Autoformalizing Fast–Slow Equivalence Proofs for Atlas Computations in the Unitary Dual Problem

Mario Carneiro[1], Stephen D. Miller[2], Magnus Myreen[1], and Josef Urban[1,3]

[1]Chalmers University and University of Gothenburg [2]Yeshiva University [3]AI4REASON

Goal The goal of this project is the formal verification of optimized computations arising in the unitary dual problem. The motivating case is an atlas workflow for determining all unitary representations of a Lie group with infinitesimal character in the fundamental parallelepiped. The production code uses many hand-written optimizations, implemented partly in the Axis scripting layer, while a slower brute-force checker gives a simpler reference computation. We report on an ongoing `.at`-to-Standard ML translation and a staged HOL4/CakeML verification stack (about 500 commits) aimed at proving this fast–slow equivalence for the case of the Lie group $F_{4(4)}$, and scaling to larger computations of interest such as for $E_{8(8)}$.

Mathematical motivation: in what ways can a group action preserve norms? Lie groups and their representations (namely, the ways they act on vector spaces) appear ubiquitously in mathematics and its applications. It was recognized already in the 1930s in Gelfand's work on dynamical systems that *unitary* representations – those that preserve a norm on a Hilbert space – play a fundamental and special role, as they were already known to do in quantum mechanics. Nearly a century later, this role has spread to number theory as well, since the highly mysterious and fascinating automorphic representations are always unitary. It has been known for over half a century how to construct and classify representations in general, but the *unitary dual problem* – to completely describe the irreducible unitary represenations – remains as the last major unsolved problem in the representation theory of Lie groups [7].

The unitary dual of select examples of Lie groups has been computed, to much fanfare: perhaps most notable are David Vogan's landmark results for $SL(n,\mathbb{R})$ [11] and the split exceptional Lie group $G_{2(2)}$ [12]. The answers in these cases are subtle and intricate, and defy extrapolation to elsewhere. One particular group stands out as the most difficult challenge, being the largest split real group not part of an infinite family: $E_{8(8)}$, which has dimension 248. Over the last few decades progress has been made from several approaches: applying ideas from geometric representation theory [2], string theory injected through number theory [6], and Hodge theory [10, 4], for example has led to results not seen directly from the classical representation-theoretic viewpoint alone. At the same time, the atlas project [3] created software with an `is_unitary` command that implements the algorithm in [2] to decide unitarity, given enough resources.

Combining methods, Adams, van Leeuwen, Miller, and Vogan have recently computed a draft candidate for the unitary dual of $E_{8(8)}$. These lengthy calculations navigated through a maze of computational constraints and bottlenecks, often using individual optimizations that required proving new results in representation theory. At this stage, it has now become a verification problem, to eliminate the risk that one of these optimizations in their scripts – be it perhaps in how cases are enumerated, hashed, or pruned – changes the computed set. The goal of our present project is precisely to verify these optimizations.

Background: the atlas software The atlas project [3] has developed powerful, specialized software for computations in representation theory; it has been specifically designed with the unitary dual problem in mind. The core of the software consists of C++ libraries, which use custom data structures geared for Lie theory. In addition, numerous scripts in the Axis interpreter language are available that expand upon the libraries, but which are much slower. Much of the organization of the unitary dual project (enumeration, hashing, and deciding which of many possible optimizations to execute) is handled at the level of scripts.

Our plan: replacement architecture in SML - coded by Codex We replace (in almost automated Codex sessions) Axis (`.at`) scripts by Standard ML programs using Poly/ML [9]. A C ABI shim exposes a curated subset of the atlas C++ API as a shared library, which is then bound in Standard ML using the Poly/ML foreign-function interface. The design keeps (i) the trusted semantic core in C++ and (ii) the script-level control flow in Standard ML. This makes the control flow and data structures available for translation to CakeML [8] and reasoning/verification in HOL4 [5], while keeping the large C++ code base behind explicit

assumptions. The point is not to replace atlas mathematics, but to put the fragile optimization layer into a form where equivalence theorems can be stated, generated, reviewed, and ultimately proved. The Codex coding was supervised by us and used unit testing based on F_4 etc.

Case study: F_4 FPP unitary dual verification Our principal target is the atlas script `script_to_verify_F4_FPP_unitary_dual.at`. It checks a finite set of atlas parameters associated with facet or bottom-layer points of a fundamental parallelepiped for the split real form $F_{4(4)}$. In representation theoretic terms, these parameters describe candidate pieces of the unitary dual inside a fixed finite region; in verification terms, they form the concrete finite domain over which fast and slow computations should agree.

Our Codex-translated Standard ML entry point `script_to_verify_F4_FPP_unitary_dual.sml` runs a fast pipeline that: (a) enumerates candidates using atlas KGB elements, a precomputed lambda table, and folded barycenters; (b) constructs and normalizes parameters via FFI calls; (c) filters by standard, final, hermitian, and unitary predicates; and (d) deduplicates using a bucketed hash set, `ParamHash`, implemented in Standard ML using atlas equality and hashing. The expected output set size for $F_{4(4)}$ is 1864, which is checked as a regression invariant.

