Source Code Origination - OpenTofu's Implementation of the Removed Block

Preface

The functionality of the "removed" block in OpenTofu was developed not based on the similarly named, but functionally different Terraform feature, but rather on the already existing "moved" block in both the OpenTofu and Terraform codebase under the MPL-2.0 license. While we do not have information about the HashiCorp development process, it is likely, that much of the code in Terraform also originates from the "moved" block implementation and is, in large parts, not an original work of authorship but rather a derivative work. This is supported by the following comment in the Terraform codebase:

```
// Like MoveEndpoint, RemoveTarget is a wrapping struct that captures the result
// of decoding an HCL traversal representing a relative path from the current
// module to a removeable object.
```

To accurately show the origin of the code, in this document we will show the differences and similarities by logical code blocks rather than line by line, which can be misleading since the order of functions and code blocks influences the differential view.

We believe that this is just a case of a misunderstanding where the code came from.

On the matter of using BUSL-1.1-licensed code in OpenTofu

The OpenTofu team never has and will not copy, and never has and will not knowingly accept copies of BUSL-1.1-licensed code into the OpenTofu repository.

On the matter of monitoring and accepting third-party submissions

On or about January [redacted], 2024 a community contributor submitted a pull request (see [redacted]) that was a direct copy of a pull request filed against the Terraform repository (see [redacted]). The contributor did not have any rights to the pull request against the Terraform repository and the Terraform contributor licensed their contribution under the BUSL-1.1 license, which the OpenTofu team discovered and promptly rejected.

As a follow-up to this event, the OpenTofu core team instituted the policy of the "taint team". If a pull request mirroring a Terraform feature is submitted by a contributor, one core team member is dedicated as the "taint team" (typically the first team member reviewing the PR). This team member compares the code in the Terraform PR and the OpenTofu PR. If the PR is found to be likely in breach of intellectual property rights, the pull request is closed and the contributor is barred from working on that area of the code in the future. Multiple offenders are banned from contributing to the OpenTofu repository as a whole. Core team members participating in such a review regularly recuse themselves from future work on the reviewed PR because they are unable to provide an unbiased review.

As a result of this process, a core team member flagged the following PR: [redacted] as being very similar to the Terraform PR [redacted] based on the highly specific and non-trivial go-cty handling code present in the PR. Multiple team members were pulled into the review, and even though unsure, the PR was rejected on the grounds that it may be a copy. The reason for the closure was privately communicated to the contributor.

On the matter of educating contributors

On or about January [redacted], 2024 a [redacted] user highlighted the similarities between the implementation of the "removed" block between OpenTofu and Terraform (see [redacted]). The OpenTofu team took the allegations of possible copyright infringement by one of its own very seriously and promptly conducted a thorough review of the code also presented in this document. We found no wrongdoing as both implementations are based on the "moved" block rather than on each other. The OpenTofu code is functionally different from the Terraform code, contains more and diverging tests and the similarities are attributable to the "moved" block under the MPL-2.0 license.

Following the events described above, the core team instituted a change to the contribution guidelines presented in the main OpenTofu repository (https://github.com/opentofu/opentofu/blob/main/CONTRIBUTING.md) in order to make clear to all contributors that copyright infringement is not acceptable when contributing to OpenTofu (see https://github.com/opentofu/opentofu/pull/1209). Specifically, the contribution guide now contains the following section:

> *A note on copyright*
>
> *We take copyright and intellectual property very seriously. A few quick rules should help you:*
>
> - *When you submit a PR, you are responsible for the code in that pull request. You signal your acceptance of the DCO with your sign-off.*
> - *If you include code in your PR that you didn't write yourself, make sure you have permission from the author. If you have permission, always add the Co-authored-by sign-off to your commits to indicate the author of the code you are adding.*
> - *Be careful about AI coding assistants! Coding assistants based on large language models (LLMs), such as ChatGPT or GitHub Copilot, are awesome tools to help. However, in the specific case of OpenTofu the training data may include the BSL-licensed Terraform. Since the OpenTofu/Terraform codebase is very specific and LLMs don't have any other training sources, they may emit copyrighted code. Please avoid using LLM-based coding assistants as much as possible.*
> - *When you copy/paste code from within the OpenTofu code, always make it explicit where you copied from. This helps us resolve issues later on.*
> - *Before you copy code from external sources, make sure that the license allows this. Also make sure that any licensing requirements, such as attribution, are met. When in doubt, ask first!*
> - *Specifically, do not copy from the Terraform repository, or any PRs others have filed against that repository. This code is licensed under the BSL, a license which is not compatible with OpenTofu. (You may submit the same PR to both Terraform and OpenTofu as long as you are the author of both.)*
>
> *Warning*
>
> *To protect the OpenTofu project from legal issues violating these rules will immediately disqualify your PR from being merged and you from working on that area of the OpenTofu code base in the future. Repeat violations may get you barred from contributing to OpenTofu.*

This section is referenced in multiple places, such as in the "I've been assigned an issue, now what?" section often linked to new contributors:

> *Have you read this document? If not, please give it a quick skim, especially the sections about copyright and signoffs.*

Additionally, ever since OpenTofu's inception, contributors are required to sign-off their commits on the repository attesting to their adherence to the DCO (https://developercertificate.org/).

Finally, since March 22nd, 2024 the core team is also in the process of refining a PR (https://github.com/opentofu/opentofu/pull/1423) requiring contributors to not only sign off their commits, but also attest to the fact that any code they have not explicitly written themselves is marked with comments explaining their source.

On the matter of copyright headers

The OpenTofu team has never and will not knowingly mislabel BUSL-1.1-licensed code as MPL-2.0. It has adequate processes in place to ensure that third party submissions also do not mislabel code with the incorrect license.

OpenTofu and its predecessor, a version of Terraform shortly after 1.5.5, are licensed under the MPL-2.0 license. Much of the original code base contained copyright headers with an SPDX license identifier as follows:

```
// Copyright (c) HashiCorp, Inc.
// SPDX-License-Identifier: MPL-2.0
```

It is customary to move code into new files or copy code within the existing project during development and in doing so, creating a derivative work of the original, HashiCorp-owned code under the MPL-2.0 license. If the license header was not added to newly created files, it would likely constitute removal of copyright management information (CMI). Unfortunately, it would not be practical to accurately keep track of which code pieces were originally written or licensed to HashiCorp under their CLA and which ones were purely new creations in OpenTofu. Arguably, OpenTofu in its entirety is a derivative work of the MPL-2.0-licensed version of Terraform. For both potential legal complications and wishing to honor HashiCorps original copyright, the core developer team decided to do so by adding the following headers to all files containing code:

```
// Copyright (c) The OpenTofu Authors
// SPDX-License-Identifier: MPL-2.0
// Copyright (c) 2023 HashiCorp, Inc.
// SPDX-License-Identifier: MPL-2.0
```

The only files that do not fall under this rule are supplemental files that did not originally have copyright headers, such as example code snippets in the documentation or supplemental files that were newly added to OpenTofu only (such as the installation instructions on the website).

On the matter of file names

In both projects the file names presented in this document are identical or similar. However, these file names originate with the naming convention of the "moved" block, which has the following file names:

- move_statement.go
- move_endpoint.go
- move_endpoint_test.go
- moved.go
- moved_test.go

On the matter of remove_statement.go

We compare the move_statement.go (OpenTofu/Terraform, MPL-2.0), remove_statement.go (OpenTofu, MPL-2.0) and remove_statement.go (Terraform, BUSL-1.1) files code block by code block and remove the comments for readability. Where we have removed code comments in the code analysis below, all code comments differed between HashiCorp's BUSL-1.1-licensed code and OpenTofu's MPL-2.0-licensed code.

The MoveStatement struct

We start with the move_statement.go (MPL-2.0 license):

```
type MoveStatement struct {
    From, To  *addrs.MoveEndpointInModule
    DeclRange tfdiags.SourceRange

    Implied bool
}
```

Comparing to the remove_statement.go in OpenTofu (MPL-2.0 license):

```
type RemoveStatement struct {
    From      addrs.ConfigRemovable
    DeclRange tfdiags.SourceRange
}
```

And to remove_statement.go in Terraform (BUSL-1.1 license):

```
type RemoveStatement struct {
    From addrs.ConfigMoveable

    Destroy   bool
    DeclRange tfdiags.SourceRange
}
```

The GetEndpointsToRemove (OpenTofu, MPL-2.0) / FindRemoveStatements (Terraform, BUSL-1.1) function

Again, we compare the functions to the original `FindMoveStatements` function (MPL-2.0):

```
// FindMoveStatements recurses through the modules of the given configuration
// and returns a flat set of all "moved" blocks defined within, in a
// deterministic but undefined order.
func FindMoveStatements(rootCfg *configs.Config) []MoveStatement {
    return findMoveStatements(rootCfg, nil)
}
```

Comparing to the OpenTofu implementation (MPL-2.0):

```
// GetEndpointsToRemove recurses through the modules of the given configuration
// and returns an array of all "removed" addresses within, in a
// deterministic but undefined order.
// We also validate that the removed modules/resources configuration blocks were removed.
func GetEndpointsToRemove(rootCfg *configs.Config) ([]addrs.ConfigRemovable,
tfdiags.Diagnostics) {
    rm := findRemoveStatements(rootCfg, nil)
    diags := validateRemoveStatements(rootCfg, rm)
```

```
    removedAddresses := make([]addrs.ConfigRemovable, len(rm))
    for i, rs := range rm {
        removedAddresses[i] = rs.From
    }
    return removedAddresses, diags
}
```

Finally, the Terraform version (BUSL-1.1):

```
// FindRemoveStatements recurses through the modules of the given configuration
// and returns a set of all "removed" blocks defined within after deduplication
// on the From address.
//
// Error diagnostics are returned if any resource or module targeted by a remove
// block is still defined in configuration.
//
// A "removed" block in a parent module overrides a removed block in a child
// module when both target the same configuration object.
func FindRemoveStatements(rootCfg *configs.Config) (addrs.Map[addrs.ConfigMoveable,
RemoveStatement], tfdiags.Diagnostics) {
    stmts := findRemoveStatements(rootCfg, addrs.MakeMap[addrs.ConfigMoveable,
RemoveStatement]())
    diags := validateRemoveStatements(rootCfg, stmts)
    return stmts, diags
}
```

It is worth noting that function names starting with the word "validate" are customary in Go programming and there are at least 40 such occurrences in the entire codebase, such as ValidateMoves, ValidateMoveStatementGraph, validateProviderConfig, ValidateTargetFile, etc. In other respects, the OpenTofu implementation is functionally dissimilar to the Terraform implementation.

