Typechecking of Overloading

in Programming Languages and Mechanized Mathematics

Arthur Charguéraud and Martin Bodin
with Louis Riboulet and Jana Dunfield

Inria

April 14th, 2026

Contribution

Present the first overloading resolution algorithm accompanied with a polynomial complexity bound.

Bound in terms of the size of the output typed tree.

Algorithm complete with respect to *iteratively solvable programs*, i.e. solvable with a chain of deduction steps (otherwise problem is NP-hard).

Assumes annotation on polymorphic definitions.

Contribution

Implementation in OCaml, parses extended OCaml syntax.

Overloading of functions, constants, constructors and record fields.

Supports instances with assumptions, e.g., instance for show on `'a list` requires instance of show for `'a`.

Output fully typed AST, with symbols resolved.
Extracts to standard, executable OCaml code.

Algorithm overview

Typechecking algorithm:

1. Gather constraints from ML typechecking, via standard unifications
2. Collect symbols in traversal order, overloaded functions both ways
3. Iteratively try to resolve symbols, in a carefully chosen order.

Algorithm overview

Typechecking algorithm:

1. Gather constraints from ML typechecking, via standard unifications
2. Collect symbols in traversal order, overloaded functions both ways
3. Iteratively try to resolve symbols, in a carefully chosen order.

Resolution rule:
Assume symbol x with a type T constrained by the context.
Assume x has candidate instances $(v_1 : T_1)$, ..., $(v_n : T_n)$.
Then x resolves to v_i if T unifies with T_i but not with T_j for $j \neq i$.

No backtracking

If an instance has *assumptions*, they are investigated only after the choice of the instance is made.

Overlapping

An instance can be declared to *override* a prior instance: in case of overlap, the second will be prefered.

Ex: generic instance for show-list, but specific instance for show-list-of-char, to print using string notation.

Possible outcomes

Termination is ensured by limiting the dependency depth.

- Success: all resolved
- Ill-typed: type conflict, or no remaining candidate
- Stuck: cannot make any deduction.
- MaxDepthExceeded: unexpectedly high search depth.

Soundness

Typechecking: ML-typechecking followed by iterated symbol resolution.

Extracted program: the program obtained by replacing overloaded symbols with the values they resolved to.

Theorem (Type soundness)

If a program successfully typechecks, then the extracted program is well-typed in ML.

Soundness

Typechecking: ML-typechecking followed by iterated symbol resolution.

Extracted program: the program obtained by replacing overloaded symbols with the values they resolved to.

Theorem (Type soundness)

If a program successfully typechecks, then the extracted program is well-typed in ML.

Theorem (Non-ambiguity)

If our typechecking algorithm succeeds, then no other instantiation of the overloaded symbols leads to a well-typed ML program.

Implication of success and failure

Solvable program: a program for which exactly one instantiation of its overloaded symbols leads to a well-typed extracted program.

Implication of success and failure

Solvable program: a program for which exactly one instantiation of its overloaded symbols leads to a well-typed extracted program.

Theorem (Implication of a success)

If our algorithm successfully typechecks a program, then if finds the only instantiation of the overloaded symbols that leads to a well-typed ML program. In particular, the program is solvable.

Implication of success and failure

Solvable program: a program for which exactly one instantiation of its overloaded symbols leads to a well-typed extracted program.

Theorem (Implication of a success)

If our algorithm successfully typechecks a program, then if finds the only instantiation of the overloaded symbols that leads to a well-typed ML program. In particular, the program is solvable.

Theorem (Implication of a type error)

If our typechecking algorithm raises an error, then no instantiation of the overloaded symbols leads to a well-typed ML program. In particular, the program is not solvable.

Completeness

Iteratively solvable program: iff there exists an ordering $x_1, x_2, \ldots, x_n$ of the overloaded symbols such that, after resolving the $x_{k<i}$ it is possible to resolve x_i without inspecting the candidates for the $x_{k>i}$.
Assumption depth: maximal number of nesting of assumption in the resolution of a solvable program.

Theorem (Characterization of successful typechecking)

Our algorithm with parameter max-depth set to D successfully typechecks a program if and only if this program is iteratively solvable and with an assumption depth not exceeding D.

Details

Ask for our soon-to-be-submitted article for details.