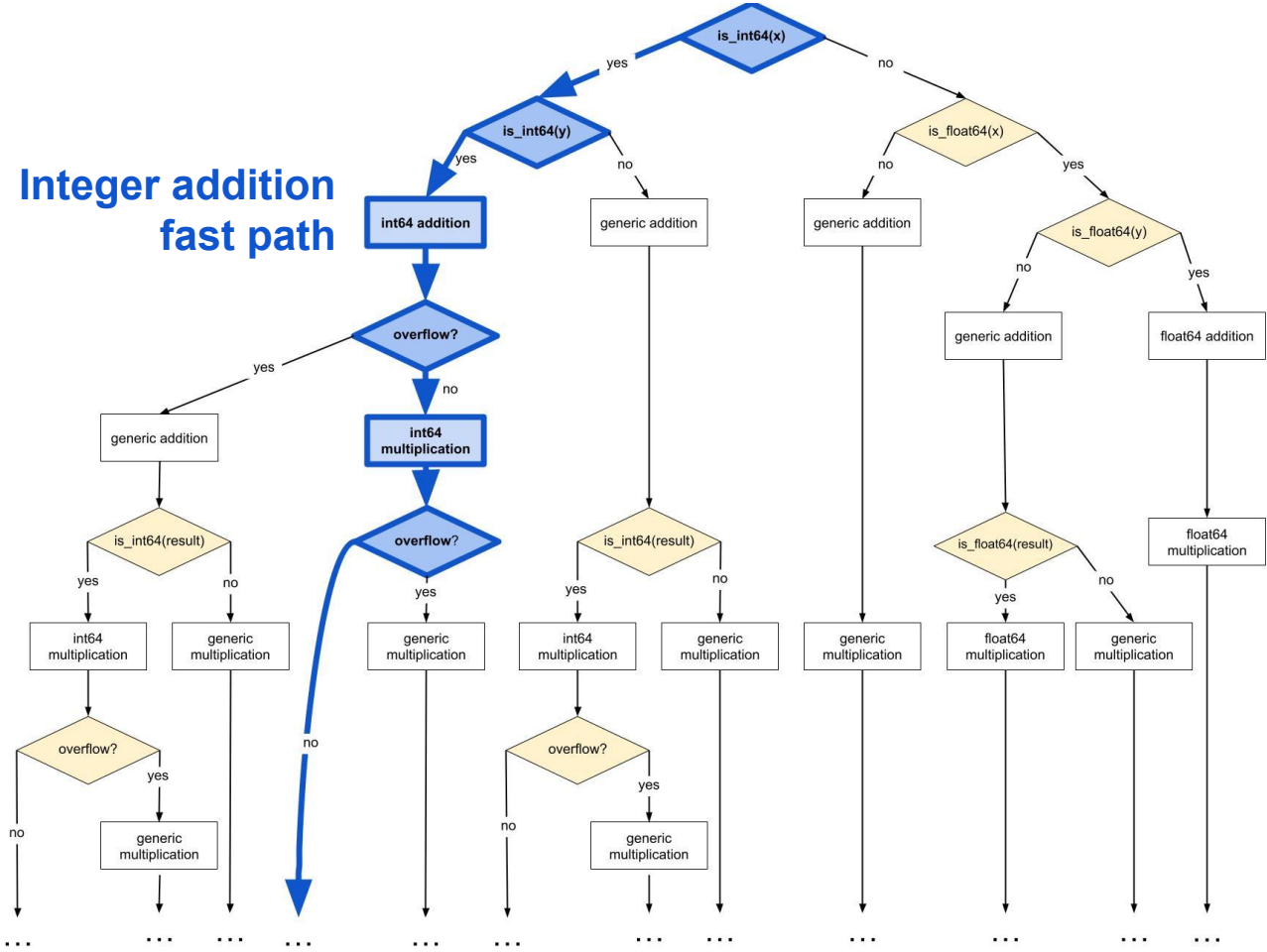


Duplicate!

Duplicating join points

prevents loss…

but causes **bloat**.



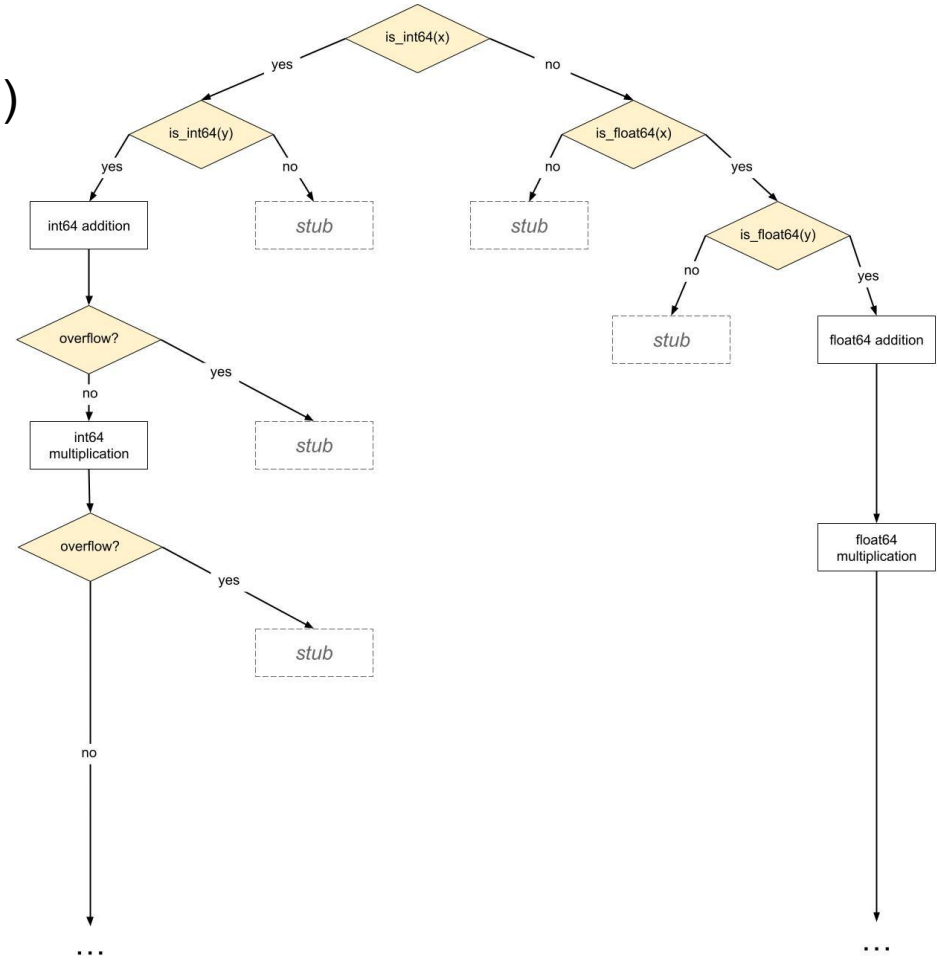
Duplicate a little…

Set a limit.

Only duplicate *useful* versions.

But which versions?

Lazy Basic Block Versioning (Just-in-Time)



Use runtime context to choose.