The findRemoveStatements function

Here we note that upper case and lower case functions are different in the Go programming language and should not be confused. We compare to the findMoveStatements function in the original MPL-2.0 licensed code:

```
func findMoveStatements(cfg *configs.Config, into []MoveStatement) []MoveStatement {
    modAddr := cfg.Path
    for _, mc := range cfg.Module.Moved {
        fromAddr, toAddr := addrs.UnifyMoveEndpoints(modAddr, mc.From, mc.To)
        if fromAddr == nil || toAddr == nil {
            // Invalid combination should've been caught during original
            // configuration decoding, in the configs package.
            panic(fmt.Sprintf("incompatible move endpoints in %s", mc.DeclRange))
        }

        into = append(into, MoveStatement{
            From:      fromAddr,
            To:        toAddr,
            DeclRange: tfdiags.SourceRangeFromHCL(mc.DeclRange),
            Implied:   false,
```

```
        })
    }

    for _, childCfg := range cfg.Children {
        into = findMoveStatements(childCfg, into)
    }

    return into
}
```

Comparing to the findRemoveStatements in OpenTofu (MPL-2.0):

```
func findRemoveStatements(cfg *configs.Config, into []*RemoveStatement) []*RemoveStatement {
    modAddr := cfg.Path

    for _, rc := range cfg.Module.Removed {
        var removedEndpoint *RemoveStatement
        switch FromAddress := rc.From.RelSubject.(type) {
        case addrs.ConfigResource:
            absModule := make(addrs.Module, 0, len(modAddr)+len(FromAddress.Module))
            absModule = append(absModule, modAddr...)
            absModule = append(absModule, FromAddress.Module...)

            var absConfigResource addrs.ConfigRemovable = addrs.ConfigResource{
                Resource: FromAddress.Resource,
                Module:   absModule,
            }

            removedEndpoint = &RemoveStatement{From: absConfigResource, DeclRange:
tfdiags.SourceRangeFromHCL(rc.DeclRange)}

        case addrs.Module:
            var absModule = make(addrs.Module, 0, len(modAddr)+len(FromAddress))
            absModule = append(absModule, modAddr...)
            absModule = append(absModule, FromAddress...)
            removedEndpoint = &RemoveStatement{From: absModule, DeclRange:
tfdiags.SourceRangeFromHCL(rc.DeclRange)}

        default:
            panic(fmt.Sprintf("unhandled address type %T", FromAddress))
        }

        into = append(into, removedEndpoint)

    }

    for _, childCfg := range cfg.Children {
        into = findRemoveStatements(childCfg, into)
    }
```

```
    return into
}
```

Comparing to the Terraform version (BUSL-1.1):

```
func findRemoveStatements(cfg *configs.Config, into addrs.Map[addrs.ConfigMoveable,
RemoveStatement]) addrs.Map[addrs.ConfigMoveable, RemoveStatement] {
    for _, mc := range cfg.Module.Removed {
        switch mc.From.ObjectKind() {
        case addrs.RemoveTargetResource:
            res := mc.From.RelSubject.(addrs.ConfigResource)
            fromAddr := addrs.ConfigResource{
                Module:   append(cfg.Path, res.Module...),
                Resource: res.Resource,
            }

            existingStatement, ok := into.GetOk(fromAddr)
            if ok {
                if existingResource, ok := existingStatement.From.(addrs.ConfigResource); ok
&&
                    existingResource.Equal(fromAddr) {
                    continue
                }
            }

            into.Put(fromAddr, RemoveStatement{
                From:      fromAddr,
                Destroy:   mc.Destroy,
                DeclRange: tfdiags.SourceRangeFromHCL(mc.DeclRange),
            })
        case addrs.RemoveTargetModule:
            mod := mc.From.RelSubject.(addrs.Module)
            absMod := append(cfg.Path, mod...)

            existingStatement, ok := into.GetOk(mc.From.RelSubject)
            if ok {
                if existingModule, ok := existingStatement.From.(addrs.Module); ok &&
                    existingModule.Equal(absMod) {
                    continue
                }
            }

            into.Put(absMod, RemoveStatement{
                From:      absMod,
                Destroy:   mc.Destroy,
                DeclRange: tfdiags.SourceRangeFromHCL(mc.DeclRange),
            })
        default:
            panic("Unsupported remove target kind")
        }
    }
```

```
    for _, childCfg := range cfg.Children {
        into = findRemoveStatements(childCfg, into)
    }

    return into
}
```

Since these blocks are lengthy to read, we will compare the functions line by line:

Moved (OpenTofu, MPL-2.0):

```
func findMoveStatements(cfg *configs.Config, into []MoveStatement) []MoveStatement {
```

Removed (OpenTofu, MPL-2.0):

```
func findRemoveStatements(cfg *configs.Config, into []*RemoveStatement) []*RemoveStatement {
```

Removed (Terraform, BUSL-1.1):

```
func findRemoveStatements(cfg *configs.Config, into addrs.Map[addrs.ConfigMoveable,
RemoveStatement]) addrs.Map[addrs.ConfigMoveable, RemoveStatement] {
```

Here we note that `configs.Config` is a configuration structure in the original MPL-2.0-licensed code. The `into` parameter has a completely different type in the Terraform version (a key-value mapping instead of a list), which makes the Terraform variant function different to the original "moved" block.

The next line is present in both the "moved" and "removed" implementation in OpenTofu (MPL-2.0), but missing from Terraform (BUSL-1.1):

```
modAddr := cfg.Path
```

The next line is an iteration, which has a fixed structure in the Go programming language.

Moved (OpenTofu, MPL-2.0):

```
for _, mc := range cfg.Module.Moved {
```

Removed (OpenTofu, MPL-2.0):

```
for _, rc := range cfg.Module.Removed {
```

Removed (Terraform, BUSL-1.1):

```
for _, mc := range cfg.Module.Removed {
```

The next switch block is not present in the "moved" block, and is substantially dissimilar between OpenTofu and Terraform. We start with the switch statement itself. It is worth noting that "switch" is a language element in Go, not a Terraform-specific name.

OpenTofu (MPL-2.0):

```
switch FromAddress := rc.From.RelSubject.(type) {
```

Terraform (BUSL-1.1):

```
switch mc.From.ObjectKind() {
```

We note that the type switch statement is customary in Go and appears in the codebase in over 100 places. Specifically, the move_endpoint.go (MPL-2.0) contains a similar switch-case:

```
switch relAddr := relAddr.(type) {
case ModuleInstance:
    // [...]
case ModuleAddrType:
    // [...]
default:
    // [...]
}
```

Now we evaluate the individual switch cases with comments omitted for readability as they are completely dissimilar.

OpenTofu (MPL-2.0):

```
case addrs.ConfigResource:
    absModule := make(addrs.Module, 0, len(modAddr)+len(FromAddress.Module))
    absModule = append(absModule, modAddr...)
    absModule = append(absModule, FromAddress.Module...)

    var absConfigResource addrs.ConfigRemovable = addrs.ConfigResource{
        Resource: FromAddress.Resource,
        Module:   absModule,
    }

    removedEndpoint = &RemoveStatement{From: absConfigResource, DeclRange:
tfdiags.SourceRangeFromHCL(rc.DeclRange)}
```

Terraform (BUSL-1.1):

```
case addrs.RemoveTargetResource:
    res := mc.From.RelSubject.(addrs.ConfigResource)
    fromAddr := addrs.ConfigResource{
        Module:   append(cfg.Path, res.Module...),
        Resource: res.Resource,
    }

    existingStatement, ok := into.GetOk(fromAddr)
    if ok {
        if existingResource, ok := existingStatement.From.(addrs.ConfigResource); ok &&
            existingResource.Equal(fromAddr) {
```

```
            continue
        }
    }

    into.Put(fromAddr, RemoveStatement{
        From:      fromAddr,
        Destroy:   mc.Destroy,
        DeclRange: tfdiags.SourceRangeFromHCL(mc.DeclRange),
    })
```

Here it is worth noting that the Terraform (BUSL-1.1) code duplicates the last section (`into.Put`) between the two cases, while the OpenTofu (MPL-2.0) code takes a different approach and only append to the "into" variable after the switch statement using a different method. The OpenTofu (MPL-2.0) method is similar to the one used in the "moved" implementation (MPL-2.0), but is dissimilar to the implementation in Terraform (BUSL-1.1), which uses "into.Put" instead of "append".

The contents of the item placed into the "into" variable in OpenTofu are, again, substantially similar to the original "moved" version (MPL-2.0) present in move_statement.go:

Moved (MPL-2.0):

```
into = append(into, MoveStatement{
    From:      fromAddr,
    To:        toAddr,
    DeclRange: tfdiags.SourceRangeFromHCL(mc.DeclRange),
    Implied:   false,
})
```

OpenTofu (MPL-2.0, formatted for readability):

```
removedEndpoint = &RemoveStatement{
    From: absConfigResource,
    DeclRange: tfdiags.SourceRangeFromHCL(rc.DeclRange)
}
```

And further down in the code:

```
into = append(into, removedEndpoint)
```

Terraform (BUSL-1.1):

```
into.Put(fromAddr, RemoveStatement{
    From:      fromAddr,
    Destroy:   mc.Destroy,
    DeclRange: tfdiags.SourceRangeFromHCL(mc.DeclRange),
})
```

Going back to the "switch" statement in question, we compare without the dissimilar comments.

