Improvements Plan

This document analyzes the current implementation against PLAN.md and identifies areas for improvement.

Executive Summary

The linter has successfully completed **Phase 1** (Foundation Upgrade) and partially completed **Phase 2** (Enhanced Analysis). Key improvements are needed in:

1. **Full HIE file parsing** (currently stubbed)
2. **Test coverage** (missing property tests and edge case tests)
3. **GHC Plugin for Template Haskell** (Phase 3 not started)
4. **Advanced refactoring features** (Phase 4 not started)
5. **Editor integrations** (Phase 5 not started)

Current Implementation Status

Phase 1: Foundation Upgrade ✅ COMPLETE

Feature	Status	Notes
ghc-lib-parser migration	✅ Done	Replaced haskell-src-exts
Package-qualified imports	✅ Done	Using `{-# LANGUAGE PackageImports #-}`
ghc-exactprint integration	✅ Done	Module exists, basic functionality
Word-boundary pattern matching	✅ Done	`matchPattern` respects identifiers
Word-boundary substitution	✅ Done	`substituteExpr` doesn't corrupt identifiers
Format-preserving replacements	⚠️ Partial	Basic implementation, needs more testing

Phase 2: Enhanced Analysis ⚠️ PARTIAL

Feature	Status	Notes
HIE file discovery	✅ Done	`findHieFiles` works
HIE file parsing	❌ Stubbed	Returns empty symbols/references
Dependency graph	⚠️ Partial	Structure exists, needs HIE data
Unused code detection	⚠️ Partial	Algorithm exists, needs real HIE data
Cross-module analysis	❌ Not working	Requires full HIE parsing
Type-aware patterns	❌ Not working	Requires HIE type info

Phase 3: Template Haskell Support ❌ NOT STARTED

Feature	Status

GHC Plugin skeleton	❌ Not implemented
TH splice tracking	❌ Not implemented
Integration with dep graph	❌ Not implemented

Phase 4: Advanced Refactoring ❌ NOT STARTED

Feature	Status
Pattern equation parser	❌ Not implemented
Retrie-style rules	❌ Not implemented
Batch refactoring mode	❌ Not implemented

Phase 5: Polish & Integration ⚠️ PARTIAL

Feature	Status	Notes
Terminal output	✅ Done	Full color support
JSON output	✅ Done	Working
SARIF output	✅ Done	Full v2.1.0 implementation
HTML reports	❌ Not implemented	
LSP server	❌ Not implemented	
VSCode extension	❌ Not implemented	
GitHub Action	❌ Not implemented	

Priority Improvements

HIGH PRIORITY

1. Complete HIE File Parsing

Current State: `loadHieFile` in `src/Linter/Analysis/Semantic.hs:92-117` returns empty data.

Problem: The HIE parsing was simplified due to GHC API compatibility issues between `ghc` and `ghc-lib-parser` packages.

Solution Options:

a) **Use hiedb library** (Recommended)

- Add `hiedb >= 0.6` to dependencies
- It handles GHC version compatibility internally

```
import HieDb

loadHieFile :: FilePath -> IO (Maybe HieData)
```

```
loadHieFile path = do
  result <- readHieFile path
  -- Convert HieFile to our HieData type
```

b) **Match exact GHC version**

- Pin to a specific GHC version where ghc-lib-parser aligns
- More fragile but simpler

Files to modify:

- `src/Linter/Analysis/Semantic.hs`
- `package.yaml` (add hiedb dependency)

2. Add Property-Based Tests

Current State: Only unit tests in `test/Spec.hs`, no QuickCheck properties.

Missing Tests:

```
-- Round-trip property: parse then print equals original
prop_parseRoundTrip :: Property
prop_parseRoundTrip = forAll genValidHaskell $ \code ->
  let parsed = parseFile code
      printed = printModule parsed
  in printed == code

-- Pattern matching properties
prop_matchIdentifierBoundary :: Property
prop_matchIdentifierBoundary = forAll genIdentifier $ \ident ->
  let prefixed = "prefix" <> ident
      suffixed = ident <> "suffix"
  in not (matchIdentifier ident prefixed) &&
     not (matchIdentifier ident suffixed)

-- Substitution idempotence
prop_substituteIdempotent :: Property
prop_substituteIdempotent = forAll genSubstitution $ \(from, to, text) ->
  let once = substituteExpr (Map.singleton from to) text
      twice = substituteExpr (Map.singleton from to) once
  in once == twice
```

Files to create/modify:

- `test/Spec.hs` (add properties)
- `test/Generators.hs` (new file for QuickCheck generators)

3. Fix Word Boundary Check in `replaceWordBoundary`

Current State: `src/Linter/Refactor/Substitution.hs:106-129`

Issue: The current implementation only checks the end boundary, not the start:

```
-- Line 118-119: beforeChar is unused!
beforeChar = Nothing  -- We check word start differently
```

```
isWordBoundary = isWordEnd afterMatch
```