The corresponding slow program, `SimplerVerifyF4FPP.sml` is a direct translation of a brute-force checker that ranges over a large triple domain of (x, λ, ν). Here x ranges over KGB elements, λ over the relevant integral or rational spectral parameters, and ν over barycenters of facets in the fundamental parallelepiped decomposition. For each triple the slow checker constructs the associated parameter, finalizes it using atlas, tests unitarity, and reports a counterexample if the resulting unitary final term is missing from the fast set.

The ultimate theorem: fast–slow equivalence The theorem we want is an optimization-correctness theorem. At a high level, we define:

$$\begin{aligned} D_{\mathsf{slow}}(g) &= \{(x,\lambda,\nu) \mid x \in \mathrm{KGB}(g),\ \lambda \in \mathrm{FPP\_lambdas}(g,x),\ \nu \in \mathrm{AllBarycenters}(g)\}, \\ U_{\mathsf{slow}}(g,\mathrm{dom}) &= \{\,\pi \mid \exists t \in \mathrm{dom}.\ \mathrm{firstFinal}(\mathrm{mkParam}(g,t)) = \pi \ \wedge\ \mathrm{isUnitary}(\pi)\,\}, \\ U_{\mathsf{fast}}(g) &= \text{the set represented by the fast program's ParamHash.} \end{aligned}$$

Because the implementation uses the atlas semantic equality `atlas_param_equal` rather than pointer identity, the final statement is naturally phrased using a relation $\approx_{\mathsf{atlas}}$ and its induced set equivalence. Informally, for $F_{4(4)}$:

$$U_{\mathsf{fast}}(F_{4(4)}) \approx_{\mathsf{atlas}} U_{\mathsf{slow}}(F_{4(4)}, D_{\mathsf{slow}}(F_{4(4)})) \quad \text{and} \quad \mathrm{BottomLayerOK}(F_{4(4)}, U_{\mathsf{fast}}(F_{4(4)})).$$

The first conjunct says that the optimized hash-based computation has not lost or added any unitary final terms compared with the slow reference algorithm. The second records the additional bottom-layer invariant checked by the original scripts. Together they connect the mathematical motivation to the formal task: if the atlas primitives have their specified meanings, then the optimized script-level computation is a faithful implementation of the simple finite search arising from the unitary dual problem.

Autoformalization: a top-down refinement stack in HOL4 The main challenge is that both programs depend on many external atlas C++ functions. We therefore structure the proof as a refinement stack. First, HOL4 theories define the abstract sets and predicates above, treating atlas operations as uninterpreted functions equipped with explicit contracts. Second, we prove algorithmic skeleton lemmas about the slow checker as a list fold searching for missing witnesses. Third, we reduce the fast checker to obligations about domain pruning and witnessing, correctness of the `ParamHash` set view, and correctness of the bottom-layer traversal. Finally, we keep an explicit inventory of the remaining bridge obligations that connect actual Standard ML execution to the abstract predicates.

Auto-translation and AI assistance The `.at`-to-Standard ML translation is large and repetitive, but sensitive to subtle semantic details, including coordinate conventions, normalization, finalization, and equality. We use an agentic workflow to translate scripts into a compilable and documented Standard ML module graph, add small FFI bindings and tests as needed, and generate the HOL4/CakeML refinement skeletons and goal inventories. The intended output is not merely a working Standard ML replacement for a legacy script, but an automatically maintained specification of what has been proved and what remains as an explicit contract.

References

[1] J. Adams, D. Barbasch, and D. A. Vogan, Jr. *The Langlands Classification and Irreducible Characters for Real Reductive Groups*. Progress in Mathematics 104, Birkhäuser, 1992.

[2] J. Adams, M. van Leeuwen, P. Trapa, and D. Vogan. *Unitary representations of real reductive groups*. Astérisque 417, Société Mathématique de France, 2020. Preprint version: arXiv:1212.2192.

[3] J. Adams, M. van Leeuwen, D. Vogan, and collaborators. The Atlas of Lie Groups and Representations. https://www.liegroups.org/.

[4] D. Davis and L. Mason-Brown. *The FPP Conjecture for Real Reductive Groups*. Preprint, 2024. arXiv:2411.01372.

[5] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[6] M. B. Green, S. D. Miller, and P. Vanhove. *Small representations, string instantons, and Fourier modes of Eisenstein series*. *Journal of Number Theory* **146** (2015), 187–309. With an appendix by D. Ciubotaru and P. Trapa. Preprint version: arXiv:1111.2983.

[7] A. W. Knapp. *Representation Theory of Semisimple Groups: An Overview Based on Examples*. Princeton University Press, 2001.

[8] R. Kumar, M. O. Myreen, M. Norrish, and S. Owens. CakeML: a verified implementation of ML. In *Proceedings of POPL 2014*, pages 179–191, 2014.

[9] D. C. J. Matthews. Poly/ML. https://polyml.org/.

[10] W. Schmid and K. Vilonen. *Hodge theory and unitary representations of reductive Lie groups*. *Frontiers of Mathematical Sciences*, International Press, Somerville, MA, 2011, pp. 397–420. Preprint version: arXiv:1206.5547.

