Nix lang v2

The Nix language unfortunately is full of little and big design accidents. Only so much can be fixed without breaking backwards compatibility.

Our goal is to design an improved Nix language revision, working title "Nix 2". To keep the scope manageable, the first iteration of language improvements will be restricted to be mostly backwards compatible and only require minimal migration effort. This allows us to test the process on a smaller scale, as well as allows us to get the quick and easy improvements out as soon as possible for others to use.

Join the discussion on Matrix: #nix-lang2:lix.systems

The rough action plan is:

1. Fork the grammar and gate its usage behind a feature flag.
2. Use the new grammar as a playground to experiment and implement fixes and improvements to the language, free of any constraints of backwards compatibility.
3. Figure out language versioning and prepare interoperability.
4. Provide a migration path, stabilize the new language, and make it available to users.

Initial language changes

Fixing floats

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/1979
- Confidence: High

Grammar: All floats must have a digit before the `.`. This is a hard requirement for making some of the other proposed syntax changes parse unambiguously in the first place.

Moreover, floating point semantics are currently broken in several ways. They need to strictly follow IEE754 double semantics instead.

Given that such a switch is not easy to make in a safe way, as an intermediate solution all floating point operations should be forbidden, effectively making floating point values opaque to the language.

Set slicing

Partially adapted from https://github.com/NixOS/rfcs/pull/110.

- Status: Draft implemented in https://gerrit.lix.systems/c/lix/+/1987
- Confidence: High

Sets can be sliced using `set.[key1, key2]` and `set.{key1, key2}`. The first returns a projection of the listed keys into a list, the second one a subset. All keys must be identifiers (or string identifiers), scoped to the attribute set.

[TBD: it is unclear as to whether interpolation is useful and how easy it is to implement] Identifiers may be interpolated: `set.[key1, ${key2}]` is equivalent to `[ set.key1, set.${key2} ]`, `set.{key1, ${key2}}` is equivalent to `{ key1 = set.key1; ${key2} = set.${key2}; }`.

Slicing into lists is a replacement for using `with`:

```
dependencies = python.pkgs.[
    arabic-reshaper
    babel
    beautifulsoup4
    bleach
    celery
    chardet
    cryptography
];
```

List and Set unpacking

- Status: Draft implementation in https://gerrit.lix.systems/c/lix/+/1988 and https://gerrit.lix.systems/c/lix/+/1989
- Confidence: Mid

In a list, elements which are lists themselves can be unpacked with the `*` operator. They will be concatenated in-place. `["hello", *list, "world"]` is equivalent to `["hello"] ++ list ++ ["world"]`

This can be easily combined with set slicing. The operator precedence facilitates patterns like the following:

```
  configureFlags = [
    "--without-ensurepip"
    "--with-system-expat"
    *(optionals (!(stdenv.isDarwin && pythonAtLeast "3.12")) [
      #  ./Modules/_decimal/_decimal.c:4673:6: error: "No valid combination of CONFIG_64,
CONFIG_32 and _PyHASH_BITS",
      # https://hydra.nixos.org/build/248410479/nixlog/2/tail
      "--with-system-libmpdec",
```

```
    ])
    *(optionals (openssl != null) [
      "--with-openssl=${openssl.dev}",
    ])
  ];
```

In a set, one can unpack elements like this:

`let baz = { bar = "foo"; }; in { foo = "bar"; *baz.{bar}; }`

This combines well with `optionalAttrs`:

```
{
  meta = with lib; {
    maintainers = with maintainers; [ matthewbauer qyliss ];
    platforms = platforms.unix;
    license = licenses.bsd2;
  };

  HOST_SH = stdenv'.shell;

  *lib.optionalAttrs stdenv'.hasCC {
    # TODO should CC wrapper set this?
    CPP = "${stdenv'.cc.targetPrefix}cpp";
  };

  *attrs;

  *lib.optionalAttrs (attrs.headersOnly or false) {
    installPhase = "includesPhase";
    dontBuild = true;
  };

  # Files that use NetBSD-specific macros need to have nbtool_config.h
  # included ahead of them on non-NetBSD platforms.
  postPatch = lib.optionalString (!stdenv'.hostPlatform.isNetBSD) ''
    set +e
    grep -Zlr "^__RCSID
    ^__BEGIN_DECLS" $COMPONENT_PATH | xargs -0r grep -FLZ nbtool_config.h |
        xargs -0tr sed -i '0,/^#/s//#include <nbtool_config.h>\n\0/'
    set -e
```

```
  '' + attrs.postPatch or "";

}
```