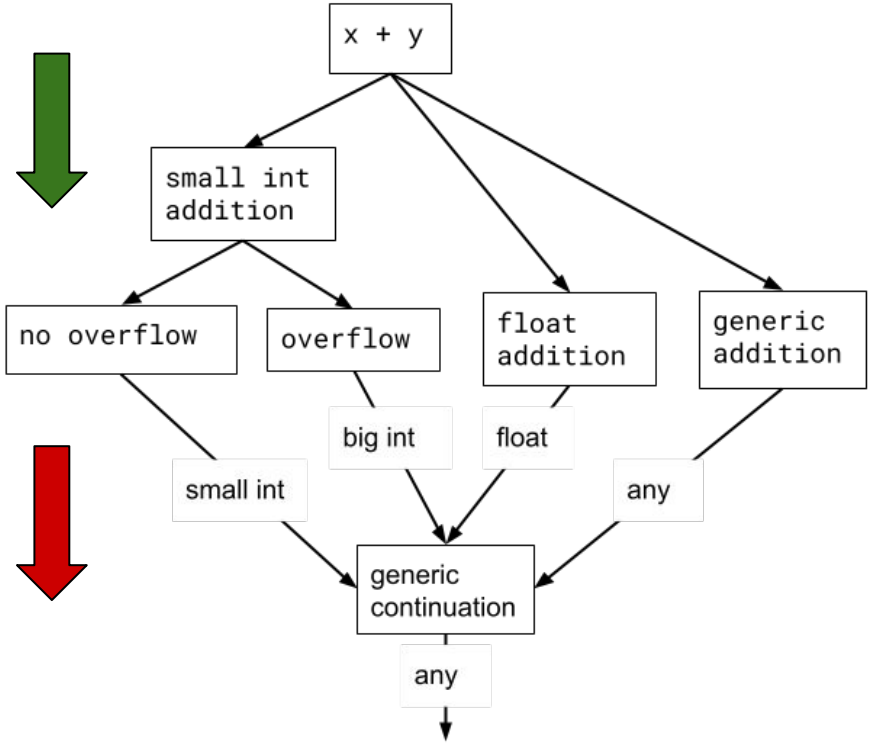
Generic version when limit is reached.

But what if we can’t use a JIT?

Propagation of Type Information

Branches: **gain** information

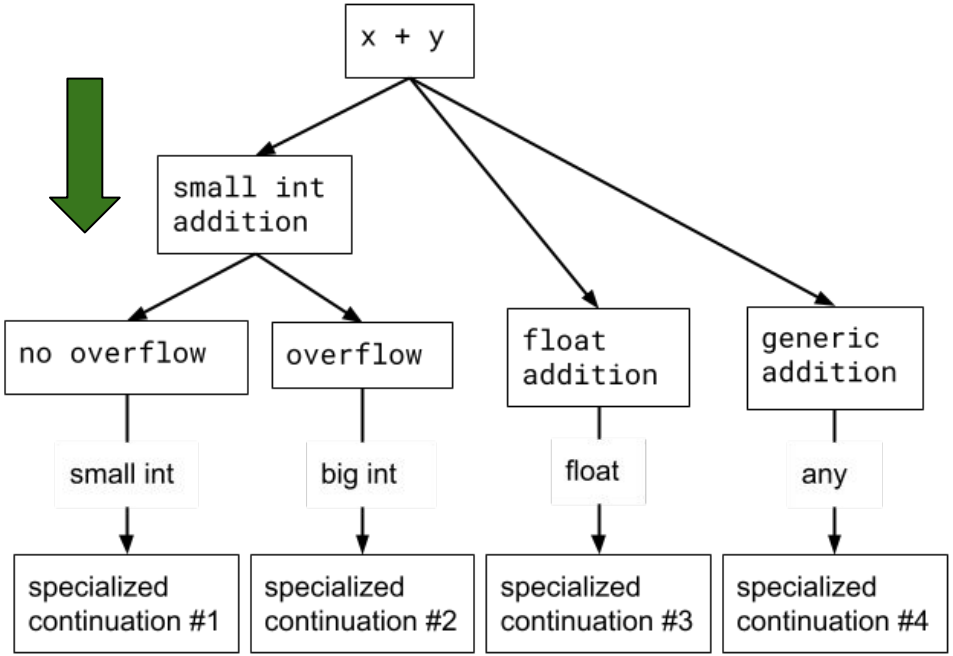
Join point: **lose** information



Duplicate!

Duplicating join points prevents loss

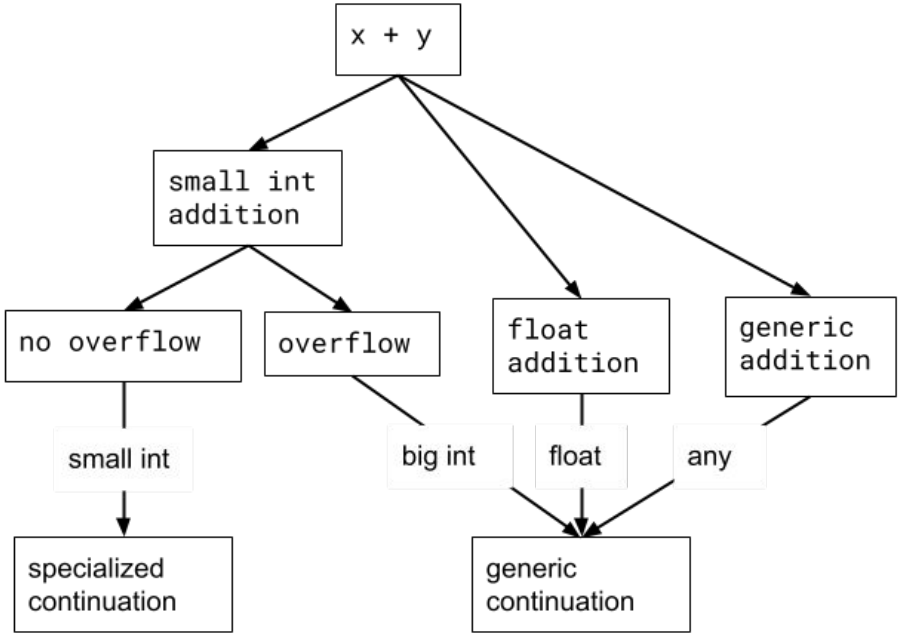
… but causes **bloat**.



Duplicate a little…

Set a version limit.

But which versions?

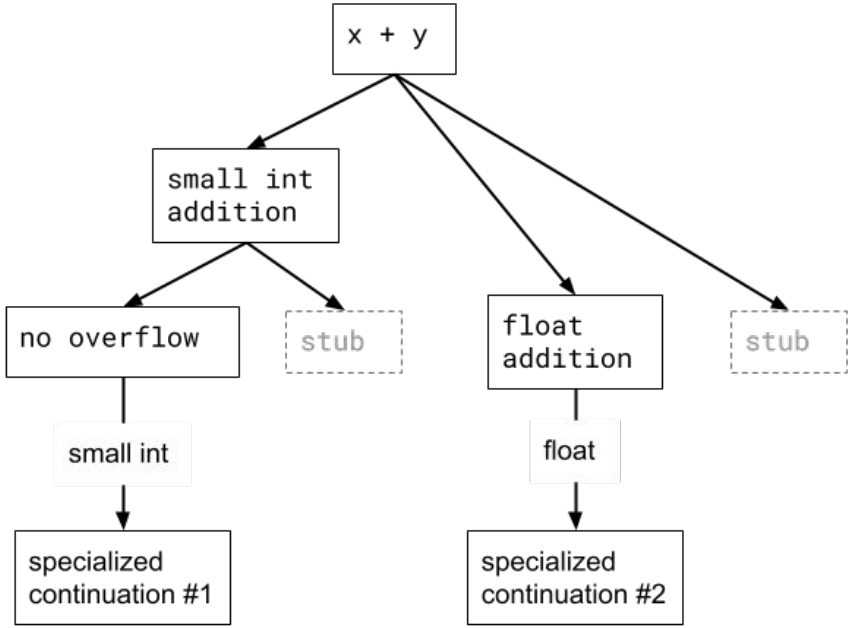


Lazy Basic Block Versioning (Just-in-Time)

Use runtime context to choose.

Generic version when limit is exceeded.

But what if we can’t use a JIT?



