A Mechanized First-Order Theory of Algebraic Data Types with Pattern Matching

Joshua M. Cohen

Princeton University

ITP 2025

ADTs and Pattern Matching are Widely Used

Functional Programming

Interactive Theorem Provers

SMT-Based Verifiers and IVLs

Compiling Pattern Matching and ADTs

Patterns Decision Trees

ADTs Pointers

Simple Patterns

Built-in

Simple Patterns

First-Order Formulas

Same pattern matching compilation algorithm can be used for all (non-dependent) cases!

Compiling Pattern Matching and ADTs

How do we know these compilation steps are correct?

P

ADTs

ormulas

Same pattern matching compilation algorithm can be used for all (non-dependent) cases!

Our Contributions

- We give the first verified general-purpose, real-world pattern matching compiler.
- We use this compiler to implement a verified exhaustiveness checker, extend proofs from the literature, and formulate a *robustness property* with which we discover an exhaustiveness-related bug in Why3.
- We use this to give the first formally proved-sound first-order axiomatization of ADTs.

Background

- Build on Why3Sem, Rocq formalization of Why3's logic [Cohen and Johnson-Freyd 2024]
- Why3's logic: First-order logic with polymorphism, ADTs, pattern matching, recursive functions, and inductive predicates

```
theory TreeForest
type list 'a = Nil | Cons  'a (list 'a)
type tree 'a = Leaf 'a | Node (tree 'a) (forest 'a)
with forest 'a = list (tree 'a)

use int.Int

function count_forest (f: forest int) : int =
  match f with
  | Nil -> 0
  | Cons t' f' -> count_tree t' + count_forest f'
  end
with count_tree (t: tree int) : int =
  match t with
  | Leaf i -> i
  | Node t' f' -> count_tree t' + count_tree f'
  end
```

Pattern Matching in Why3

$$
\begin{aligned}
p \coloneqq \quad & \mid \_ \\
& \mid x \\
& \mid c(p_1, \ldots, p_n) \\
& \mid p_1 | p_2 \\
& \mid p \; as \; x
\end{aligned}
$$

Pattern Matching in Why3

$$
\begin{aligned}
p \coloneqq \quad & \mid \_ \\
& \mid x \\
& \mid c(p_1, \ldots, p_n) \\
& \mid p_1 | p_2 \\
& \mid p \; as \; x
\end{aligned}
$$

Complicated!

- Nested matching
- Simultaneous matching
- Interactions with termination checking

Pattern Matching in Why3

$$
\begin{aligned}
p \coloneqq \quad & \mid \_ \\
& \mid x \\
& \mid c(p_1, \ldots, p_n) \\
& \mid p_1 | p_2 \\
& \mid p \; as \; x
\end{aligned}
$$

Complicated!

- Nested matching
- Simultaneous matching
- Interactions with termination checking

Why3Sem has pattern/matching

- Syntax
- Typing
- Semantics

Pattern Matching Compilation

```
match l1, l2 with
| [], [] -> x1
| [_], _ -> x2
| _ :: _, _ -> x3
| [], _ :: _ -> x4
end
```

```
match l1 with
| [] -> match l2 with
        | [] -> x1
        | y3 :: y4 -> x4
        end
| y1 :: y2 ->
        match y2 with
        | [] -> x2
        | y5 :: y6 -> x3
        end
end
```

Pattern Matching Compilation

```
match l1, l2 with
| [], [] -> x1
| [_], _ -> x2
| _ :: _, _ -> x3
| [], _ :: _ -> x4
end
```

```
match l1 with
| [] -> match l2 with
        | [] -> x1
        | y3 :: y4 -> x4
        end
| y1 :: y2 ->
        match y2 with
        | [] -> x2
        | y5 :: y6 -> x3
        end
end
```

```
match l1, l2 with
| [], [] -> x1
| [_], _ -> x2
| _ :: _, _ -> x3
end
```

Non-exhaustive!