OpenTofu (MPL-2.0):

```
case addrs.Module:
    var absModule = make(addrs.Module, 0, len(modAddr)+len(FromAddress))
    absModule = append(absModule, modAddr...)
    absModule = append(absModule, FromAddress...)
    removedEndpoint = &RemoveStatement{From: absModule, DeclRange:
tfdiags.SourceRangeFromHCL(rc.DeclRange)}
```

Terraform (BUSL-1.1):

```
case addrs.RemoveTargetModule:
    mod := mc.From.RelSubject.(addrs.Module)
    absMod := append(cfg.Path, mod...)

    existingStatement, ok := into.GetOk(mc.From.RelSubject)
    if ok {
        if existingModule, ok := existingStatement.From.(addrs.Module); ok &&
            existingModule.Equal(absMod) {
            continue
        }
    }

    into.Put(absMod, RemoveStatement{
        From:      absMod,
        Destroy:   mc.Destroy,
        DeclRange: tfdiags.SourceRangeFromHCL(mc.DeclRange),
    })
```

As before, the mechanism of appending to "into" is dissimilar.

The last case of the "switch" block is the default case, which is present in the moved block (MPL-2.0) and similar code is present in over 100 places in the codebase.

Moved (MPL-2.0):

```
default:
    panic(fmt.Sprintf("unsupported address type %T", addr))
```

OpenTofu (MPL-2.0):

```
default:
    panic(fmt.Sprintf("unhandled address type %T", FromAddress))
```

Terraform (BUSL-1.1):

```
default:
    panic("Unsupported remove target kind")
```

Finally, the last block is again almost identical to the original (MPL-2.0) "moved" block.

Moved (MPL-2.0):

```
for _, childCfg := range cfg.Children {
    into = findMoveStatements(childCfg, into)
}
```

Removed (OpenTofu, MPL-2.0):

```
for _, childCfg := range cfg.Children {
    into = findRemoveStatements(childCfg, into)
}
```

Removed (Terraform, BUSL-1.1):

```
for _, childCfg := range cfg.Children {
    into = findRemoveStatements(childCfg, into)
}
```

The validateRemoveStatements function

This function originates in the `ValidateMoves` function, an MPL-2.0-licensed function, but has been heavily refactored in OpenTofu to accommodate the functionality of the "removed" block. In OpenTofu, the internal logic is also based on the `findRemoveStatements` function (MPL-2.0) described above. The code in Terraform (BUSL-1.1) likely originates in the same place. As before, we present the entire functions and then compare the individual parts, including any code parts that are equivalent to the `ValidateMoves` function (MPL-2.0). We only include the relevant parts of the `ValidateMoves` function (MPL-2.0) for brevity.

ValidateMoves (OpenTofu, MPL-2.0):

```
func ValidateMoves(stmts []MoveStatement, rootCfg *configs.Config, declaredInsts
instances.Set) tfdiags.Diagnostics {

    // [...]

    for _, stmt := range stmts {
        // [...]
                diags = diags.Append(&hcl.Diagnostic{
                    Severity: hcl.DiagError,
                    Summary:  "Moved object still exists",
                    Detail: fmt.Sprintf(
                        "This statement declares a move from %s, but that %s is still
declared%s.\n\nChange your configuration so that this %s will be declared as %s instead.",
                        absFrom, noun, declaredAt, shortNoun, absTo,
                    ),
                    Subject: stmt.DeclRange.ToHCL().Ptr(),
                })
        // [...]

    return diags
}
```

OpenTofu (MPL-2.0):

```
func validateRemoveStatements(cfg *configs.Config, removeStatements []*RemoveStatement)
tfdiags.Diagnostics {
    var diags tfdiags.Diagnostics

    for _, rs := range removeStatements {
        fromAddr := rs.From
        if fromAddr == nil {
            panic(fmt.Sprintf("incompatible Remove endpoint in %s", rs.DeclRange.ToHCL()))
        }

        switch fromAddr := fromAddr.(type) {
        case addrs.ConfigResource:
            moduleConfig := cfg.Descendent(fromAddr.Module)
            if moduleConfig != nil && moduleConfig.Module.ResourceByAddr(fromAddr.Resource)
!= nil {
                diags = diags.Append(&hcl.Diagnostic{
                    Severity: hcl.DiagError,
                    Summary:  "Removed resource block still exists",
                    Detail: fmt.Sprintf(
                        "This statement declares a removal of the resource %s, but this
resource block still exists in the configuration. Please remove the resource block.",
                        fromAddr,
                    ),
                    Subject: rs.DeclRange.ToHCL().Ptr(),
                })
            }
        case addrs.Module:
            if cfg.Descendent(fromAddr) != nil {
                diags = diags.Append(&hcl.Diagnostic{
                    Severity: hcl.DiagError,
                    Summary:  "Removed module block still exists",
                    Detail: fmt.Sprintf(
                        "This statement declares a removal of the module %s, but this module
block still exists in the configuration. Please remove the module block.",
                        fromAddr,
                    ),
                    Subject: rs.DeclRange.ToHCL().Ptr(),
                })
            }
        default:
            panic(fmt.Sprintf("incompatible Remove endpoint address type in %s",
rs.DeclRange.ToHCL()))
        }
    }

    return diags
}
```

Terraform (BUSL-1.1):

```
func validateRemoveStatements(cfg *configs.Config, stmts addrs.Map[addrs.ConfigMoveable,
RemoveStatement]) (diags tfdiags.Diagnostics) {
    for _, rst := range stmts.Keys() {
        switch rst := rst.(type) {
        case addrs.ConfigResource:
            m := cfg.Descendent(rst.Module)
            if m == nil {
                break
            }

            if r := m.Module.ResourceByAddr(rst.Resource); r != nil {
                diags = diags.Append(&hcl.Diagnostic{
                    Severity: hcl.DiagError,
                    Summary:  "Removed resource still exists",
                    Detail:   fmt.Sprintf("This statement declares that %s was removed, but
it is still declared in configuration.", rst),
                    Subject:  r.DeclRange.Ptr(),
                })
            }
        case addrs.Module:
            if m := cfg.Descendent(rst); m != nil {
                diags = diags.Append(&hcl.Diagnostic{
                    Severity: hcl.DiagError,
                    Summary:  "Removed module still exists",
                    Detail:   fmt.Sprintf("This statement declares that %s was removed, but
it is still declared in configuration.", rst),
                    Subject:  m.CallRange.Ptr(),
                })
            }
        }
    }
    return diags
}
```

Since the function is long, again, we will go line by line.

ValidateMoves (MPL-2.0):

```
func ValidateMoves(stmts []MoveStatement, rootCfg *configs.Config, declaredInsts
instances.Set) tfdiags.Diagnostics {
```

Removed (OpenTofu, MPL-2.0):

```
func validateRemoveStatements(cfg *configs.Config, removeStatements []*RemoveStatement)
tfdiags.Diagnostics {
```

Removed (Terraform, BUSL-1.1):

```
func validateRemoveStatements(cfg *configs.Config, stmts addrs.Map[addrs.ConfigMoveable,
```

```
RemoveStatement]) (diags tfdiags.Diagnostics) {
```

In both the OpenTofu (MPL-2.0) and Terraform (BUSL-1.1) version, the arguments and return values are similar to `ValidateMoves` (MPL-2.0).

The next line is present only in OpenTofu (MPL-2.0), as the Terraform (BUSL-1.1) variant includes this in the function declaration (an uncommon Go programming technique not used or known by many).

OpenTofu (MPL-2.0):

```
var diags tfdiags.Diagnostics
```

The next block, again, is an iteration over all remove statements and the corresponding checks.

OpenTofu (MPL-2.0):

```
for _, rs := range removeStatements {
```

Terraform (BUSL-1.1):

```
for _, rst := range stmts.Keys() {
```

As noted before, the "for" and "range" keywords are inherent to the Go language and are not specific to Terraform (BUSL-1.1). In fact, this is the most obvious way to write an iteration and writing it differently would make the code unnecessarily complicated and possibly have a performance impact.

The next section is present in OpenTofu (MPL-2.0) and missing in Terraform (BUSL-1.1). It hardens the code against bugs as indicated by the comment:

OpenTofu (MPL-2.0):

```
fromAddr := rs.From
if fromAddr == nil {
    // Invalid value should've been caught during original
    // configuration decoding, in the configs package.
    panic(fmt.Sprintf("incompatible Remove endpoint in %s", rs.DeclRange.ToHCL()))
}
```

The next section performs a switch-case similar to `findRemoveStatements` (MPL-2.0).