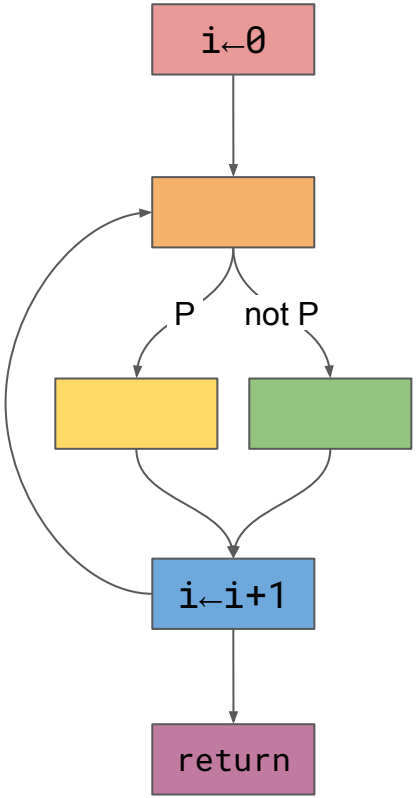
Static
Basic Block Versioning

An **ahead-of-time** variant of Basic Block Versioning

Overview

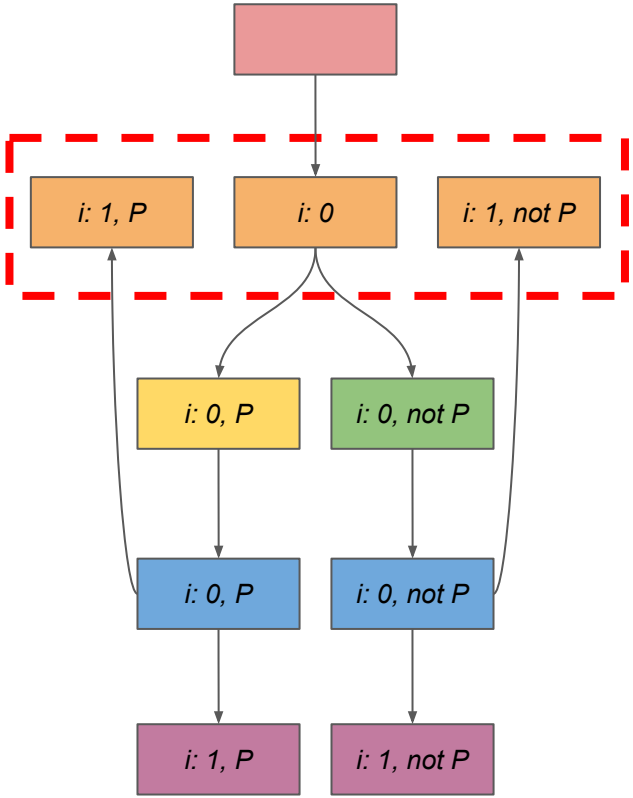
1. Set a **version limit**;
2. **Traverse** the Control Flow Graph and accumulate information from checks;
3. Greedily create a new version of a basic block for **each distinct context**;
4. **Merge versions** together when the limit is exceeded;
5. **Repeat** until fixed point.