Pattern Matching Compilation

- Widely applicable and well-studied problem: e.g. OCaml, Haskell, Rocq

[Augustsson 1985], [Baudinet and MacQueen 1985], [Laville 1988], [Puel and Suarez 1990], [Maranget 1992], [Pettersson 1992], [Sekar et al. 1995], [Sestoft 1996], [Scott and Ramsey 2000], [Le Fessant and Maranget 2001], [Maranget 2007], [Maranget 2008], [Karachalias 2015], [Tuerk et al. 2015]

Pattern Matching Compilation

- Widely applicable and well-studied problem: e.g. OCaml, Haskell, Rocq

[Augustsson 1985], [Baudinet and MacQueen 1985], [Laville 1988], [Puel and Suarez 1990], [Maranget 1992], [Pettersson 1992], [Sekar et al. 1995], [Sestoft 1996], [Scott and Ramsey 2000], [Le Fessant and Maranget 2001], [Maranget 2007], [Maranget 2008], [Karachalias 2015], [Tuerk et al. 2015]

Pattern Matching Compilation

- Widely applicable and well-studied problem: e.g. OCaml, Haskell, Rocq

[Augustsson 1985], [Baudinet and MacQueen 1985], [Laville 1988], [Puel and Suarez 1990], [Maranget 1992], [Pettersson 1992], [Sekar et al. 1995], [Sestoft 1996], [Scott and Ramsey 2000], [Le Fessant and Maranget 2001], [Maranget 2007], [Maranget 2008], [Karachalias 2015], [Tuerk et al. 2015]

Pattern Matching Compilation [Le Fassant and Maranget 2001, Maranget 2008]

```
match l1, l2 with
| [], [] -> x1
| [_], _ -> x2
| _ :: _, _ -> x3
| [], _ :: _ -> x4
end
```

$$\begin{pmatrix} nil & nil & x_1 \\ cons(\_, nil) & nil & x_2 \\ cons(\_, \_) & \_ & x_3 \\ nil & cons(\_, \_) & x_4 \end{pmatrix}$$

Pattern Matching Compilation [Le Fassant and Maranget 2001, Maranget 2008]

```
match l1, l2 with
| [], [] -> x1
| [_], _ -> x2
| _ :: _, _ -> x3
| [], _ :: _ -> x4
end
```

$$\begin{pmatrix} nil & nil & x_1 \\ cons(\_, nil) & nil & x_2 \\ cons(\_,\_) & \_ & x_3 \\ nil & cons(\_,\_) & x_4 \end{pmatrix}$$

$S(c, P)$: rest of match, assuming the first term matches constructor c

$$S(nil, P) = \begin{pmatrix} nil & x_1 \\ cons(\_,\_) & x_4 \end{pmatrix} \qquad S(cons, P) = \begin{pmatrix} \_ & nil & nil & x_2 \\ \_ & \_ & \_ & x_3 \end{pmatrix}$$

Pattern Matching Compilation [Le Fassant and Maranget 2001, Maranget 2008]

```
match l1, l2 with
| [_], _ -> x1
| _, _ :: _ -> x2
| _, _ -> x3
```

$$\begin{pmatrix} cons(\_, nil) & nil & x_1 \\ \_ & cons(\_, \_) & x_2 \\ \_ & \_ & x_3 \end{pmatrix}$$

Pattern Matching Compilation [Le Fassant and Maranget 2001, Maranget 2008]

```
match l1, l2 with
| [_], _ -> x1
| _, _ :: _ -> x2
| _, _ -> x3
```

$$\begin{pmatrix} cons(\_, nil) & nil & x_1 \\ \_ & cons(\_, \_) & x_2 \\ \_ & \_ & x_3 \end{pmatrix}$$

$D(P)$: rest of match, assuming the first term matches no constructor in column

$$D(P) = \begin{pmatrix} cons(\_, \_) & x_2 \\ \_ & x_3 \end{pmatrix}$$

Pattern Matching Compilation Algorithm

```
compile(ts, P)


match t1 with
| c1 (vs1) -> compile (vs1 ++ ts, S(c1, P))
| ...
| cn (vsn) -> compile (vsn ++ ts, S(cn, P))
| _ -> compile (ts, D(P))
```