It also allows to have "local" `let` bindings for just some of the keys, without having to move them out of the entire attrset:

```
{
  key1 = "value1";
  *let
    stuff = "foo";
  in
  {
    inherit stuff;
    key2 = stuff;
  };
}
```

As with convential set declaration, duplicate keys are not allowed.

Note that the pattern of `inherit (foo) bar baz;` is equivalent to `*foo.{bar, baz};`.

Pipe operator function application: `|>`

This is being worked on in [RFC 148]

- Status: Implemented and released in Nix and Lix as an experimental feature flag `pipe-operator`
- Confidence: High

In `nixpkgs`, there is the `lib.pipe` function which will allows to write `g f a` as `pipe a [f g]`. Especially with deep nested and complicated data transformations, it makes the code flow from left to right and thus easier to read. Sadly, it is under-used because many people are not aware of it.

The fundamental problem it tries to solve though is that function calls are prefix, i.e. that a data processing chain with multiple entries is read from right to left. (Or, when adding parentheses, from the inside to the out side.)

Therefore, we introduce the `|>` operator. `a |> f |> g` is equivalent to `g(f(a))`.

List indexing

- Status: Not implemented yet
- Confidence: High

TODO link to RFC

Introduce `list.INDEX` on lists as syntax sugar for `builtins.elemAt list index`. ~~`list.${index}` interpolation for dynamic variables also works like it does for attribute sets.~~ To avoid type ambiguities at runtime, `ident.${expr}` is reserved for dynamic attribute access only, dynamic list indexing still requires using `builtins.elemAt`

~~Optional: We could even introduce `.last` `.tail` and `.length` as attributes. Need to think about that.~~ Is a bad idea because of dynamic typing.

Function list destructuring

- Status: Not implemented yet
- Confidence: Mid

The same way as function arguments can be destructured into an attrset with `{…}`, it should also work with lists. Some restrictions:

- Because order matters, arguments cannot have default values.
- Like with the attrset syntax, `...` indicates that the list may have more arguments.
- For now, the `...` must always be at the end. This restriction can easily be lifted some time in the future.
- Unlike in other languages, capturing the rest of the list (for example in head:tail patterns like in Haskell) is not possible because of performance considerations.

This, together with list indexing syntax, will make tuple-style code constructs a first-class citizen of the language. Replacing `nameValuePair` alone is expected to give significant performance gains (short lists are heavily optimized in the evaluator).

Disallow inner-attribute merging

- Status: Not implemented yet
- Confidence: Mid

Nix has syntax sugar for merging attrsets within attrset declarations: `{ a = {}; a.b = "hello"; }` will be fused into `{ a = { b = "hello"; }; }` at parse time.

This feature, only rarely used, does not compose well with other features like `rec` attrsets, leading to unintuitive semantics and potential foot guns: https://git.lix.systems/lix-project/lix/issues/350, https://github.com/NixOS/nix/issues/6251, https://github.com/NixOS/nix/issues/9020, https://github.com/NixOS/nix/issues/11268, https://md.darmstadt.ccc.de/xtNP7JuIQ5iNW1FjuhUccw#inherit-from-scopes-differently-than-inherit

Since these problems would be deeply aggravated by the new set unpacking syntax (defined below), it is best to completely remove this feature altogether. Since it only is convenience syntax sugar, no replacement syntax is necessary.