Control Flow Graph

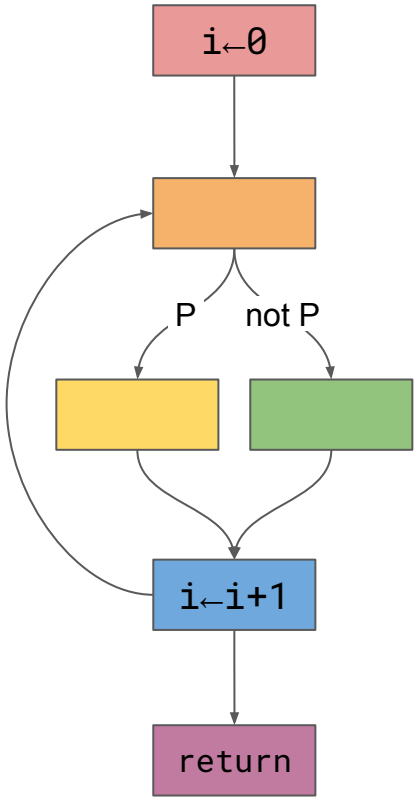


Specialized Control Flow Graph
Version Limit = 2

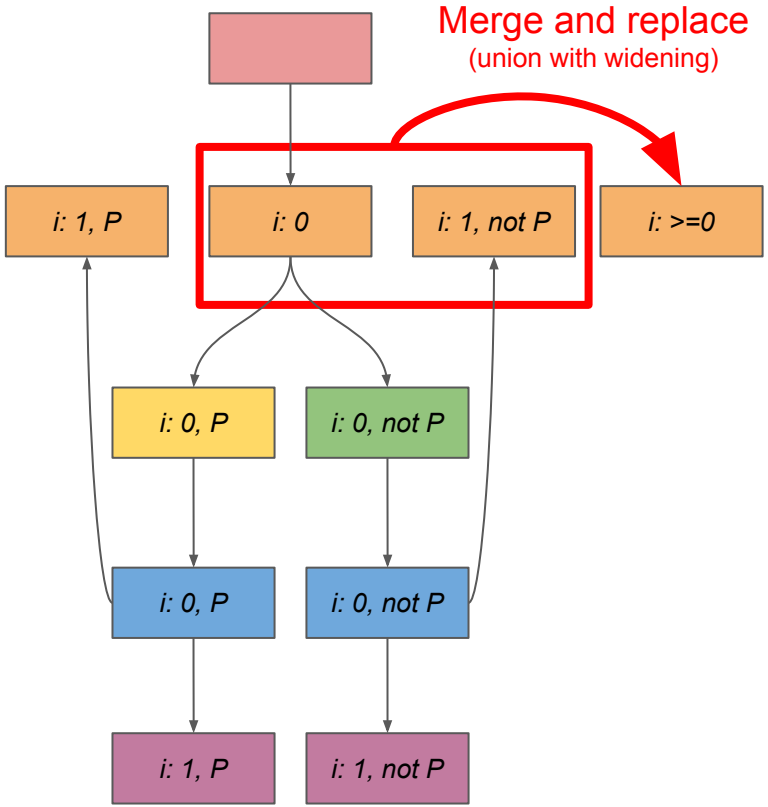
Limit exceeded



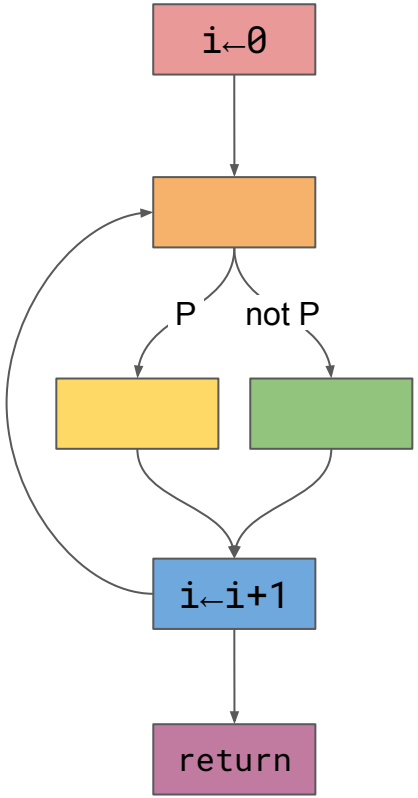
Control Flow Graph



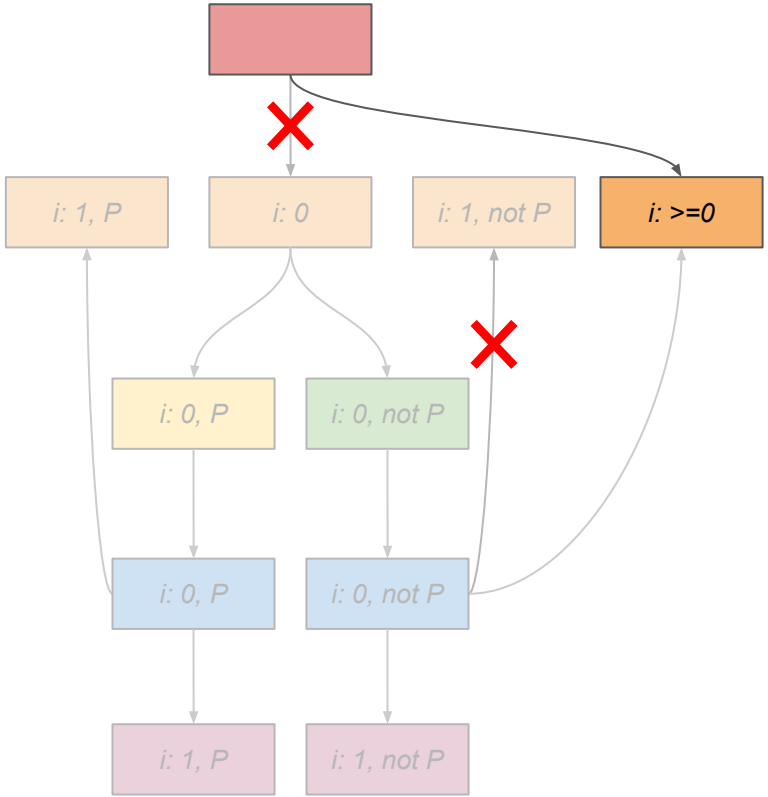
Specialized Control Flow Graph
Version Limit = 2



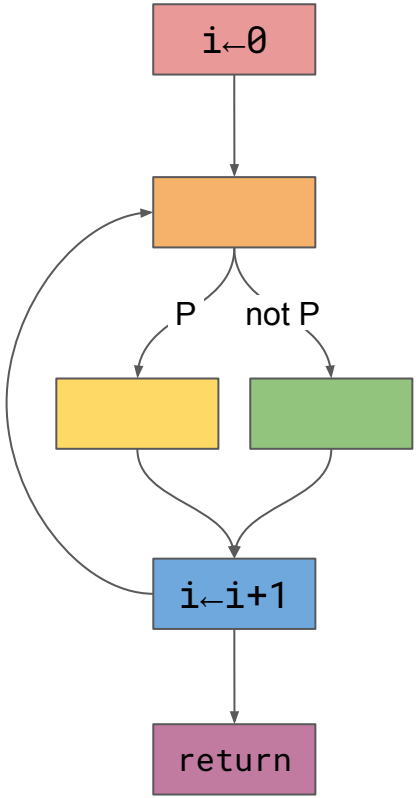
Control Flow Graph



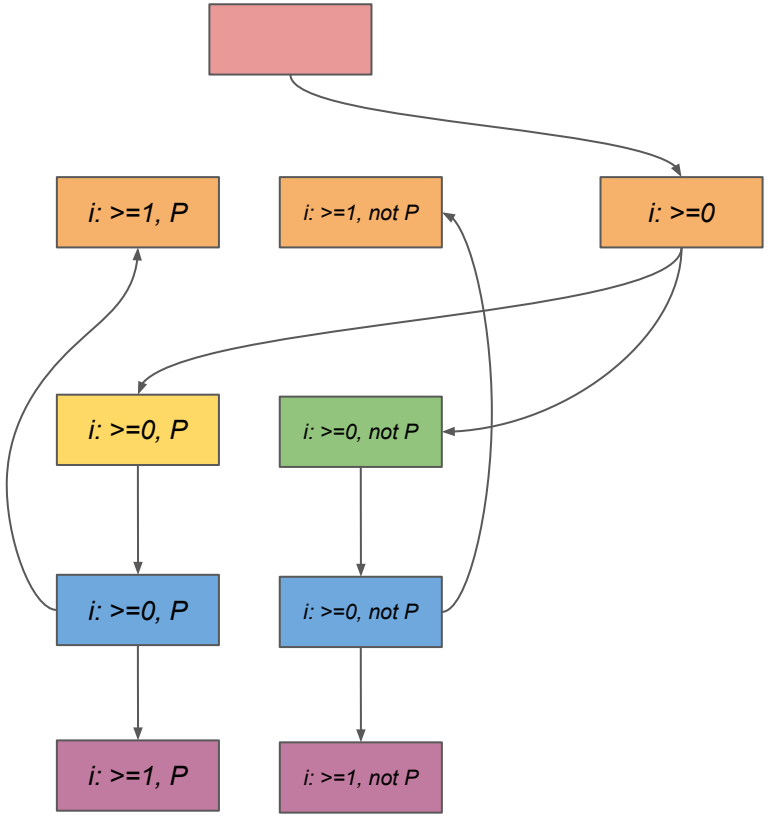
Specialized Control Flow Graph
Version Limit = 2