Pattern Matching Compilation Algorithm

```
com
```

Nest terms in
defined order

```
match t1 with
| c1 (vs1) -> compile (vs1 ++ ts, S(c1, P))
| ...
| cn (vsn) -> compile (vsn ++ ts, S(cn, P))
| _ -> compile (ts, D(P))
```

Pattern Matching Compilation Algorithm

Each constructor in first column

Nest terms in defined order

```
com
...
atch t1 with
  c1 (vs1) -> compile (vs1 ++ ts, S(c1, P))
| ...
| cn (vsn) -> compile (vsn ++ ts, S(cn, P))
| _ -> compile (ts, D(P))
```

Pattern Matching Compilation Algorithm

Each constructor in first column

Nest terms in defined order

```
com
atch t1 with
  c1 (vs1) -> compile (vs1 ++ ts, S(c1, P))
| ...
| cn (vsn) -> compile (vsn ++ ts, S(cn, P))
| _ -> compile (ts, D(P))
```

Default case if missing constructor

Preprocessing

```
match l with
| x -> e
```

```
match l with
| _ -> let x = l in e
```

```
match l with
| p as x -> e
```

```
match l with
| p -> let x = l in e
```

```
match l1, l2 with
| (p1 | p2), p3 -> e
```

```
match l1, l2 with
| p1, p3 -> e
| p2, p3 -> e
```

An Optimization

```
match [x] with
| [y] -> z
| _ -> a
```

→

```
z
```

```
match (t1, t2) with
| (p1, p2) -> x1
| (p3, p4) -> x2
```

→

```
match t1, t2 with
| p1, p2 -> x1
| p3, p4 -> x2
```

Properties of Pattern Matching Compilation

- Termination
- Semantic Correctness
- Exhaustiveness Checking
- Robustness

1. Termination

```
match t1 with
| c1 (vs1) -> compile (vs1 ++ ts, S(c1, P))
| ...
| cn (vsn) -> compile (vsn ++ ts, S(cn, P))
| _ -> compile (ts, D(P))
```

1. Termination

```
match t1 with
| c1 (vs1) -> compile (vs1 ++ ts, S(c1, P))
| ...
| cn (vsn) -> compile (vsn ++ ts, S(cn, P))
| _ -> compile (ts, D(P))
```

Recurse on decompositions,
not subterms!

1. Termination

```
match l1, l2 with
| (p1 | p2), p3 -> e
```

⟹

```
match l1, l2 with
| p1, p3 -> e
| p2, p3 -> e
```

1. Termination

```
match l1, l2 with
| (p1 | p2), p3 -> e
```

⇒

```
match l1, l2 with
| p1, p3 -> e
| p2, p3 -> e
```

$$|P| = 1 + |p_1| + |p_2| + |p_3|$$

$$|P| = |p_1| + |p_2| + 2|p_3|$$

1. Termination

```
match l1, l2 with
| (p1 | p2), p3 -> e
```

```
match l1, l2 with
| p1, p3 -> e
| p2, p3 -> e
```

$$|P| = 1 + |p_1| + |p_2| + |p_3|$$

$$|P| = |p_1| + |p_2| + 2|p_3|$$

Matrix becomes larger in presence of “or” patterns!

1. Termination

$$\begin{pmatrix} nil & nil & x_1 \\ cons(\_, nil) & nil & x_2 \\ \_ & \_ & x_3 \\ nil & cons(\_, \_) & x_4 \end{pmatrix}$$

$$S(cons, P) = \begin{pmatrix} \_ & nil & nil & x_2 \\ \_ & \_ & \_ & x_3 \end{pmatrix}$$

1. Termination

$$\begin{pmatrix} nil & nil & x_1 \\ cons(\_, nil) & nil & x_2 \\ \_ & \_ & x_3 \\ nil & cons(\_,\_) & x_4 \end{pmatrix}$$

S matrix adds *k* wildcards for *k*-argument constructor

$$S(cons, P) = \begin{pmatrix} \_ & nil & nil & x_2 \\ \_ & \_ & \_ & x_3 \end{pmatrix}$$