OpenTofu (MPL-2.0):

```
switch fromAddr := fromAddr.(type) {
```

Terraform (BUSL-1.1):

```
switch rst := rst.(type) {
```

Again, we note that "switch" and "(type)" are specific to the Go language, not Terraform (BUSL-1.1). Next, we compare the cases in the switch. Here we also include the relevant part from the original `ValidateMoves` function (MPL-2.0):

ValidateMoves (MPL-2.0):

```
        diags = diags.Append(&hcl.Diagnostic{
            Severity: hcl.DiagError,
            Summary:  "Moved object still exists",
            Detail: fmt.Sprintf(
                "This statement declares a move from %s, but that %s is still
declared%s.\n\nChange your configuration so that this %s will be declared as %s instead.",
                absFrom, noun, declaredAt, shortNoun, absTo,
            ),
            Subject: stmt.DeclRange.ToHCL().Ptr(),
        })
```

OpenTofu (MPL-2.0):

```
case addrs.ConfigResource:
    moduleConfig := cfg.Descendent(fromAddr.Module)
    if moduleConfig != nil && moduleConfig.Module.ResourceByAddr(fromAddr.Resource) != nil {
        diags = diags.Append(&hcl.Diagnostic{
            Severity: hcl.DiagError,
            Summary:  "Removed resource block still exists",
            Detail: fmt.Sprintf(
                "This statement declares a removal of the resource %s, but this resource
block still exists in the configuration. Please remove the resource block.",
                fromAddr,
            ),
            Subject: rs.DeclRange.ToHCL().Ptr(),
        })
    }
```

Terraform (BUSL-1.1):

```
case addrs.ConfigResource:
    m := cfg.Descendent(rst.Module)
    if m == nil {
        break
    }

    if r := m.Module.ResourceByAddr(rst.Resource); r != nil {
        diags = diags.Append(&hcl.Diagnostic{
            Severity: hcl.DiagError,
            Summary:  "Removed resource still exists",
            Detail:   fmt.Sprintf("This statement declares that %s was removed, but it is
still declared in configuration.", rst),
            Subject:  r.DeclRange.Ptr(),
        })
    }
```

Here there are only two similarities. The `cfg.Descendent` call is a call to the MPL-2.0-licensed `Config` structure discussed before and is the only practical way to retrieve a descendant config. The same is true for the call to

`ResourceByAddr`, which is the only practical way to retrieve the resource associated with an address entered by the user. Both implementations check if the module configuration is `nil` (empty value), which is again necessary and customary to do in the Go programming language. However, the OpenTofu (MPL-2.0) implementation structures this differently and is more compact than the Terraform (BUSL-1.1) implementation. A similar calling structure is also present in `transform_diff.go` (MPL-2.0 license):

```
if t.Config == nil {
    return false
}

cfg := t.Config.DescendentForInstance(addr.Module)
if cfg == nil {
    return false
}

res := cfg.Module.ResourceByAddr(addr.ConfigResource().Resource)
if res == nil {
    return false
}
```

Similar error handling is also present in `import.go` (MPL-2.0):

```
targetConfig := config.DescendentForInstance(addr.Module)
if targetConfig == nil {
    modulePath := addr.Module.String()
    diags = diags.Append(&hcl.Diagnostic{
        Severity: hcl.DiagError,
        Summary:  "Import to non-existent module",
        Detail: fmt.Sprintf(
            "%s is not defined in the configuration. Please add configuration for this
module before importing into it.",
            modulePath,
        ),
    })
    c.showDiagnostics(diags)
    return 1
}
```

As discussed before, the diagnostics return block is wide spread in the code base and is not specific to the BUSL-1.1-licensed Terraform code.

The same analysis applies to the second case, which deals with modules.

OpenTofu (MPL-2.0):

```
if cfg.Descendent(fromAddr) != nil {
    diags = diags.Append(&hcl.Diagnostic{
        Severity: hcl.DiagError,
        Summary:  "Removed module block still exists",
        Detail: fmt.Sprintf(
            "This statement declares a removal of the module %s, but this module block still
```

```
exists in the configuration. Please remove the module block.",
            fromAddr,
        ),
        Subject: rs.DeclRange.ToHCL().Ptr(),
    })
}
```

Terraform (BUSL-1.1):

```
if m := cfg.Descendent(rst); m != nil {
    diags = diags.Append(&hcl.Diagnostic{
        Severity: hcl.DiagError,
        Summary:  "Removed module still exists",
        Detail:   fmt.Sprintf("This statement declares that %s was removed, but it is still
declared in configuration.", rst),
        Subject:  m.CallRange.Ptr(),
    })
}
```

On the matter of move_endpoint.go

This section will proceed as before. However, it is worth noting that the BUSL-1.1-licensed Terraform code calls out to the prior art that served as inspiration for this code in the first comment of this file:

```
// Like MoveEndpoint, RemoveTarget is a wrapping struct that captures the result
// of decoding an HCL traversal representing a relative path from the current
// module to a removeable object.
```

The RemoveEndpoint (OpenTofu, MPL-2.0) and RemoveTarget (Terraform, BUSL-1.1) structs

First, we will inspect the struct itself and then the receiver functions defined on this struct.

Moved (MPL-2.0):

```
// MoveEndpoint is to AbsMoveable and ConfigMoveable what Target is to
// Targetable: a wrapping struct that captures the result of decoding an HCL
// traversal representing a relative path from the current module to
// a moveable object.
//
// Its name reflects that its primary purpose is for the "from" and "to"
// addresses in a "moved" statement in the configuration, but it's also
// valid to use MoveEndpoint for other similar mechanisms that give
// OpenTofu hints about historical configuration changes that might
// prompt creating a different plan than OpenTofu would by default.
//
// To obtain a full address from a MoveEndpoint you must use
// either the package function UnifyMoveEndpoints (to get an AbsMoveable) or
// the method ConfigMoveable (to get a ConfigMoveable).
type MoveEndpoint struct {
    // SourceRange is the location of the physical endpoint address
```

```
    // in configuration, if this MoveEndpoint was decoded from a
    // configuration expresson.
    SourceRange tfdiags.SourceRange

    // Internally we (ab)use AbsMoveable as the representation of our
    // relative address, even though everywhere else in OpenTofu
    // AbsMoveable always represents a fully-absolute address.
    // In practice, due to the implementation of ParseMoveEndpoint,
    // this is always either a ModuleInstance or an AbsResourceInstance,
    // and we only consider the possibility of interpreting it as
    // a AbsModuleCall or an AbsResource in UnifyMoveEndpoints.
    // This is intentionally unexported to encapsulate this unusual
    // meaning of AbsMoveable.
    relSubject AbsMoveable
}
```

Removed (OpenTofu, MPL-2.0):

```
// RemoveEndpoint is to ConfigRemovable what Target is to Targetable:
// a wrapping struct that captures the result of decoding an HCL
// traversal representing a relative path from the current module to
// a removable object. It is very similar to MoveEndpoint.
//
// Its purpose is to represent the "from" address in a "removed" block
// in the configuration.
//
// To obtain a full address from a RemoveEndpoint we need to combine it
// with any ancestor modules in the configuration
type RemoveEndpoint struct {
    // SourceRange is the location of the physical endpoint address
    // in configuration, if this RemoveEndpoint was decoded from a
    // configuration expression.
    SourceRange tfdiags.SourceRange

    // the representation of our relative address as a ConfigRemovable
    RelSubject ConfigRemovable
}
```

Removed (Terraform, BUSL-1.1):

```
// Like MoveEndpoint, RemoveTarget is a wrapping struct that captures the result
// of decoding an HCL traversal representing a relative path from the current
// module to a removeable object.
//
// Remove targets are somewhat simpler than move endpoints, in that they deal
// only with resources and modules defined in configuration, not instances of
// those objects as recorded in state. We are therefore able to determine the
// ConfigMoveable up front, since specifying any resource or module instance key
// in a removed block is invalid.
//
// An interesting quirk of RemoveTarget is that RelSubject denotes a
```

```
// configuration object that, if the removed block is valid, should no longer
// exist in configuration. This "last known address" is used to locate and delete
// the appropriate state objects, or, in the case in which the user has forgotten
// to remove the object from configuration, to report the address of that block
// in an error diagnostic.
type RemoveTarget struct {
    // SourceRange is the location of the target address in configuration.
    SourceRange tfdiags.SourceRange

    // RelSubject, like MoveEndpoint's relSubject, abuses an absolute address
    // type to represent a relative address.
    RelSubject ConfigMoveable
}
```

Note that the `ObjectKind`, `String` and `Equal` receiver functions present in Terraform (BUSL-1.1) are not present in OpenTofu (MPL-2.0).

The ParseRemoveEndpoint (OpenTofu, MPL-2.0) and ParseRemoveTarget (Terraform, BUSL-1.1) functions

These functions both originate in the `ParseMoveEndpoint` (MPL-2.0) function.

ParseMoveEndpoint (MPL-2.0):

```
// ParseMoveEndpoint attempts to interpret the given traversal as a
// "move endpoint" address, which is a relative path from the module containing
// the traversal to a movable object in either the same module or in some
// child module.
//
// This deals only with the syntactic element of a move endpoint expression
// in configuration. Before the result will be useful you'll need to combine
// it with the address of the module where it was declared in order to get
// an absolute address relative to the root module.
func ParseMoveEndpoint(traversal hcl.Traversal) (*MoveEndpoint, tfdiags.Diagnostics) {
    path, remain, diags := parseModuleInstancePrefix(traversal)
    if diags.HasErrors() {
        return nil, diags
    }

    rng := tfdiags.SourceRangeFromHCL(traversal.SourceRange())

    if len(remain) == 0 {
        return &MoveEndpoint{
            relSubject:  path,
            SourceRange: rng,
        }, diags
    }

    riAddr, moreDiags := parseResourceInstanceUnderModule(path, remain)
    diags = diags.Append(moreDiags)
    if diags.HasErrors() {
        return nil, diags
```

```
    }

    return &MoveEndpoint{
        relSubject:  riAddr,
        SourceRange: rng,
    }, diags
}
```