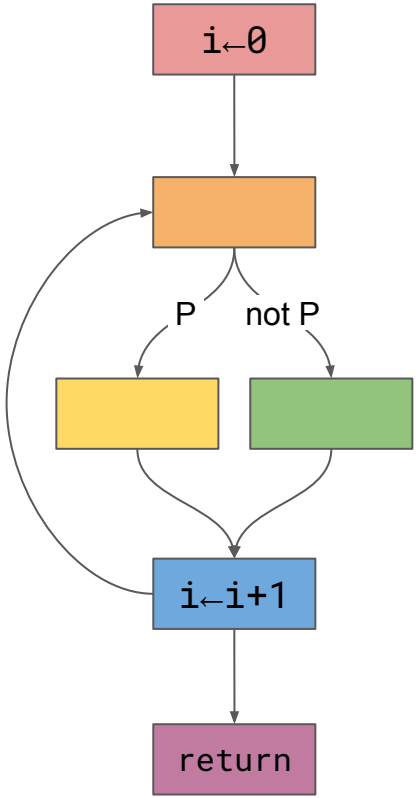
Control Flow Graph



Specialized Control Flow Graph
Version Limit = 2

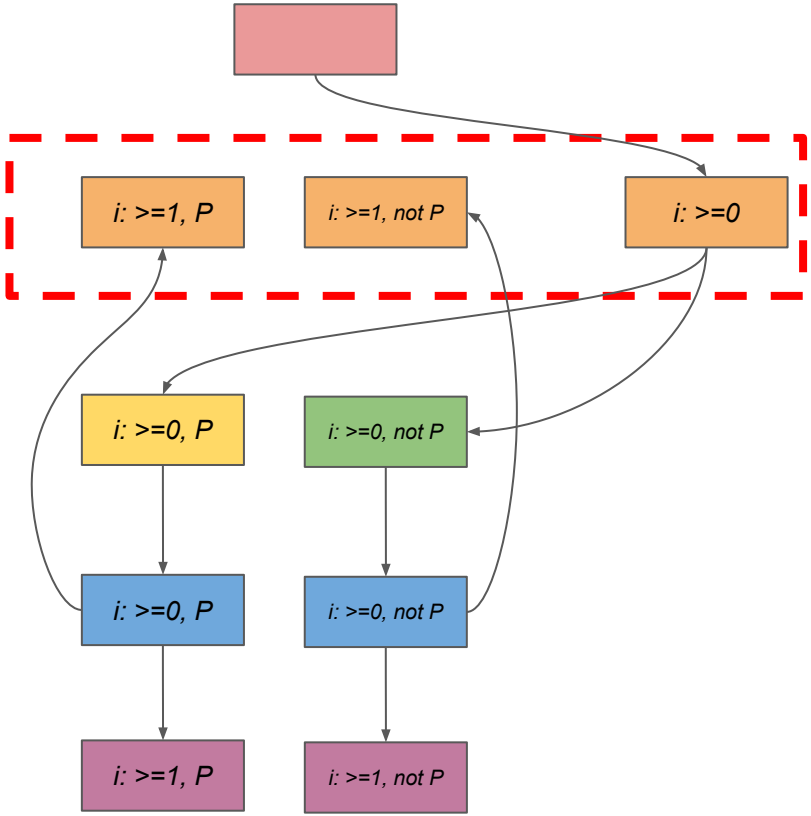


Control Flow Graph

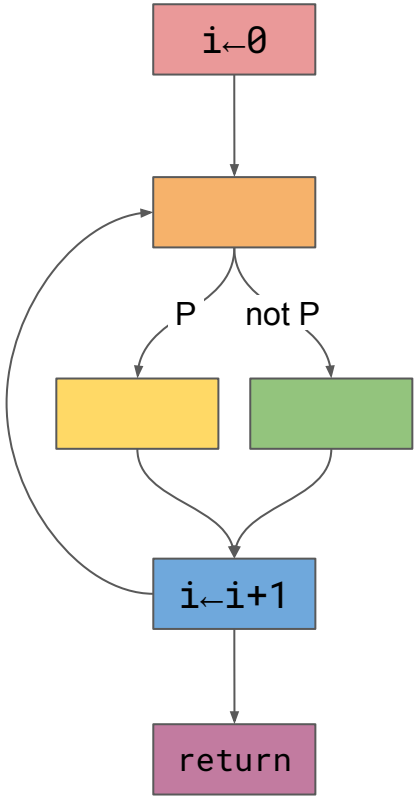


Specialized Control Flow Graph
Version Limit = 2

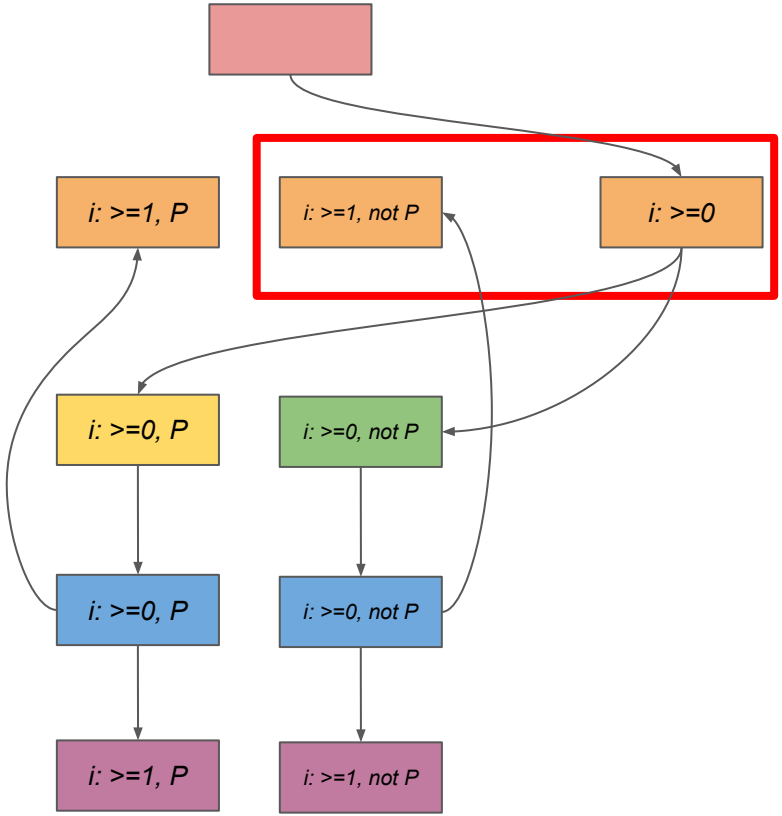
Limit exceeded



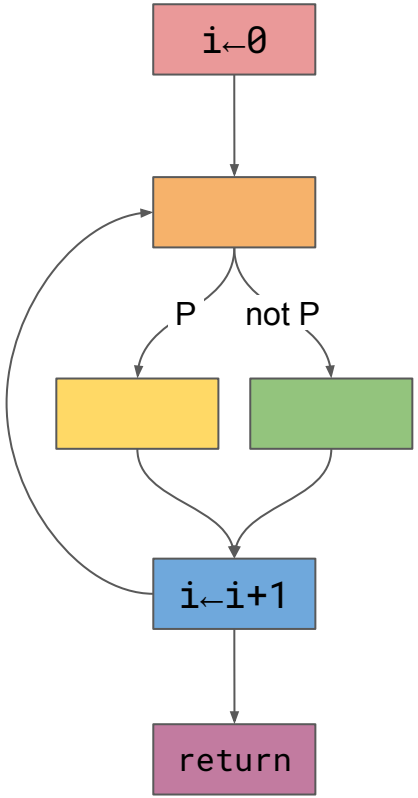
Control Flow Graph



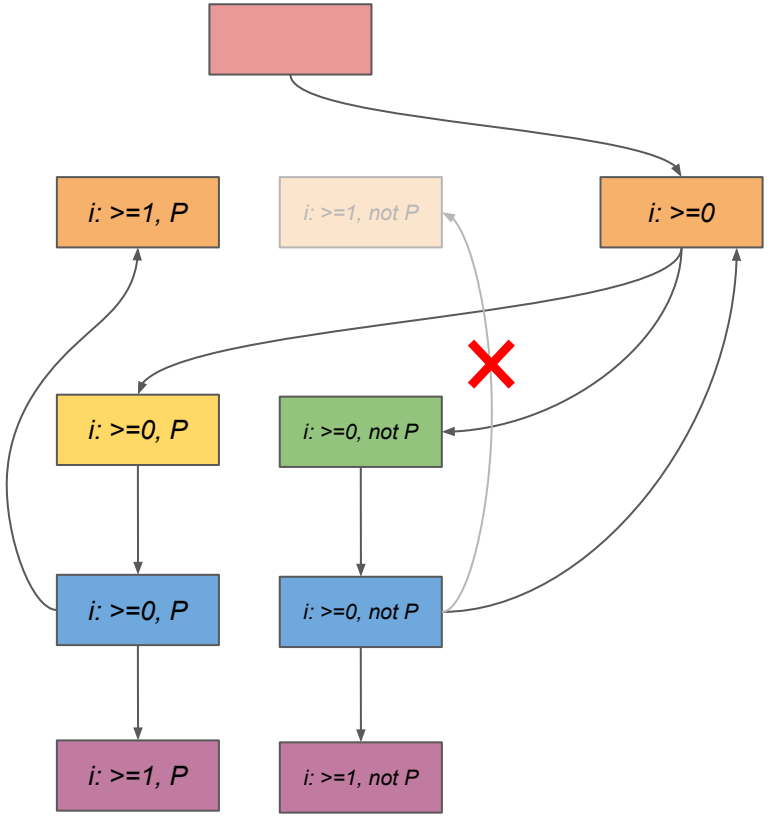
Specialized Control Flow Graph
Version Limit = 2