1. Termination

$$|E^M(P)|_{b(P)+1}$$

1. Termination

$$|E^M(P)|_{b(P)+1}$$

Size of fully-expanded matrix

1. Termination

Constructor adds additional $b(P) + 1$ fuel

$$|E^M(P)|_{b(P)+1}$$

Size of fully-expanded matrix

1. Termination

Constructor adds additional $b(P) + 1$ fuel

$$|E^M(P)|_{b(P)+1}$$

Size of fully-expanded matrix

$b(P)$: number of rows × max arguments to any constructors in matrix

1. Termination

Constructor adds additional $b(P) + 1$ fuel

$$|E^M(P)|_{b(P)+1}$$

Size of fully-expanded matrix

$b(P)$: number of rows $\times$ max arguments to any constructors in matrix

- Expansion ($E^M(P)$) and $b(P)$ change throughout algorithm → need monotonicity results
- Prove *S* and *D* matrices decrease under $|E^M|$
- Results: prove *any* similar algorithm terminates

2. Semantics

If $compile(P, ts) = Some\ t$, then $\left[[ts]_v, P\right]_v = Some\ [t]_v$

If *compile* succeeds, original match also succeeds and is semantically equivalent to compiled term

2. Semantics

If $compile(P, ts) = Some\ t$, then $\left[[ts]_v, P\right]_v = Some\ [t]_v$

Semantics of matching matrix *P* against
term list *ts*

If *compile* succeeds, original match also succeeds and is semantically equivalent to compiled term

2. Semantics

Semantics of compiled term

If $compile(P, ts) = Some\ t$, then $\left[[ts]_v, P\right]_v = Some\ [t]_v$

Semantics of matching matrix *P* against term list *ts*

If *compile* succeeds, original match also succeeds and is semantically equivalent to compiled term

Detour: Why3 Semantics

- Hilbert-style (denotational) semantics

Interpretation of type, function, predicate symbols

Valuation of type and term variables

$$[g_1 \wedge g_2]_{I,v} = [g_1]_{I,v} \wedge [g_2]_{I,v}$$

Why3 “and”

Meta-logic (Rocq) “and”

- Recursive structures (types, functions) impose conditions on interpretations

Why3Sem: Algebraic Data Types

1. Constructors are injective: if $[c](t_1) = [c](t_2)$, then $t_1 = t_2$
2. Constructors are disjoint: if $[c_1](t_1) = [c_2](t_2)$, then $c_1 = c_2$
3. There is a (computable) function $\boldsymbol{find}$ that gives the constructor c and arguments t for any element x of ADT type such that
$$x = [c](t)$$
4. A generalized induction principle holds

Pattern matching: describe new bound variables, use $\boldsymbol{find}$ for constructors

2. Semantics

If $compile(P, ts) = Some\ t$, then $\left[[ts]_v, P\right]_v = Some\ [t]_v$

- Purely *semantic* reasoning – need to reason about ADTs, $\boldsymbol{find}(x)$, interpretation
- Existing proofs in literature – based on *syntactic* match relation on values
  - $c(v_1, \dots, v_n) \preccurlyeq c(p_1, \dots, p_n) \ \leftrightarrow \ \forall i, v_i \ \preccurlyeq p_i$
- Different than our setting: match semantics depends on interpretation!
- Idea: prove semantics of *S*, *D*, simplification, and compiled match

3. Exhaustiveness Checking

Corollary of semantic correctness:

If $\left[[ts]_v, P\right]_v = None$, then $compile(P, ts) = None$

- If no row in matrix matches terms, *compile* correctly reports “non-exhaustive”
- Augment Why3Sem type system with exhaustiveness check requiring $compile$ to be $Some$
- Other direction proved for cbv and lazy eval [Maranget 2007], interpretations make this tricky

4. Robustness

- Exhaustiveness check succeeds under reasonable changes to types, terms, patterns, etc
- E.g. substitution, alpha-conversion, rewriting, etc
- Problem: constructor optimization!