Fix:

```
replaceWordBoundary :: Text -> Text -> Text -> Text
replaceWordBoundary from to text =
  T.pack $ go Nothing (T.unpack text) (T.unpack from) (T.unpack to)
  where
    go :: Maybe Char -> String -> String -> String -> String
    go _ [] _ _ = []
    go prev haystack@(h:hs) fromStr toStr
      | fromStr `isPrefixOf` haystack && isWordBoundary =
          toStr ++ go (Just (last toStr)) (drop (length fromStr) haystack) fromStr
toStr
      | otherwise = h : go (Just h) hs fromStr toStr
      where
        afterMatch = drop (length fromStr) haystack
        isWordBoundary = isWordStart prev && isWordEnd afterMatch

    isWordStart Nothing = True  -- Start of string
    isWordStart (Just c) = not (isIdentChar c)

    isWordEnd [] = True
    isWordEnd (c:_) = not (isIdentChar c)

    isIdentChar c = isAlphaNum c || c == '_' || c == '\''
```

MEDIUM PRIORITY

4. Add Test Coverage for Edge Cases

Missing test cases:

```
-- Pattern matching edge cases
describe "Pattern edge cases" $ do
  it "handles empty patterns" $ do
    matchPattern "" "anything" `shouldBe` False

  it "handles unicode identifiers" $ do
    matchPattern "αβγ" "test αβγ test" `shouldBe` True
    matchPattern "αβγ" "αβγδ" `shouldBe` False  -- Not at word boundary

  it "handles operators" $ do
    matchPattern ">>=" "x >>= f" `shouldBe` True

  it "handles qualified names" $ do
    matchPattern "Data.head" "Data.head xs" `shouldBe` True
    matchPattern "head" "Data.head xs" `shouldBe` False  -- Qualified

-- Substitution edge cases
describe "Substitution edge cases" $ do
```

```
  it "preserves string literals" $ do
    substituteExpr (Map.singleton "head" "headMay")
      "\"head of the list\""
      `shouldBe` "\"head of the list\""  -- Don't replace in strings

  it "preserves comments" $ do
    substituteExpr (Map.singleton "head" "headMay")
      "-- Use head carefully"
      `shouldBe` "-- Use head carefully"
```

5. Add TOML Configuration Support

Current State: Only YAML config supported via `loadLegacyConfig`.

PLAN.md specifies TOML as the new format:

```
[naming]
enabled = true

[[naming.types]]
pattern = "*Id"
replacement = "Key {1}"
```

Implementation needed:

- Add `tomland >= 1.3` (already in package.yaml but not used)
- Create TOML parser in `src/Linter/Config.hs`

6. Improve Error Messages

Current State: Parse errors are basic.

Improvement:

```
-- Add context to diagnostics
data Diagnostic = Diagnostic
  { diagSpan :: SrcSpan
  , diagSeverity :: Severity
  , diagMessage :: Text
  , diagContext :: Maybe Text     -- NEW: surrounding code
  , diagSuggestions :: [Text]     -- NEW: potential fixes
  , diagRelatedInfo :: [(SrcSpan, Text)]  -- NEW: related locations
  }
```

LOW PRIORITY

7. Add Performance Benchmarks

Missing:

- Criterion benchmarks for parsing
- Memory usage tracking
- Large file stress tests

```haskell
-- benchmark/Main.hs
main :: IO ()
main = defaultMain
  [ bgroup "parsing"
      [ bench "small file" $ nfIO (parseFile "test/data/Small.hs")
      , bench "large file" $ nfIO (parseFile "test/data/Large.hs")
      ]
  , bgroup "pattern matching"
      [ bench "simple" $ nf (matchPattern "head") "head xs"
      , bench "wildcard" $ nf (matchPattern "* $ head *") "f $ head xs"
      ]
  ]
```

8. Add HTML Output Format

PLAN.md specifies HTML reports for visual analysis.

Implementation: Create `src/Linter/Output/Html.hs` with:

- Summary statistics
- File-by-file breakdown
- Syntax-highlighted code snippets
- Expandable fix previews

9. Create GHC Plugin (Phase 3)

This is the major missing feature for TH support.