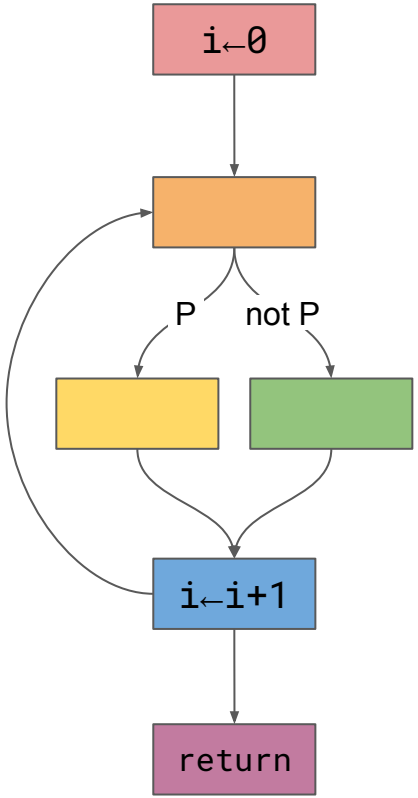
Control Flow Graph



Specialized Control Flow Graph
Version Limit = 2

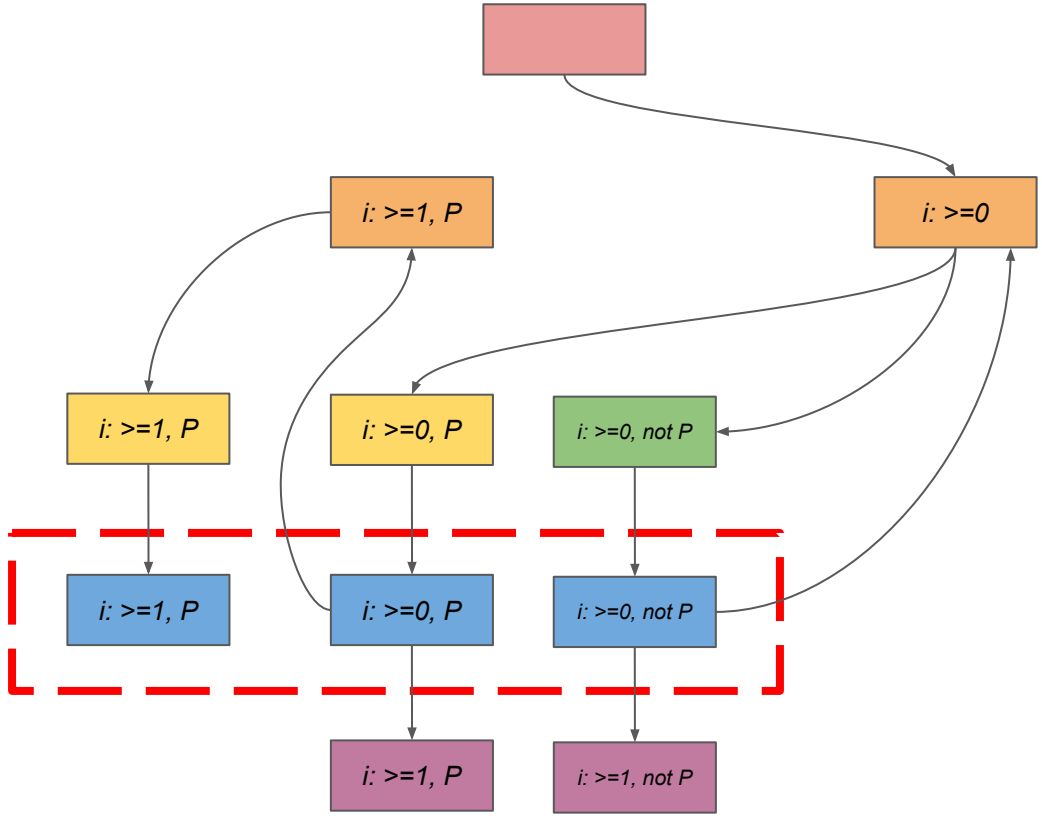


Control Flow Graph

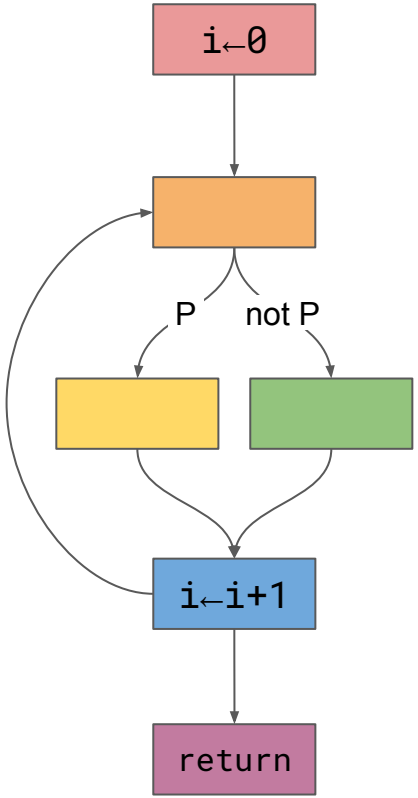


Limit exceeded

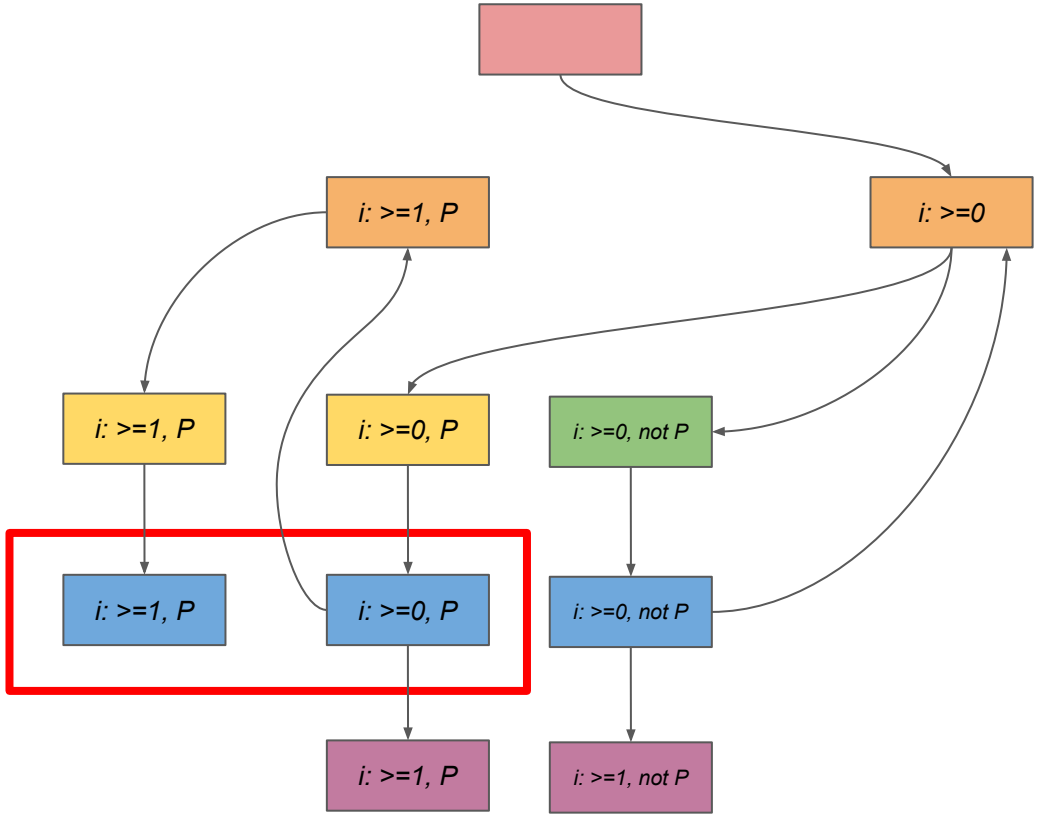
Specialized Control Flow Graph
Version Limit = 2