4. Robustness

```
match [1] with
| [x] → x
end
```

```
H: [1] = y
```

rewrite ← H

4. Robustness

```
match [1] with
| [x] → x
end
```

```
H: [1] = y
```

rewrite ← H

Please report:
Transformation rewrite <- l_eq failed:
Uncaught: File ... :
Pattern not covered by a match:
Nil

4. Robustness

- Found and reported bug to Why3 developers

How to fix?

- Cannot remove constructor optimization – need for tuples
- Create new version without optimization, use for exhaustiveness check, use old for compilation
- Prove new version satisfies robustness property
- Prove new version strictly stronger

ADT Axiomatization

1. Make types and constructors abstract
2. Introduce new abstract functions (projections, selectors, indexers) with axioms
3. Compile and eliminate pattern matching in all terms and formulas

```
(* Projections *)
function cons_proj_1 : list 'a → 'a
function cons_proj_2 : list 'a → list 'a
axiom cons_proj_1_def: ∀ u1 u2. cons_proj_1 (Cons u1 u2) = u1
axiom cons_proj_2_def: ∀ u1 u2. cons_proj_2 (Cons u1 u2) = u2

(* Selector *)
function match_list : list 'a → 'b → 'b → 'b
axiom match_list_cons: ∀ z1 z2 u1 u2. match_list (Cons u1 u2) z1 z2 = z1
axiom match_list_nil: ∀ z1 z2. match_list Nil z1 z2 = z2

(* Indexer *)
function index_list : list 'a → int
axiom index_list_cons: ∀ u1 u2. index_list (Cons u1 u2) = 0
axiom index_list_nil: index_list Nil = 1

(* Disjointness *)
axiom cons_nil: ∀ u1 u2. Cons u1 u2 <> Nil
(* Inversion *)
axiom list_inversion: ∀ u. u = Cons (cons_proj_1 u) (cons_proj_2 u) ∨ u = Nil
```

Eliminating Pattern Matching

- Axiomatized by selectors or represented as formula

```
match t with
| c(a) -> f
...
```

$(\forall a, t = c(a) \rightarrow f) \ \wedge \ \ldots$

$(\exists a, t = c(a) \wedge f) \ \vee \ \ldots$

$match\_list(t, f, \ldots)$

- Method (and proof) relies on simple patterns and results about shape of $compile$

Putting It All Together

- Define new context with new types and function symbols
- Construct interpretation for all added symbols
- Prove axioms satisfied by interpretation
- Prove pattern matching elimination preserves semantics
  - Note: patterns in hypotheses and goals - one direction not enough!
- Prove everything well-typed (tricky!)
- Result: sound first-order axiomatization of ADTs

Related Work

- Lots of work on pattern matching compilation, as we have seen
- Very little work on verified compilation
  - CakeML handles only simple patterns
  - MetaRocq (and CertiCoq) assumes patterns simple
- Sniper [Blot et al 2021; Blot et al 2023]: SMTCoq extension which turns Rocq goals into first-order formulas
  - Certifying: generates theorems and proof tactic scripts to validate
- IVLs without ADTs (e.g. Boogie, Viper) have certifying implementations

Conclusion

- Pattern match compilation is complicated!
- First *formally verified* sophisticated pattern matching compiler and first-order ADT axiomatization
- Key step in enabling verified verification tools (e.g. IVLs)
- Many of our results are more broadly applicable
    - Termination results for pattern matching compiler
    - Compiler general (e.g. allows any "return" type)
    - Could refactor proofs to remove dependence on Why3Sem
    - Other first-order axiomatizations of ADTs
- Thanks for listening!

References

Abrahamsson, O., Ho, S., Kanabar, H., Kumar, R., Myreen, M. O., Norrish, M. and Tan, Y. K. Proof Producing Synthesis of CakeML from Monadic HOL Functions. Journal of Automated Reasoning 64.7 (Oct 2020), 1287-1306.

Alain Laville. Implementation of lazy pattern matching algorithms. In Proceedings of the 2nd European Symposium on Programming, ESOP ’88, page 298–316, Berlin, Heidelberg, 1988. Springer-Verlag. ISBN 3540190279.