ParseRemoveEndpoint (OpenTofu, MPL-2.0):

```
// ParseRemoveEndpoint attempts to interpret the given traversal as a
// "remove endpoint" address, which is a relative path from the module containing
// the traversal to a removable object in either the same module or in some
// child module.
//
// This deals only with the syntactic element of a remove endpoint expression
// in configuration. Before the result will be useful you'll need to combine
// it with the address of the module where it was declared in order to get
// an absolute address relative to the root module.
func ParseRemoveEndpoint(traversal hcl.Traversal) (*RemoveEndpoint, tfdiags.Diagnostics) {
    path, remain, diags := parseModulePrefix(traversal)
    if diags.HasErrors() {
        return nil, diags
    }

    rng := tfdiags.SourceRangeFromHCL(traversal.SourceRange())

    if len(remain) == 0 {
        return &RemoveEndpoint{
            RelSubject:  path,
            SourceRange: rng,
        }, diags
    }

    riAddr, moreDiags := parseResourceUnderModule(path, remain)
    diags = diags.Append(moreDiags)
    if diags.HasErrors() {
        return nil, diags
    }

    if riAddr.Resource.Mode == DataResourceMode {
        diags = diags.Append(&hcl.Diagnostic{
            Severity: hcl.DiagError,
            Summary:  "Data source address is not allowed",
            Detail:   "Data sources cannot be destroyed, and therefore, 'removed' blocks are
not allowed to target them. To remove data sources from the state, you should remove the
data source block from the configuration.",
            Subject:  traversal.SourceRange().Ptr(),
        })

        return nil, diags
```

```
    }

    return &RemoveEndpoint{
        RelSubject:  riAddr,
        SourceRange: rng,
    }, diags
}
```

ParseRemoveTarget (Terraform, BUSL-1.1):

```
func ParseRemoveTarget(traversal hcl.Traversal) (*RemoveTarget, tfdiags.Diagnostics) {
    path, remain, diags := parseModulePrefix(traversal)
    if diags.HasErrors() {
        return nil, diags
    }

    rng := tfdiags.SourceRangeFromHCL(traversal.SourceRange())

    if len(remain) == 0 {
        return &RemoveTarget{
            RelSubject:  path,
            SourceRange: rng,
        }, diags
    }

    rAddr, moreDiags := parseConfigResourceUnderModule(path, remain)
    diags = diags.Append(moreDiags)
    if diags.HasErrors() {
        return nil, diags
    }

    if rAddr.Resource.Mode == DataResourceMode {
        diags = diags.Append(&hcl.Diagnostic{
            Severity: hcl.DiagError,
            Summary:  "Data source address not allowed",
            Detail:   "Data sources are never destroyed, so they are not valid targets of
removed blocks. To remove the data source from state, remove the data source block from
configuration.",
            Subject:  rng.ToHCL().Ptr(),
        })
    }

    return &RemoveTarget{
        RelSubject:  rAddr,
        SourceRange: rng,
    }, diags
}
```

The only difference to the `MoveEndpoint` in both implementation is the testing for the presence of data sources. The lack of this test in OpenTofu (MPL-2.0) was originally raised as part of the code review process here (https://github.com/opentofu/opentofu/pull/1158#discussion_r1463578492) and subsequently fixed by the developer

writing this code in commit 686a809575a3afe3ed6ca6df4d7de825110019b6. As the developer was using GitHub Copilot at the time, the error message was likely created by the AI assistant and when prompted, the same AI assistant was producing a similar error message when testing for this report. However, we do not have accurate records of this event as we do not require contributors to screen-record their coding sessions.

On the matter of the remove_endpoint_test.go (OpenTofu, MPL-2.0) and remove_target_test.go (Terraform, BUSL-1.1) files

The OpenTofu (MPL-2.0) implementation originates in the move_endpoint_test.go (MPL-2.0) file, which contains very similar test cases. It is worth noting that the OpenTofu (MPL-2.0) implementation has 27 test cases, whereas the Terraform (BUSL-1.1) implementation contains only 8.

Since the test definition is very long, we'll first highlight the general structure of the `TestParseMoveEndpoint` (MPL-2.0) function, then the `TestParseRemoveEndpoint` (OpenTofu, MPL-2.0) function, and finally the `TestParseRemoveTarget` (Terraform BUSL-1.1) function.

`TestParseMoveEndpoint` (MPL-2.0):

```
func TestParseMoveEndpoint(t *testing.T) {
    tests := []struct {
        Input   string
        WantRel AbsMoveable // funny intermediate subset of AbsMoveable
        WantErr string
    }{
        // [...]
    }

    for _, test := range tests {
        t.Run(test.Input, func(t *testing.T) {
            traversal, hclDiags := hclsyntax.ParseTraversalAbs([]byte(test.Input), "",
hcl.InitialPos)
            if hclDiags.HasErrors() {
                // We're not trying to test the HCL parser here, so any
                // failures at this point are likely to be bugs in the
                // test case itself.
                t.Fatalf("syntax error: %s", hclDiags.Error())
            }

            moveEp, diags := ParseMoveEndpoint(traversal)

            switch {
            case test.WantErr != "":
                if !diags.HasErrors() {
                    t.Fatalf("unexpected success\nwant error: %s", test.WantErr)
                }
                gotErr := diags.Err().Error()
                if gotErr != test.WantErr {
                    t.Fatalf("wrong error\ngot:  %s\nwant: %s", gotErr, test.WantErr)
                }
            default:
                if diags.HasErrors() {
```

```
                t.Fatalf("unexpected error: %s", diags.Err().Error())
            }
            if diff := cmp.Diff(test.WantRel, moveEp.relSubject); diff != "" {
                t.Errorf("wrong result\n%s", diff)
            }
        }
    })
  }
}
```

`TestParseRemoveEndpoint` **(OpenTofu, MPL-2.0):**

```
func TestParseRemoveEndpoint(t *testing.T) {
    tests := []struct {
        Input   string
        WantRel ConfigRemovable
        WantErr string
    }{
        // [...]
    }

    for _, test := range tests {
        t.Run(test.Input, func(t *testing.T) {
            traversal, hclDiags := hclsyntax.ParseTraversalAbs([]byte(test.Input), "",
hcl.InitialPos)
            if hclDiags.HasErrors() {
                // We're not trying to test the HCL parser here, so any
                // failures at this point are likely to be bugs in the
                // test case itself.
                t.Fatalf("syntax error: %s", hclDiags.Error())
            }

            moveEp, diags := ParseRemoveEndpoint(traversal)

            switch {
            case test.WantErr != "":
                if !diags.HasErrors() {
                    t.Fatalf("unexpected success\nwant error: %s", test.WantErr)
                }
                gotErr := diags.Err().Error()
                if gotErr != test.WantErr {
                    t.Fatalf("wrong error\ngot:  %s\nwant: %s", gotErr, test.WantErr)
                }
            default:
                if diags.HasErrors() {
                    t.Fatalf("unexpected error: %s", diags.Err().Error())
                }
                if diff := cmp.Diff(test.WantRel, moveEp.RelSubject); diff != "" {
                    t.Errorf("wrong result\n%s", diff)
                }
            }
```

```
        })
    }
}
```

`TestParseRemoveTarget` (BUSL-1.1):

```
func TestParseRemoveTarget(t *testing.T) {
    tests := []struct {
        Input   string
        Want    ConfigMoveable
        WantErr string
    }{
        // [...]
    }

    for _, test := range tests {
        t.Run(test.Input, func(t *testing.T) {
            traversal, hclDiags := hclsyntax.ParseTraversalAbs([]byte(test.Input), "",
hcl.InitialPos)
            if hclDiags.HasErrors() {
                // We're not trying to test the HCL parser here, so any
                // failures at this point are likely to be bugs in the
                // test case itself.
                t.Fatalf("syntax error: %s", hclDiags.Error())
            }

            remT, diags := ParseRemoveTarget(traversal)

            switch {
            case test.WantErr != "":
                if !diags.HasErrors() {
                    t.Fatalf("unexpected success\nwant error: %s", test.WantErr)
                }
                gotErr := diags.Err().Error()
                if gotErr != test.WantErr {
                    t.Fatalf("wrong error\ngot:  %s\nwant: %s", gotErr, test.WantErr)
                }
            default:
                if diags.HasErrors() {
                    t.Fatalf("unexpected error: %s", diags.Err().Error())
                }
                if diff := cmp.Diff(test.Want, remT.RelSubject); diff != "" {
                    t.Errorf("wrong result\n%s", diff)
                }
            }
        })
    }
}
```

Since the test framework is virtually identical to the original, MPL-2.0-licensed code, we will now compare the test cases. We would like to note that the test structure is mandated largely by technical necessity.

Now we will go over the individual test cases, listing every Terraform (BUSL-1.1) test case and attempting to find an OpenTofu (MPL-2.0) equivalent test case, which is not always possible. It is worth noting that these test cases are required by logic, otherwise the code could not be covered adequately.

Case 1:

This case tests a resource without a module, which is the most basic test case.

Moved (MPL-2.0):

```
{
    `foo.bar`,
    AbsResourceInstance{
        Module: RootModuleInstance,
        Resource: ResourceInstance{
            Resource: Resource{
                Mode: ManagedResourceMode,
                Type: "foo",
                Name: "bar",
            },
            Key: NoKey,
        },
    },
    ``,
},
```

Removed (OpenTofu, MPL-2.0):

```
{
    `foo.bar`,
    ConfigResource{
        Module: RootModule,
        Resource: Resource{

            Mode: ManagedResourceMode,
            Type: "foo",
            Name: "bar",
        },
    },
    ``,
},
```

Removed (Terraform, BUSL-1.1):

```
{
    `test_instance.bar`,
    ConfigResource{
        Module: RootModule,
        Resource: Resource{
            Mode: ManagedResourceMode,
            Type: "test_instance",
```

```
            Name: "bar",
        },
    },
    ``,
}
```

Case 2:

This test case tests a resource inside a module. OpenTofu (MPL-2.0) has two test cases to cover this case, whereas Terraform (BUSL-1.1) and the Moved (MPL-2.0) implementation only have one.