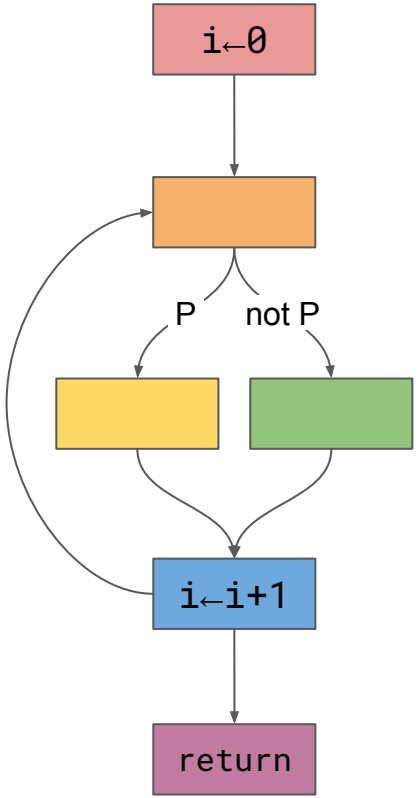
Control Flow Graph



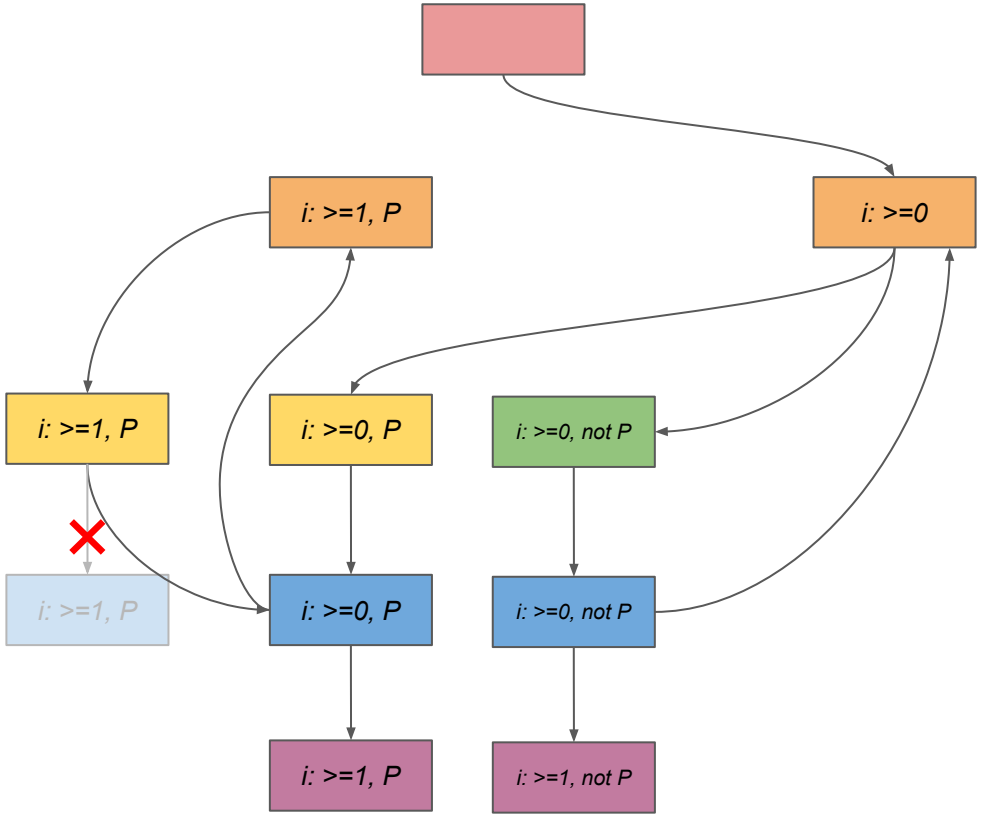
Specialized Control Flow Graph
Version Limit = 2



Control Flow Graph



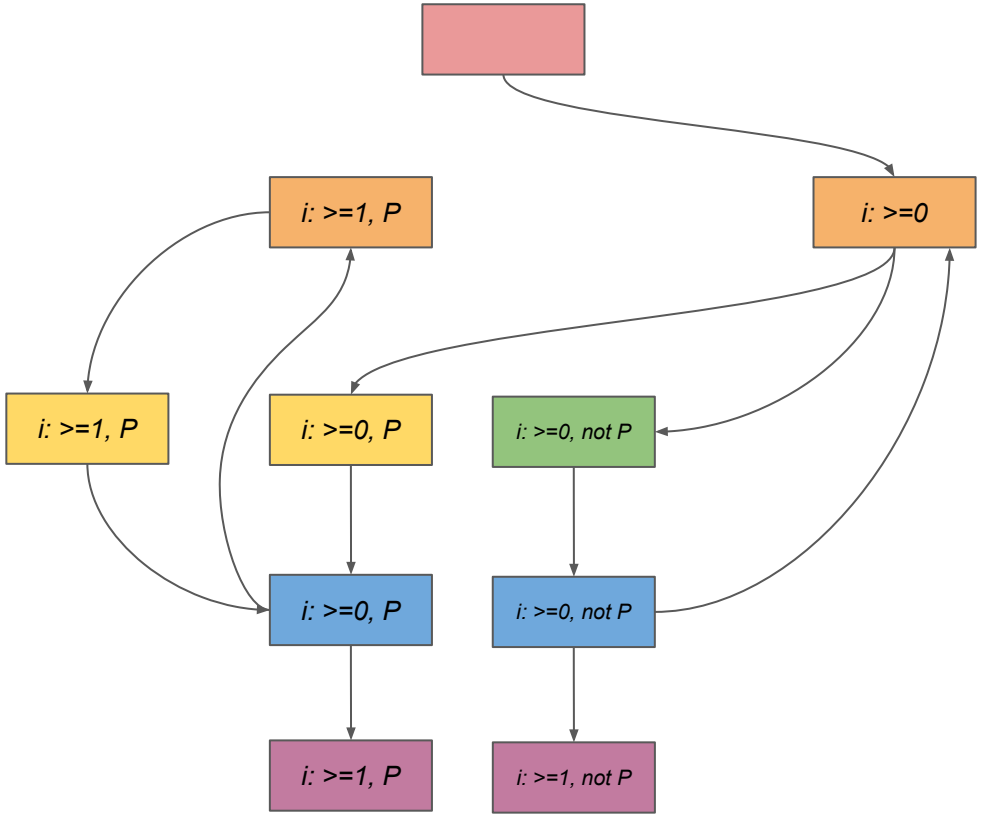
Specialized Control Flow Graph
Version Limit = 2



Desirable properties

- P never tested inside left loop;
- No generic version;
- At most two versions of each block.

Specialized Control Flow Graph
Version Limit = 2



Experimental results

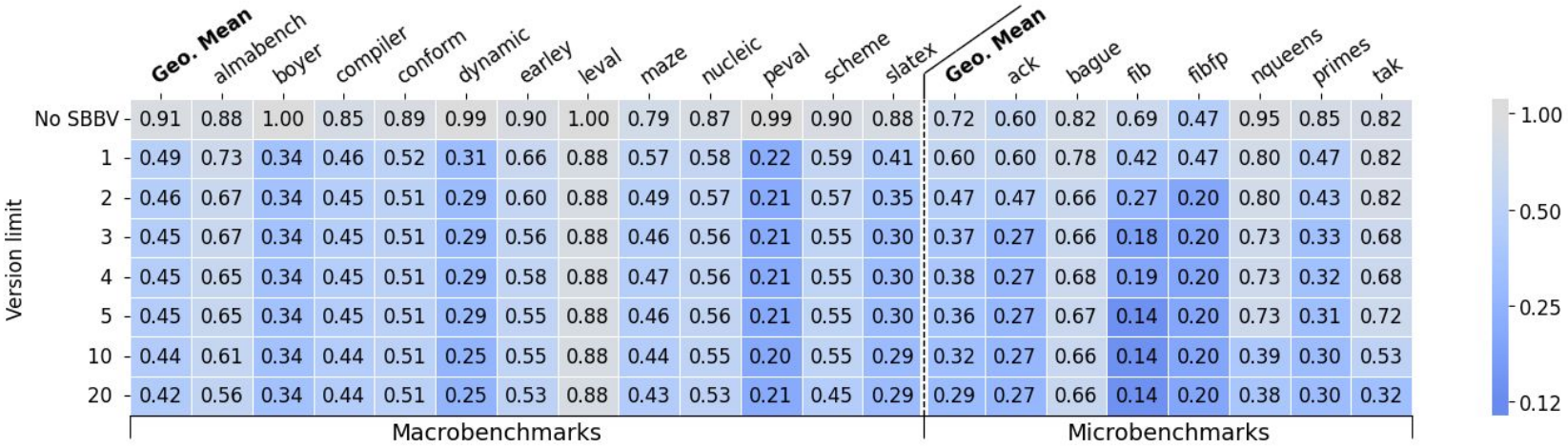
Experimental setup

Integration in **Bigloo** and **Gambit** which are among the fastest Scheme compilers