Structure needed:

```
linter-plugin/
├── src/
│   └── LinterPlugin.hs
├── linter-plugin.cabal
└── README.md
```

Basic implementation:

```haskell
{-# LANGUAGE OverloadedStrings #-}
module LinterPlugin (plugin) where

import GHC.Plugins
import GHC.Tc.Types

plugin :: Plugin
plugin = defaultPlugin
  { renamedResultAction = trackReferences
  , spliceRunAction = trackThSplices
  , pluginRecompile = purePlugin
  }

trackThSplices :: [CommandLineOption] -> LHsExpr GhcTc -> TcM (LHsExpr GhcTc)
trackThSplices _ expr = do
```

```
  let refs = collectNames expr
  liftIO $ appendFile ".linter-th-refs" (show refs ++ "\n")
  return expr
```

Code Quality Issues

1. Duplicate Code

Location: `isIdentChar` is defined in multiple files:

- `src/Linter/Rules/Patterns.hs:91`
- `src/Linter/Rules/Patterns.hs:139`
- `src/Linter/Refactor/Substitution.hs:129`
- `src/Linter/Core.hs:311`

Fix: Extract to `src/Linter/Types.hs` or create `src/Linter/Utils.hs`:

```haskell
-- src/Linter/Utils.hs
isIdentChar :: Char -> Bool
isIdentChar c = isAlphaNum c || c == '_' || c == '\''

isWordBoundary :: Maybe Char -> Maybe Char -> Bool
isWordBoundary before after =
  maybe True (not . isIdentChar) before &&
  maybe True (not . isIdentChar) after
```

2. Incomplete Pattern Matches

Warning-prone code in `matchWildcardPattern`:

```haskell
-- src/Linter/Rules/Patterns.hs:137
(Prelude.head afterMatch)  -- Unsafe!
```

Fix: Use pattern matching or safe-head:

```haskell
case afterMatch of
  [] -> True
  (c:_) -> not (isIdentChar c)
```

3. Missing HLint Configuration

Create `.hlint.yaml`:

```yaml
- ignore: {name: "Use camelCase"}  # We have our own naming rules
- warn: {name: "Use foldl'"}
- error: {name: "Avoid fromJust"}
- error: {name: "Avoid head"}
```

Test Coverage Gaps

Current Coverage (Estimated)

Module	Coverage	Notes
Linter.Config	~60%	Missing TOML tests
Linter.Analysis.Syntactic	~40%	Missing edge cases
Linter.Analysis.Semantic	~10%	Stubbed implementation
Linter.Rules.Naming	~70%	Good coverage
Linter.Rules.Patterns	~50%	Missing wildcard tests
Linter.Refactor.Substitution	~30%	Missing boundary tests
Linter.Core	~50%	Basic integration
Linter.Output.*	~20%	Output not tested

Tests to Add

```
-- 1. Semantic analysis tests (once HIE works)
describe "Linter.Analysis.Semantic" $ do
  it "loads real HIE files" $ do
    hies <- loadHieFiles ".hie"
    hies `shouldSatisfy` (not . null)

  it "extracts symbols from HIE" $ do
    Just hie <- loadHieFile ".hie/Linter/Types.hie"
    extractSymbols hie `shouldSatisfy` (not . null)

-- 2. Unused code detection tests
describe "Linter.Analysis.Unused" $ do
  it "detects unused functions" $ do
    let graph = buildTestGraph
    let unused = detectUnused defaultConfig graph []
    urUnreachable unused `shouldContain` "unusedHelper"

-- 3. Output format tests
describe "Linter.Output.Sarif" $ do
  it "produces valid SARIF JSON" $ do
    let result = mockAnalysisResult
    let sarif = renderSarif defaultOptions result
    decode sarif `shouldSatisfy` isJust  -- Validates JSON structure
```

Recommended Action Plan

Immediate (This Week)

1. [ ] Fix `replaceWordBoundary` to check start boundary
2. [ ] Add property tests for pattern matching
3. [ ] Extract duplicate `isIdentChar` to shared module
4. [ ] Add missing edge case tests

Short-term (2-4 Weeks)

1. [ ] Integrate `hiedb` for proper HIE parsing
2. [ ] Implement TOML configuration parser
3. [ ] Add output format tests
4. [ ] Create performance benchmarks

Medium-term (1-2 Months)

1. [ ] Implement GHC plugin for TH tracking
2. [ ] Add HTML output format
3. [ ] Create LSP server skeleton
4. [ ] Improve error messages with context

Long-term (3+ Months)

1. [ ] Complete Phase 3 (TH support)
2. [ ] Complete Phase 4 (Retrie-style refactoring)
3. [ ] VSCode extension
4. [ ] GitHub Action for CI/CD

Success Metrics

From PLAN.md, we should track:

- [ ] All current test cases pass ✅
- [ ] No false positives for TH-used code ❌ (needs GHC plugin)
- [ ] Format preservation: `parse . print = id` ⚠️ (needs property tests)
- [ ] Cross-module unused detection working ❌ (needs HIE)
- [ ] < 1 second for quick mode on single file ⚠️ (needs benchmarks)
- [ ] < 30 seconds for full analysis on 10k LOC ⚠️ (needs benchmarks)
- [ ] Comprehensive test suite (> 80% coverage) ❌ (~40% estimated)
- [ ] Property-based tests for core functions ❌
- [ ] Integration tests with real codebases ⚠️ (limited)

Last updated: 2024