Moved (MPL-2.0):

```
{
    `module.foo.bar.baz`,
    RootModule,
    ConfigResource{
        Module: Module{"foo"},
        Resource: Resource{
            Mode: ManagedResourceMode,
            Type: "bar",
            Name: "baz",
        },
    },
}
```

Removed (OpenTofu, MPL-2.0):

```
{
    `module.boop`,
    Module{"boop"},
    ``,
},
{
    `module.boop.foo.bar`,
    ConfigResource{
        Module: Module{"boop"},
        Resource: Resource{
            Mode: ManagedResourceMode,
            Type: "foo",
            Name: "bar",
        },
    },
    ``,
}
```

Removed (Terraform, BUSL-1.1):

```
{
    `module.foo.test_instance.bar`,
    ConfigResource{
```

```
        Module: []string{"foo"},
        Resource: Resource{
            Mode: ManagedResourceMode,
            Type: "test_instance",
            Name: "bar",
        },
    },
    ``,
}
```

Case 3:

This test case shows a resource in a module in a module, which is an extension of test case 2.

Removed (OpenTofu, MPL-2.0):

```
{
    `module.boop.module.bip.foo.bar`,
    ConfigResource{
        Module: Module{"boop", "bip"},
        Resource: Resource{
            Mode: ManagedResourceMode,
            Type: "foo",
            Name: "bar",
        },
    },
    ``,
}
```

Removed (Terraform, BUSL-1.1):

```
{
    `module.foo.module.baz.test_instance.bar`,
    ConfigResource{
        Module: []string{"foo", "baz"},
        Resource: Resource{
            Mode: ManagedResourceMode,
            Type: "test_instance",
            Name: "bar",
        },
    },
    ``,
}
```

Case 4 and 5:

These test cases test that data sources are not supported. This is the data source equivalent of case 1 and case 2. There is no equivalent code in move_endpoint_test.go (MPL-2.0) because the "moved" endpoint supports data sources. It is worth noting that the error messages are the exact error messages emitted by the code by `ParseRemoveEndpoint` (OpenTofu, MPL-2.0) / `ParseRemoveEndpoint` (Terraform, BUSL-1.1). The entire test case

is strictly required on a technical level to achieve testing the code. It is also worth noting that there is no logical equivalent of the Terraform (BUSL-1.1) case 5.

Removed (OpenTofu, MPL-2.0):

```
{
    `data.foo.bar`,
    nil,
    `Data source address is not allowed: Data sources cannot be destroyed, and therefore, 'removed' blocks are not allowed to target them. To remove data sources from the state, you should remove the data source block from the configuration.`,
}
```

Removed (Terraform, BUSL-1.1):

```
{
    `data.test_ds.moo`,
    nil,
    `Data source address not allowed: Data sources are never destroyed, so they are not valid targets of removed blocks. To remove the data source from state, remove the data source block from configuration.`,
},
{
    `module.foo.data.test_ds.noo`,
    nil,
    `Data source address not allowed: Data sources are never destroyed, so they are not valid targets of removed blocks. To remove the data source from state, remove the data source block from configuration.`,
}
```

Case 6, 7, 8:

These test cases test numbered instances or modules. Case 7 is not present in OpenTofu (MPL-2.0).

Removed (OpenTofu, MPL-2.0):

```
{
    `foo.bar[0]`,
    nil,
    `Resource instance address with keys is not allowed: Resource address cannot be a resource instance (e.g. "null_resource.a[0]"), it must be a resource instead (e.g. "null_resource.a").`,
},
// [...]
{
    `module.boop.foo.bar[0]`,
    nil,
    `Resource instance address with keys is not allowed: Resource address cannot be a resource instance (e.g. "null_resource.a[0]"), it must be a resource instead (e.g. "null_resource.a").`,
},
```

Removed (Terraform, BUSL 1.1):

```
{
    `test_instance.foo[0]`,
    nil,
    `Resource instance keys not allowed: Resource address must be a resource (e.g.
"test_instance.foo"), not a resource instance (e.g. "test_instance.foo[1]").`,
},
{
    `module.foo[0].test_instance.bar`,
    nil,
    `Module instance keys not allowed: Module address must be a module (e.g. "module.foo"),
not a module instance (e.g. "module.foo[1]").`,
},
{
    `module.foo.test_instance.bar[0]`,
    nil,
    `Resource instance keys not allowed: Resource address must be a resource (e.g.
"test_instance.foo"), not a resource instance (e.g. "test_instance.foo[1]").`,
}
```

Cases not present in Terraform (BUSL-1.1)

As mentioned before, OpenTofu (MPL-2.0) has 27 test cases, whereas the Terraform (BUSL-1.1) implementation contains only 8. As shown above, some test cases originate with the "moved" block (MPL-2.0), while other test cases are diverging or logically necessary to write tests that cover all desired behavior.

On the matter of removed.go

As before, the removed.go file is based on the moved.go (MPL-2.0) file in OpenTofu (MPL-2.0), which appears to be the case in Terraform (BUSL-1.1) as well.

The Removed struct

Moved (MPL-2.0):

```
type Moved struct {
    From *addrs.MoveEndpoint
    To   *addrs.MoveEndpoint

    DeclRange hcl.Range
}
```

Removed (OpenTofu, MPL-2.0):

```
type Removed struct {
    From *addrs.RemoveEndpoint

    DeclRange hcl.Range
}
```

Removed (Terraform, BUSL-1.1):

```
type Removed struct {
    From *addrs.RemoveTarget

    Destroy bool

    DeclRange hcl.Range
}
```

The decodeRemovedBlock function

As before, we will first list the entire function and then go line by line.

Moved (MPL-2.0):

```
func decodeMovedBlock(block *hcl.Block) (*Moved, hcl.Diagnostics) {
    var diags hcl.Diagnostics
    moved := &Moved{
        DeclRange: block.DefRange,
    }

    content, moreDiags := block.Body.Content(movedBlockSchema)
    diags = append(diags, moreDiags...)

    if attr, exists := content.Attributes["from"]; exists {
        from, traversalDiags := hcl.AbsTraversalForExpr(attr.Expr)
        diags = append(diags, traversalDiags...)
        if !traversalDiags.HasErrors() {
            from, fromDiags := addrs.ParseMoveEndpoint(from)
            diags = append(diags, fromDiags.ToHCL()...)
            moved.From = from
        }
    }

    if attr, exists := content.Attributes["to"]; exists {
        to, traversalDiags := hcl.AbsTraversalForExpr(attr.Expr)
        diags = append(diags, traversalDiags...)
        if !traversalDiags.HasErrors() {
            to, toDiags := addrs.ParseMoveEndpoint(to)
            diags = append(diags, toDiags.ToHCL()...)
            moved.To = to
        }
    }

    // we can only move from a module to a module, resource to resource, etc.
    if !diags.HasErrors() {
        if !moved.From.MightUnifyWith(moved.To) {
            // We can catch some obviously-wrong combinations early here,
            // but we still have other dynamic validation to do at runtime.
            diags = diags.Append(&hcl.Diagnostic{
```

```
                Severity: hcl.DiagError,
                Summary:  "Invalid \"moved\" addresses",
                Detail:   "The \"from\" and \"to\" addresses must either both refer to
resources or both refer to modules.",
                Subject:  &moved.DeclRange,
            })
        }
    }

    return moved, diags
}
```

Removed (OpenTofu, MPL-2.0):

```
func decodeRemovedBlock(block *hcl.Block) (*Removed, hcl.Diagnostics) {
    var diags hcl.Diagnostics
    removed := &Removed{
        DeclRange: block.DefRange,
    }

    content, moreDiags := block.Body.Content(removedBlockSchema)
    diags = append(diags, moreDiags...)

    if attr, exists := content.Attributes["from"]; exists {
        from, traversalDiags := hcl.AbsTraversalForExpr(attr.Expr)
        diags = append(diags, traversalDiags...)
        if !traversalDiags.HasErrors() {
            from, fromDiags := addrs.ParseRemoveEndpoint(from)
            diags = append(diags, fromDiags.ToHCL()...)
            removed.From = from
        }
    }

    return removed, diags
}
```

Removed (Terraform, BUSL-1.1):

```
func decodeRemovedBlock(block *hcl.Block) (*Removed, hcl.Diagnostics) {
    var diags hcl.Diagnostics
    removed := &Removed{
        DeclRange: block.DefRange,
    }

    content, moreDiags := block.Body.Content(removedBlockSchema)
    diags = append(diags, moreDiags...)

    if attr, exists := content.Attributes["from"]; exists {
        from, traversalDiags := hcl.AbsTraversalForExpr(attr.Expr)
        diags = append(diags, traversalDiags...)
        if !traversalDiags.HasErrors() {
```

```
            from, fromDiags := addrs.ParseRemoveTarget(from)
            diags = append(diags, fromDiags.ToHCL()...)
            removed.From = from
        }
    }

    removed.Destroy = true

    for _, block := range content.Blocks {
        switch block.Type {
        case "lifecycle":
            lcContent, lcDiags := block.Body.Content(removedLifecycleBlockSchema)
            diags = append(diags, lcDiags...)

            if attr, exists := lcContent.Attributes["destroy"]; exists {
                valDiags := gohcl.DecodeExpression(attr.Expr, nil, &removed.Destroy)
                diags = append(diags, valDiags...)
            }
        }
    }

    return removed, diags
}
```

Going line by line, we will first examine the function:

Moved (MPL-2.0):

```
func decodeMovedBlock(block *hcl.Block) (*Moved, hcl.Diagnostics) {
```

Removed (OpenTofu, MPL-2.0):

```
func decodeRemovedBlock(block *hcl.Block) (*Removed, hcl.Diagnostics) {
```

Removed (Terraform, BUSL-1.1):

```
func decodeRemovedBlock(block *hcl.Block) (*Removed, hcl.Diagnostics) {
```

The setup code:

Moved (MPL-2.0):

```
var diags hcl.Diagnostics
moved := &Moved{
    DeclRange: block.DefRange,
}

content, moreDiags := block.Body.Content(movedBlockSchema)
diags = append(diags, moreDiags...)
```