Besson, F. Itauto: An Extensible Intuitionistic SAT Solver. In 12th International Conference on Interactive Theorem Proving (ITP 2021) (Wroc law, Poland, 2021), L. Cohen and C. Kaliszyk, Eds., vol. 193 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 9:1–9:18.

Blanchette, J. C., Böhme, S., and Paulson, L. C. Extending Sledgehammer with SMT Solvers. In Automated Deduction – CADE-23 (Wroc law, Poland, 2011), N. Bjørner and V. Sofronie-Stokkermans, Eds., Springer, pp. 116–130.

Blot, V., Cousineau, D., Crance, E., de Prisque, L. D., Keller, C., Mahboubi, A., and Vial, P. Compositional Pre-processing for Automated Reasoning in Dependent Type Theory. In Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs (Boston, USA, Jan. 2023), CPP 2023, Association for Computing Machinery, pp. 63–77

Boulmé, S. Formally Verified Defensive Programming (Efficient Coq-verified Computations from Untrusted ML Oracles). PhD thesis, Université Grenoble-Alpes, Sept 2021.

Cohen, J. M., and Johnson-Freyd, P. A Formalization of Core Why3 in Coq. Proceedings of the ACM on Programming Languages 8, POPL (Jan. 2024), 60:1789–60:1818.

Czajka, L., and Kaliszyk, C. Hammer for Coq: Automation for Dependent Type Theory. Journal of Automated Reasoning 61, 1 (June 2018), 423–453.

Dardinier, T., Sammler, M., Parthasarathy, G., Summers, A. J., and Müller, P. Formal Foundations for Translational Separation Logic Verifiers. Proceedings of the ACM on Programming Languages 9, POPL (Jan. 2025), 20:569–20:599.

Ekici, B., Mebsout, A., Tinelli, C., Keller, C., Katz, G., Reynolds, A., and Barrett, C. SMTCoq: A Plug-In for Integrating SMT Solvers into Coq. In Computer Aided Verification (Heidelberg, Germany, 2017), R. Majumdar and V. Kunčak, Eds., Springer International Publishing, pp. 126–133.

References

Filliâtre, J.-C. One Logic to Use Them All. In Automated Deduction – CADE-24 (Lake Placid, New York, 2013), M. P. Bonacina, Ed., Springer, pp. 1–20.

Georgios Karachalias, Tom Schrijvers, Dimitrios Vytiniotis, and Simon Peyton Jones. Gadts meet their match: pattern-matching warnings that account for gadts, guards, and laziness. SIGPLAN Not., 50(9):424–436, aug 2015. ISSN 0362-1340. doi: 10.1145/2858949.2784748.

Gäher, L., Sammler, M., Jung, R., Krebbers, R., and Dreyer, D. RefinedRust: A Type System for High-Assurance Verification of Rust Programs. Proceedings of the ACM on Programming Languages 8, PLDI (June 2024), 192:1115–192:1139.

Jacobs, B., Vogels, F., and Piessens, F. Featherweight VeriFast. Logical Methods in Computer Science Volume 11, Issue 3 (Sept. 2015).

Kevin Scott and Norman Ramsey. When do match-compilation heuristics matter? Technical report, University of Virginia, USA, 2000.

Korkut, J., Stark, K., and Appel A. W. A Verified Foreign Function Interface Between Coq and C. Proceedings of the ACM on Programming Languages 9, POPL (Jan 2025), 24:687-24:717.

Laurence Puel and Ascander Suarez. Compiling pattern matching by term decomposition. In Proceedings of the 1990 ACM Conference on LISP and Functional Programming, LFP ’90, page 273–281, New York, NY, USA, 1990. Association for Computing Machinery. ISBN 089791368X. doi: 10.1145/91556.91670.

Le Fessant, F., and Maranget, L. Optimizing pattern matching. In Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (Florence, Italy, Oct. 2001), Association for Computing Machinery, pp. 26–37.

Lennart Augustsson. Compiling pattern matching. In Proc. of a Conference on Functional Programming Languages and Computer Architecture, page 368–381, Berlin, Heidelberg, 1985. Springer-Verlag. ISBN 3387159754.