[11] D. A. Vogan, Jr. *Unitarizability of certain series of representations*. Annals of Mathematics **120** (1984), 141–187.

[12] D. A. Vogan, Jr. *The unitary dual of G_2*. Inventiones Mathematicae **116** (1994), 677–791.

A Translator plan: discharging imperative obligations in CakeML

The most proof-intensive pure component is the hash-set representation used by the fast program. We develop a CakeML monadic model of `ParamHash` and prove its representation and invariant lemmas: lookup and insert correctness, bucket discipline, and membership modulo $\approx_{\mathsf{atlas}}$. The CakeML translator provides a route to connect executable ML code to these lemmas, and the HOL4 refinement stack is designed so that this evidence can be plugged in without changing the top-level statement.

B Status and outlook

The fast Standard ML verifier for $F_{4(4)}$ is functional and checks the expected set of 1864 parameters. The slow checker is translated and supports bounded sampling. On the formal side, the end-to-end theorem shape is fixed, the key list/set skeleton lemmas are in place, and the remaining assumptions are isolated as explicit contracts and bridge goals. Next steps are to strengthen the contracts around atlas equality and hashing, discharge `ParamHash` obligations via CakeML proofs, and replace bridge assumptions by proved statements or validated external specifications. The longer-term goal is to scale from the F_4 test case to the larger rank-eight computations, where running the slow checker directly is infeasible but verifying the correctness of the optimizations is most valuable.

C Reference and optimized Atlas workflows

This appendix records the computational shape motivating the theorem above. For the F_4 test case, the optimized workflow loads precomputed expected unitary parameters and runs the

main atlas command

```
FPP_unitary_hash_bottom_layer(G).
```

The command computes the relevant bottom-layer contribution to the unitary dual and stores the result in a hash table. A short wrapper can turn this into a Boolean verification script by checking that no new unitary representation is found beyond the expected list.

The slow reference workflow is a direct exhaustive checker. In schematic form, it is:

```
set G=F4_s
set all_barycenters_of_facets = FPP_barycenters(G,-1).##
<F4_FPP_points.at

for x in KGB(G) do
  for lambda in FPP_lambdas(x) do
    for nu in all_barycenters_of_facets do
      let pi = first_param(finalize(parameter(x,lambda,nu))) in
        if is_unitary(pi)=true
           and big_unitary_hash.long_match(pi,0)=-1
        then print("IT'S ALL WRONG!!!") fi
    od
  od
od
```

The point of this script is not speed but clarity: it directly loops over the parameter data (x, λ, ν), finalizes the corresponding representation, tests unitarity, and checks membership in the expected set. Some parts of the current slow script, notably the generation of `FPP_lambdas`, may still use mild pre-processing. In the proof architecture this appears as a separate domain-completeness obligation: the lambda generator must cover every parameter that the mathematical reference domain requires.

D Why the verification target is optimization correctness

The full unitary-dual computation uses exact arithmetic and mathematically specified tests, but the practical implementation differs substantially from the naive enumeration. The optimized code may avoid facets, reuse hash-table information, exploit nonunitarity implications, or replace a direct test by a mathematically equivalent shortcut. Each optimization is intended to preserve the final set, but each also creates a possible implementation error.

The theorem we seek is therefore not primarily a compiler-correctness theorem for C++ or a formalization of all background representation theory. It is a script-level equivalence theorem:

$$\forall G \in \mathcal{G}. \quad U_{\mathsf{fast}}(G) \;\approx_{\mathsf{atlas}}\; U_{\mathsf{slow}}(G, D_{\mathsf{slow}}(G)),$$

where $\mathcal{G}$ is the relevant family of real forms. For the initial case study $\mathcal{G} = \{F_{4(4)}\}$. For the eventual rank-eight computation, $\mathcal{G}$ includes the larger exceptional groups and the smaller groups that appear in the inductive structure of the classification.

E Representative proof obligations

The current refinement stack separates the work into the following obligations.

1. **FFI contracts.** The Standard ML bindings for atlas parameter construction, finalization, unitarity, equality, and hashing implement the specified abstract operations.
2. **Semantic equality.** The relation $\approx_{\mathsf{atlas}}$ is an equivalence relation and is respected by hash-table lookup and insertion.
3. **Hash-set correctness.** `ParamHash` represents a finite set of parameters modulo $\approx_{\mathsf{atlas}}$, and its operations preserve the representation invariant.

4. **Slow checker correctness.** If the slow checker terminates without printing a counterexample, then every unitary final term in the slow domain is present in the expected set.

5. **Fast checker correctness.** The fast pipeline's enumeration, pruning, and bottom-layer traversal produce exactly the same semantic set as the abstract optimized specification.

6. **Optimization equivalence.** Each pruning or shortcut used by the fast specification is equivalent to the corresponding part of the slow reference search.

This decomposition is designed to make progress possible even when the full large computation cannot be replicated locally. Small cases such as G_2 and F_4 can be executed as regression tests, while the formal proof treats the large cases parametrically through the same contracts and equivalence lemmas.