Removed (OpenTofu, MPL-2.0):

```
var diags hcl.Diagnostics
removed := &Removed{
    DeclRange: block.DefRange,
}

content, moreDiags := block.Body.Content(removedBlockSchema)
diags = append(diags, moreDiags...)
```

Removed (Terraform, BUSL-1.1):

```
var diags hcl.Diagnostics
removed := &Removed{
    DeclRange: block.DefRange,
}

content, moreDiags := block.Body.Content(removedBlockSchema)
diags = append(diags, moreDiags...)
```

Processing the "from" parameter:

Moved (MPL-2.0):

```
if attr, exists := content.Attributes["from"]; exists {
    from, traversalDiags := hcl.AbsTraversalForExpr(attr.Expr)
    diags = append(diags, traversalDiags...)
    if !traversalDiags.HasErrors() {
        from, fromDiags := addrs.ParseMoveEndpoint(from)
        diags = append(diags, fromDiags.ToHCL()...)
        moved.From = from
    }
}
```

Removed (OpenTofu, MPL-2.0):

```
if attr, exists := content.Attributes["from"]; exists {
    from, traversalDiags := hcl.AbsTraversalForExpr(attr.Expr)
    diags = append(diags, traversalDiags...)
    if !traversalDiags.HasErrors() {
        from, fromDiags := addrs.ParseRemoveEndpoint(from)
        diags = append(diags, fromDiags.ToHCL()...)
        removed.From = from
    }
}
```

Removed (Terraform, BUSL-1.1):

```
if attr, exists := content.Attributes["from"]; exists {
    from, traversalDiags := hcl.AbsTraversalForExpr(attr.Expr)
```

```
    diags = append(diags, traversalDiags...)
    if !traversalDiags.HasErrors() {
        from, fromDiags := addrs.ParseRemoveTarget(from)
        diags = append(diags, fromDiags.ToHCL()...)
        removed.From = from
    }
}
```

The subsequent code in the "moved" (MPL-2.0) and the Terraform (BUSL-1.1) implementation is not present in OpenTofu (MPL-2.0) and the function ends with the same return line in all three cases.

The removedBlockSchema

As before, the `removedBlockSchema` follows the structure of the `movedBlockSchema`.

Moved (MPL-2.0):

```
var movedBlockSchema = &hcl.BodySchema{
    Attributes: []hcl.AttributeSchema{
        {
            Name:     "from",
            Required: true,
        },
        {
            Name:     "to",
            Required: true,
        },
    },
}
```

Removed (OpenTofu, MPL-2.0):

```
var removedBlockSchema = &hcl.BodySchema{
    Attributes: []hcl.AttributeSchema{
        {
            Name:     "from",
            Required: true,
        },
    },
}
```

Removed (Terraform, BUSL-1.1):

```
var removedBlockSchema = &hcl.BodySchema{
    Attributes: []hcl.AttributeSchema{
        {
            Name:     "from",
            Required: true,
        },
    },
    Blocks: []hcl.BlockHeaderSchema{
```

```
        {
            Type: "lifecycle",
        },
    },
}
```

On the matter of the removed_test.go

The TestRemovedBlock_decode function

As with the test cases before, we will first show the structure of the tests and then analyze the test cases individually.

Moved (MPL-2.0):

```
func TestMovedBlock_decode(t *testing.T) {
    blockRange := hcl.Range{
        Filename: "mock.tf",
        Start:    hcl.Pos{Line: 3, Column: 12, Byte: 27},
        End:      hcl.Pos{Line: 3, Column: 19, Byte: 34},
    }

    foo_expr := hcltest.MockExprTraversalSrc("test_instance.foo")
    bar_expr := hcltest.MockExprTraversalSrc("test_instance.bar")

    foo_index_expr := hcltest.MockExprTraversalSrc("test_instance.foo[1]")
    bar_index_expr := hcltest.MockExprTraversalSrc("test_instance.bar[\"one\"]")

    mod_foo_expr := hcltest.MockExprTraversalSrc("module.foo")
    mod_bar_expr := hcltest.MockExprTraversalSrc("module.bar")

    tests := map[string]struct {
        input *hcl.Block
        want  *Moved
        err   string
    }{
        // [...]
    }

    for name, test := range tests {
        t.Run(name, func(t *testing.T) {
            got, diags := decodeMovedBlock(test.input)

            if diags.HasErrors() {
                if test.err == "" {
                    t.Fatalf("unexpected error: %s", diags.Errs())
                }
                if gotErr := diags[0].Summary; gotErr != test.err {
                    t.Errorf("wrong error, got %q, want %q", gotErr, test.err)
                }
            } else if test.err != "" {
                t.Fatal("expected error")
```

```
            }

            if !cmp.Equal(got, test.want, cmp.AllowUnexported(addrs.MoveEndpoint{})) {
                t.Fatalf("wrong result: %s", cmp.Diff(got, test.want))
            }
        })
    }
}
```

Removed (OpenTofu, MPL-2.0):

```
func TestRemovedBlock_decode(t *testing.T) {
    blockRange := hcl.Range{
        Filename: "mock.tf",
        Start:    hcl.Pos{Line: 3, Column: 12, Byte: 27},
        End:      hcl.Pos{Line: 3, Column: 19, Byte: 34},
    }

    foo_expr := hcltest.MockExprTraversalSrc("test_instance.foo")
    mod_foo_expr := hcltest.MockExprTraversalSrc("module.foo")
    foo_index_expr := hcltest.MockExprTraversalSrc("test_instance.foo[1]")
    mod_boop_index_foo_expr :=
hcltest.MockExprTraversalSrc("module.boop[1].test_instance.foo")
    data_foo_expr := hcltest.MockExprTraversalSrc("data.test_instance.foo")

    tests := map[string]struct {
        input *hcl.Block
        want  *Removed
        err   string
    }{
        // [...]
    }

    for name, test := range tests {
        t.Run(name, func(t *testing.T) {
            got, diags := decodeRemovedBlock(test.input)

            if diags.HasErrors() {
                if test.err == "" {
                    t.Fatalf("unexpected error: %s", diags.Errs())
                }
                if gotErr := diags[0].Summary; gotErr != test.err {
                    t.Errorf("wrong error, got %q, want %q", gotErr, test.err)
                }
            } else if test.err != "" {
                t.Fatal("expected error")
            }

            if !cmp.Equal(got, test.want, cmp.AllowUnexported(addrs.MoveEndpoint{})) {
                t.Fatalf("wrong result: %s", cmp.Diff(got, test.want))
            }
```

```
        })
    }
}
```

Removed (Terraform, BUSL-1.1):

```
func TestRemovedBlock_decode(t *testing.T) {
    blockRange := hcl.Range{
        Filename: "mock.tf",
        Start:    hcl.Pos{Line: 3, Column: 12, Byte: 27},
        End:      hcl.Pos{Line: 3, Column: 19, Byte: 34},
    }

    foo_expr := hcltest.MockExprTraversalSrc("test_instance.foo")
    foo_index_expr := hcltest.MockExprTraversalSrc("test_instance.foo[1]")
    mod_foo_expr := hcltest.MockExprTraversalSrc("module.foo")
    mod_foo_index_expr := hcltest.MockExprTraversalSrc("module.foo[1]")

    tests := map[string]struct {
        input *hcl.Block
        want  *Removed
        err   string
    }{
        // [...]
    }

    for name, test := range tests {
        t.Run(name, func(t *testing.T) {
            got, diags := decodeRemovedBlock(test.input)

            if diags.HasErrors() {
                if test.err == "" {
                    t.Fatalf("unexpected error: %s", diags.Errs())
                }
                if gotErr := diags[0].Summary; gotErr != test.err {
                    t.Errorf("wrong error, got %q, want %q", gotErr, test.err)
                }
            } else if test.err != "" {
                t.Fatal("expected error")
            }

            if !cmp.Equal(got, test.want, cmp.AllowUnexported(addrs.MoveEndpoint{})) {
                t.Fatalf("wrong result: %s", cmp.Diff(got, test.want))
            }
        })
    }
}
```

The test structure is similar, but individually different from each other in the variable declaration block:

Moved (MPL-2.0):

```
foo_expr := hcltest.MockExprTraversalSrc("test_instance.foo")
bar_expr := hcltest.MockExprTraversalSrc("test_instance.bar")

foo_index_expr := hcltest.MockExprTraversalSrc("test_instance.foo[1]")
bar_index_expr := hcltest.MockExprTraversalSrc("test_instance.bar[\"one\"]")

mod_foo_expr := hcltest.MockExprTraversalSrc("module.foo")
mod_bar_expr := hcltest.MockExprTraversalSrc("module.bar")
```

Removed (OpenTofu, MPL-2.0):

```
foo_expr := hcltest.MockExprTraversalSrc("test_instance.foo")
mod_foo_expr := hcltest.MockExprTraversalSrc("module.foo")
foo_index_expr := hcltest.MockExprTraversalSrc("test_instance.foo[1]")
mod_boop_index_foo_expr := hcltest.MockExprTraversalSrc("module.boop[1].test_instance.foo")
data_foo_expr := hcltest.MockExprTraversalSrc("data.test_instance.foo")
```

Removed (Terraform, BUSL-1.1):

```
foo_expr := hcltest.MockExprTraversalSrc("test_instance.foo")
foo_index_expr := hcltest.MockExprTraversalSrc("test_instance.foo[1]")
mod_foo_expr := hcltest.MockExprTraversalSrc("module.foo")
mod_foo_index_expr := hcltest.MockExprTraversalSrc("module.foo[1]")
```

Now we will analyze each test case, numbering them by their location in the Terraform (BUSL-1.1) code and looking for equivalent test cases in the OpenTofu (MPL-2.0) implementation and the "moved" block (MPL-2.0).