Luc Maranget. Compiling lazy pattern matching. SIGPLAN Lisp Pointers, V(1):21–31, jan 1992. ISSN 1045-3563. doi: 10.1145/141478.141499.

References

Maranget, L. Compiling pattern matching to good decision trees. In Proceedings of the 2008 ACM SIGPLAN Workshop on ML (Victoria, British Columbia, Canada, Sept. 2008), ML ’08, Association for Computing Machinery, pp. 35–46.

Maranget, L. Warnings for pattern matching. Journal of Functional Programming 17, 3 (May 2007), 387–421.

Marianne Baudinet and David B. MacQueen. Tree pattern matching for ml., 1985. URL http://www.smlnj.org/compiler-notes/85-note-baudinet.ps.

Mikael Pettersson. A term pattern-match compiler inspired by finite automata theory. In Proceedings of the 4th International Conference on Compiler Construction, CC ’92, page 258–270, Berlin, Heidelberg, 1992. Springer-Verlag. ISBN 3540559841.

Nanevski,A., Morrisett, G., Shinnar, A., Govereau, P., and Birkedal, L. Ynot: Dependent types for imperative programs. In Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (Victoria, British Columbia, Canada, Sept 2008). ICFP '08, Association for Computing Machinery, pp 229-240.

Parthasarathy, G., Dardinier, T., Bonneau, B., Müller, P., and Summers, A. J. Towards Trustworthy Automated Program Verifiers: Formally Validating Translations into an Intermediate Verification Language. Proceedings of the ACM on Programming Languages 8, PLDI (June 2024), 208:1510–208:1534.

Parthasarathy, G., Müller, P., and Summers, A. J. Formally Validating a Practical Verification Condition Generator. In Computer Aided Verification (2021), A. Silva and K. R. M. Leino, Eds., Springer International Publishing, pp. 704–727.

Peter Sestoft. Ml pattern match compilation and partial evaluation. In Olivier Danvy, Robert Glück, and Peter Thiemann, editors, Partial Evaluation, pages 446–464, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-70589-5.

R. C. Sekar, R. Ramesh, and I. V. Ramakrishnan. Adaptive pattern matching. SIAM J. Comput., 24(6):1207–1234, dec 1995. ISSN 0097-5397. doi: 10.1137/S0097539793246252.

Sakaguchi, K. Program extraction for mutable arrays. Science of Computer Programming 191 (June 2020), 102372.

References

Sammler, M., Lepigre, R., Krebbers, R., Memarian, K., Dreyer, D., and Garg, D. RefinedC: Automating the foundational verification of C code with refined ownership types. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (June 2021), Association for Computing Machinery, pp. 158–174.

Sozeau, M., Forster, Y., Lennon-Bertrand, M., Nielsen, J., Tabareau, N., and Winterhalter, T. Correct and Complete Type Checking and Certified Erasure for Coq, in Coq. Journal of the ACM 72, 1 (Jan. 2025), 8:1–8:74.

Thomas Tuerk, Magnus O. Myreen, and Ramana Kumar. Pattern matches in HOL:. In Christian Urban and Xingyuan Zhang, editors, Interactive Theorem Proving, pages 453–468, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22102-1.

Zhou, L., Qin, J., Wang, Q., Appel, A. W., and Cao, Q. VST-A: A Foundationally Sound Annotation Verifier. Proceedings of the ACM on Programming Languages 8, POPL (Jan. 2024), 69:2069–69:2098.

Extra Slides

ADT Axiomatization

Give interpretations for new function symbols in Rocq:

```
Definition indexer_interp {m a} (al: arg_list ...):=
(*Cast head of al to [adt_rep]*)
let x := indexer_args_eq a al ... in
(*Use find function*)
let (c1, _) := find_constr_rep m a x in
(*Find index of c1 in a's constructor list*)
dom_cast ... (Z.of_nat (index c1 (adt_constr_list a))).
```

Prove axioms satisfied by this interpretation

Method applied to Why3’s axiomatization, but could easily be extended to others (e.g. Dafny)