We measured how version limit affects:

- Runtime checks
- Execution time
- Program size
- Compilation time

Run time checks (Gambit)



Key takeaways

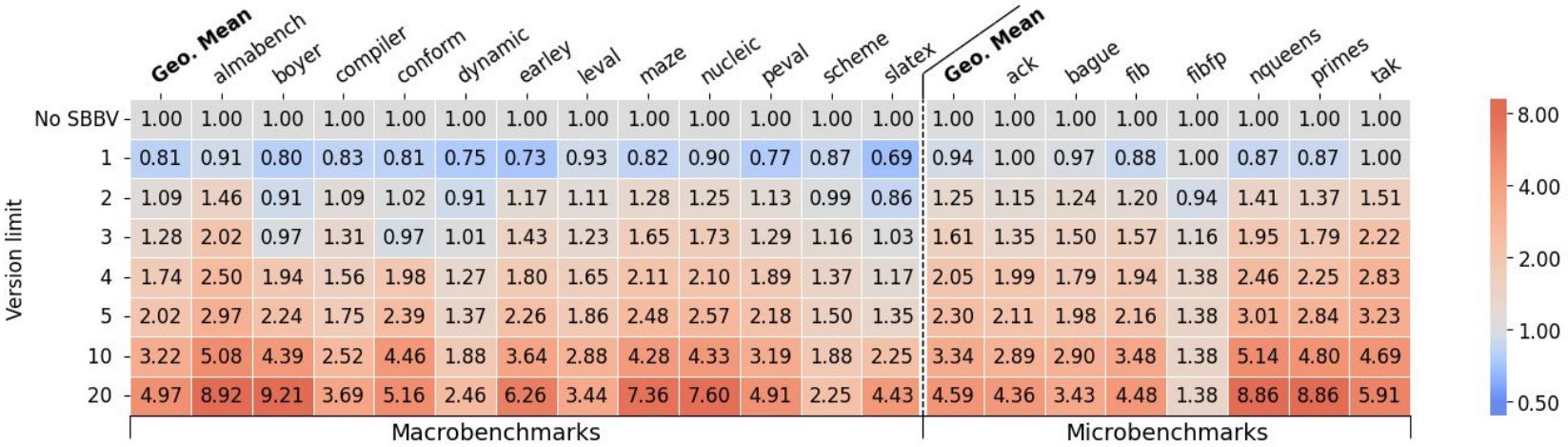
Higher limit decreases runtime checks

Diminishing return after 3 versions

Runtime checks

Type checks
Arithmetic comparison
Overflow
Array boundary

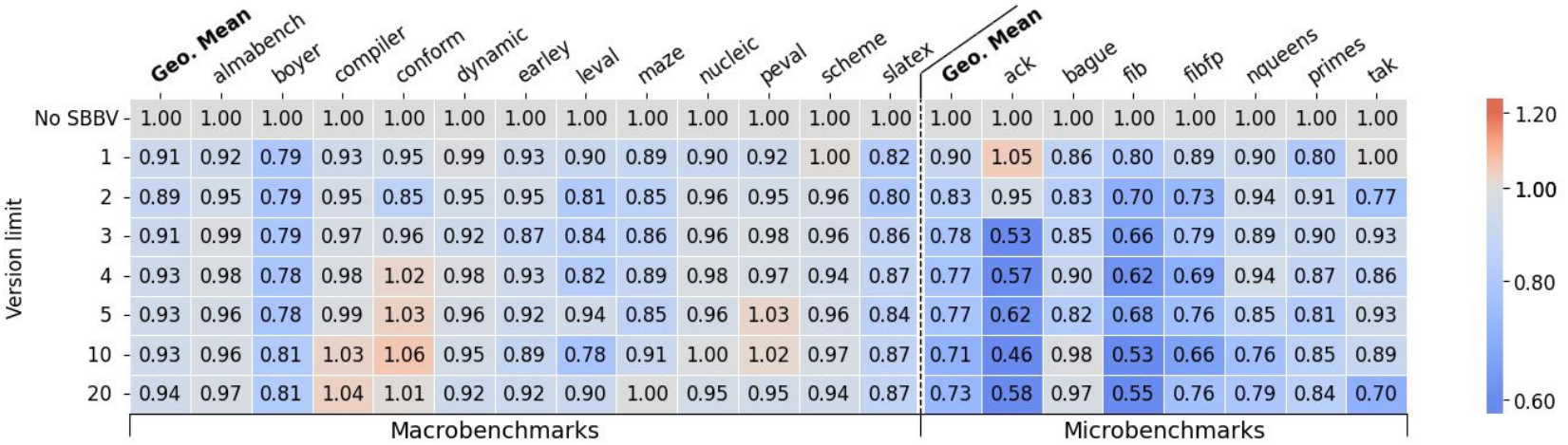
Program size (Gambit)



Key takeaways

Program can be smaller for low version limits because SBBV **implicitly applies dead code elimination**

Execution time (Gambit)

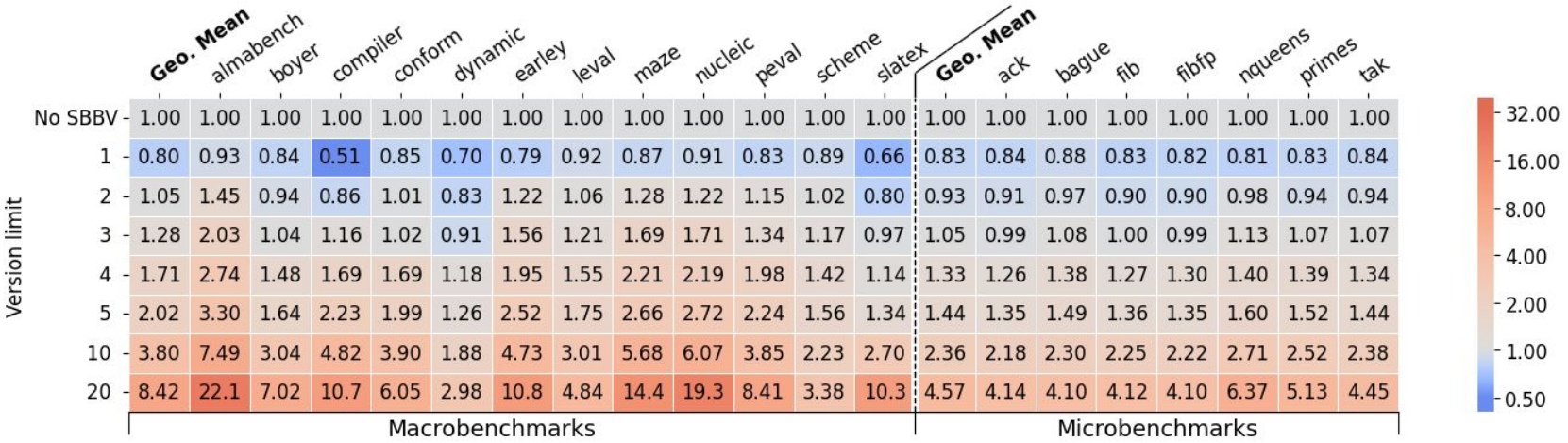


Key takeaways

Tradeoff between size and removed checks

Best with limit of 2 version: ~ 10% faster

Compilation time (Gambit)



Key takeaways

With limit of 2 versions: only 5% slower on average

Correlation with program size (due to `gcc` in our case)

Conclusion

- An **ahead-of-time** variant of Basic Block Versioning;
- Simple;
- Tested in two mature compilers;
- 10% speedup on average;
- More gain expected for languages with more costly checks (Python, JavaScript).