Test case 1 and 2:

These test cases have no equivalent ones in OpenTofu (MPL-2.0) since the "destroy" option is not present in OpenTofu (MPL-2.0):

Removed (Terraform, BUSL-1.1):

```
"destroy true": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: foo_expr,
                },
            },
            Blocks: hcl.Blocks{
                &hcl.Block{
                    Type: "lifecycle",
                    Body: hcltest.MockBody(&hcl.BodyContent{
                        Attributes: hcl.Attributes{
                            "destroy": {
```

```
                                Name: "destroy",
                                Expr: hcltest.MockExprLiteral(cty.BoolVal(true)),
                            },
                        },
                    }),
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        From:      mustRemoveEndpointFromExpr(foo_expr),
        Destroy:   true,
        DeclRange: blockRange,
    },
    ``,
},
"destroy false": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: foo_expr,
                },
            },
            Blocks: hcl.Blocks{
                &hcl.Block{
                    Type: "lifecycle",
                    Body: hcltest.MockBody(&hcl.BodyContent{
                        Attributes: hcl.Attributes{
                            "destroy": {
                                Name: "destroy",
                                Expr: hcltest.MockExprLiteral(cty.BoolVal(false)),
                            },
                        },
                    }),
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        From:      mustRemoveEndpointFromExpr(foo_expr),
        Destroy:   false,
        DeclRange: blockRange,
    },
    ``,
},
```

Test case 3:

This test case (BUSL-1.1) is functionally equivalent to test case 2 in OpenTofu (MPL-2.0) and test case 3 in the "moved" (MPL-2.0) implementation.

Moved (MPL-2.0):

```
"modules": {
    &hcl.Block{
        Type: "moved",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: mod_foo_expr,
                },
                "to": {
                    Name: "to",
                    Expr: mod_bar_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Moved{
        From:      mustMoveEndpointFromExpr(mod_foo_expr),
        To:        mustMoveEndpointFromExpr(mod_bar_expr),
        DeclRange: blockRange,
    },
    ``,
},
```

Removed (OpenTofu, MPL-2.0):

```
"modules": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: mod_foo_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        From:      mustRemoveEndpointFromExpr(mod_foo_expr),
        DeclRange: blockRange,
    },
```

```
    ``,
},
```

Removed (Terraform, BUSL-1.1):

```
"modules": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: mod_foo_expr,
                },
            },
            Blocks: hcl.Blocks{
                &hcl.Block{
                    Type: "lifecycle",
                    Body: hcltest.MockBody(&hcl.BodyContent{
                        Attributes: hcl.Attributes{
                            "destroy": {
                                Name: "destroy",
                                Expr: hcltest.MockExprLiteral(cty.BoolVal(true)),
                            },
                        },
                    }),
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        From:      mustRemoveEndpointFromExpr(mod_foo_expr),
        Destroy:   true,
        DeclRange: blockRange,
    },
    ``,
},
```

Test case 4:

This test case deals with the lifecycle block, which again is unique to Terraform (BUSL-1.1) and has no equivalent one in OpenTofu (MPL-2.0) or the "moved" block (MPL-2.0).

Removed (Terraform, BUSL-1.1):

```
// KEM Unspecified behaviour
"no lifecycle block": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
```

```
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: foo_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        From:      mustRemoveEndpointFromExpr(foo_expr),
        Destroy:   true,
        DeclRange: blockRange,
    },
    ``,
},
```

Test case 5:

This test case (BUSL-1.1) deals with a missing argument. It is functionally equivalent to the OpenTofu (MPL-2.0) test case 3 and test case 4 in the "moved" block (MPL-2.0).

Moved (MPL-2.0):

```
"error: missing argument": {
    &hcl.Block{
        Type: "moved",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: foo_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Moved{
        From:      mustMoveEndpointFromExpr(foo_expr),
        DeclRange: blockRange,
    },
    "Missing required argument",
},
```

Removed (OpenTofu, MPL-2.0):

```
"error: missing argument": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{},
```

```
        }),
        DefRange: blockRange,
    },
    &Removed{
        DeclRange: blockRange,
    },
    "Missing required argument",
},
```

Removed (Terraform, BUSL-1.1):

```
"error: missing argument": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Blocks: hcl.Blocks{
                &hcl.Block{
                    Type: "lifecycle",
                    Body: hcltest.MockBody(&hcl.BodyContent{
                        Attributes: hcl.Attributes{
                            "destroy": {
                                Name: "destroy",
                                Expr: hcltest.MockExprLiteral(cty.BoolVal(true)),
                            },
                        },
                    }),
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        Destroy:   true,
        DeclRange: blockRange,
    },
    "Missing required argument",
},
```

Test case 6:

This test case (BUSL-1.1) deals with indexed resource instances. It is functionally equivalent to test case 4 in OpenTofu (MPL-2.0) and test case 2 in the "moved" block (MPL-2.0), but with an added error necessary to cover the differing functionality.

Moved (MPL-2.0):

```
"indexed resources": {
    &hcl.Block{
        Type: "moved",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
```

```
                "from": {
                    Name: "from",
                    Expr: foo_index_expr,
                },
                "to": {
                    Name: "to",
                    Expr: bar_index_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Moved{
        From:      mustMoveEndpointFromExpr(foo_index_expr),
        To:        mustMoveEndpointFromExpr(bar_index_expr),
        DeclRange: blockRange,
    },
    ``,
},
```

Removed (OpenTofu, MPL-2.0):

```
"error: indexed resources": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: foo_index_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        DeclRange: blockRange,
    },
    "Resource instance address with keys is not allowed",
},
```

Removed (Terraform, BUSL-1.1):

```
"error: indexed resource instance": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: foo_index_expr,
```

```
                },
            },
            Blocks: hcl.Blocks{
                &hcl.Block{
                    Type: "lifecycle",
                    Body: hcltest.MockBody(&hcl.BodyContent{
                        Attributes: hcl.Attributes{
                            "destroy": {
                                Name: "destroy",
                                Expr: hcltest.MockExprLiteral(cty.BoolVal(true)),
                            },
                        },
                    }),
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        From:      nil,
        Destroy:   true,
        DeclRange: blockRange,
    },
    `Resource instance keys not allowed`,
},
```

Test case 7:

This test case (BUSL-1.1) is similar to test case 6 (BUSL-1.1), but tests indexed modules instead of indexed resources and is necessary to cover the functionality. It is functionally equivalent to test case 5 in OpenTofu (MPL-2.0) which is based on test case 2 in the "moved" block (MPL-2.0).

Moved (MPL-2.0):

```
"indexed resources": {
    &hcl.Block{
        Type: "moved",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: foo_index_expr,
                },
                "to": {
                    Name: "to",
                    Expr: bar_index_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Moved{
```

```
        From:      mustMoveEndpointFromExpr(foo_index_expr),
        To:        mustMoveEndpointFromExpr(bar_index_expr),
        DeclRange: blockRange,
    },
    ``,
},
```

Removed (OpenTofu, MPL-2.0):

```
"error: indexed modules": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: mod_boop_index_foo_expr,
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        DeclRange: blockRange,
    },
    "Module instance address with keys is not allowed",
},
```

Removed (Terraform, BUSL-1.1):

```
"error: indexed module instance": {
    &hcl.Block{
        Type: "removed",
        Body: hcltest.MockBody(&hcl.BodyContent{
            Attributes: hcl.Attributes{
                "from": {
                    Name: "from",
                    Expr: mod_foo_index_expr,
                },
            },
            Blocks: hcl.Blocks{
                &hcl.Block{
                    Type: "lifecycle",
                    Body: hcltest.MockBody(&hcl.BodyContent{
                        Attributes: hcl.Attributes{
                            "destroy": {
                                Name: "destroy",
                                Expr: hcltest.MockExprLiteral(cty.BoolVal(true)),
                            },
                        },
                    }),
```

```
                },
            },
        }),
        DefRange: blockRange,
    },
    &Removed{
        From:      nil,
        Destroy:   true,
        DeclRange: blockRange,
    },
    `Module instance keys not allowed`,
},
```

The mustMoveEndpointFromExpr function

This function is identical in all three implementations and has been copied from the "moved" block (MPL-2.0) in both cases.

Moved (MPL-2.0):

```
func mustMoveEndpointFromExpr(expr hcl.Expression) *addrs.MoveEndpoint {
    traversal, hcldiags := hcl.AbsTraversalForExpr(expr)
    if hcldiags.HasErrors() {
        panic(hcldiags.Errs())
    }

    ep, diags := addrs.ParseMoveEndpoint(traversal)
    if diags.HasErrors() {
        panic(diags.Err())
    }

    return ep
}
```

Removed (OpenTofu, MPL-2.0):

```
func mustRemoveEndpointFromExpr(expr hcl.Expression) *addrs.RemoveEndpoint {
    traversal, hcldiags := hcl.AbsTraversalForExpr(expr)
    if hcldiags.HasErrors() {
        panic(hcldiags.Errs())
    }

    ep, diags := addrs.ParseRemoveEndpoint(traversal)
    if diags.HasErrors() {
        panic(diags.Err())
    }

    return ep
}
```

Removed (Terraform, BUSL-1.1):

```
func mustRemoveEndpointFromExpr(expr hcl.Expression) *addrs.RemoveTarget {
    traversal, hcldiags := hcl.AbsTraversalForExpr(expr)
    if hcldiags.HasErrors() {
        panic(hcldiags.Errs())
    }

    ep, diags := addrs.ParseRemoveTarget(traversal)
    if diags.HasErrors() {
        panic(diags.Err())
    }

    return ep
}
```

Conclusion

The allegations of copyright infringement are unsubstantiated as both the OpenTofu (MPL-2.0) and the Terraform (BUSL-1.1) implementations appear to be derivative works of the "moved" block implementation.