Expand `inherit` syntax

- Status: Not implemented yet
- Confidence: Low

The `inherit` syntax is adapted to be both more powerful and more consisten with the slicing syntax. The `inherit (from)` is made redundant and deprecated for removal in a future language revision. Inherit can also be used outside of attrsets and let bindings now, and will behave as if it was in a let binding.

```
inherit lib.{mkIf, types};
inherit {
  lib.mkif,
  types.{attrsOf, listOf, string}
};
# Mixing old with new style syntax: Do we want to allow this?
inherit
  lib.mkif
  types.{attrsOf, listOf, string}
;
# This only makes sense within attrsets really
inherit foo;
```

Proper keywords for `null`, `true` and `false`

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/1986
- Confidence: High

I don't know why these are builtins instead of keywords but at this point I assume it's because it was faster to implement.

Proper syntax nodes for all arithmetic expressions

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/1981
- Confidence: High

No more `__sub` and `__lessThan`. These had no reason but laziness to exist in the first place.

`?` and `or` operator

- Status: Draft implementation in https://gerrit.lix.systems/c/lix/+/1990
- Confidence: Mid
- Write `pkgs.foo.bar or default` as `pkgs ? foo.bar : default`, remove the `or` pseudo-keyword
- Unlike with `or`, no attribute access is needed: `value ?: default`

    - `?:` is more powerful than `or`, since it also works outside of `.`
    - [Optional] For consistency, function default arguments use `?:` instead of `?`
- `?:` has a lower priority than function application, which solves a lot of the confusion
- `?` operator for testing attribute set keys becomes a special case of `?:` without default value.
    - This does not change any of the semantics of `?`, but fixes the weird operator precedence as well
- [Optional] Introduce a new operator `.?`, also inspired by Kotlin. `foo.?bar` is equivalent to `if foo != null then foo.bar else null`.
    - C# uses `?.` instead

All line endings must be `\n`

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/1992
- Confidence: High

The current handling of `\r` is so jank that we'd better do without.

CRLF line endings are allowed within the file for Windows compat, but in strings the line endings get consistently normalized to LF only.

All files must be valid UTF-8 text

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/1991
- Confidence: High

The world runs on UTF-8, and most tools these days expect UTF-8 encoded input by default. There's no reason to allow other encodings or invalid byte sequences.

Sane escape sequences for strings

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/2089 and https://gerrit.lix.systems/c/lix/+/2104
- Confidence: Mid
- Escape sequences are restricted from *anything* to `\t`, `\r`, `\n`, `\"`, `\$`, `\\`, `\x…`, `\u{…}`
- `\` followed by a line break escapes it, a.k.a. string continuation escape (Rust)
- `$$` does not escape `$$` anymore, so `$${}` is now a dollar with an interpolation

Indented strings

Don't strip indentation of first line

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/2104
- Confidence: High

The current behavior is just weird, both for single-line strings (commonly used for unquoted `"`) and multi-line strings. The new behavior is also what Haskell does (in its new multiline strings proposal).

Indented strings work with tabs

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/2105
- Confidence: High

Programming languages may be opinionated, but making some features work only with space indentation is crossing a line.

Tabs and spaces can be mixed as part of the string's content, but not for the string's indentation. Indentation is calculated based on the longest common prefix.

Old cruft to remove

https://wiki.lix.systems/link/21#bkmrk-bad-ideas-for-featur

Remove unquoted URLs

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/1982
- Confidence: High

DSL or not, you'll survive typing those two additional extra characters.

Remove `let {}` syntax

- Status: Implemented in https://gerrit.lix.systems/c/lix/+/1980
- Confidence: High

And also the special `body` attribute.

`__override` special attribute

- Status: Implemented in TODO
- Confidence: High

No more magic attributes please. `__functor` is already bad enough.

Fix tokenization rules

https://md.darmstadt.ccc.de/xtNP7JuIQ5iNW1FjuhUccw?view=#token-boundaries-aren%E2%80%99t-real-and-will-hurt-you-cf-nix-iceberg

Status: Partially implemented in https://gerrit.lix.systems/c/lix/+/1984

Autocaller must die

Status: Not implemented

wtf?

```
❯ nix-build --expr '[[[ ({a}: [a]) ]]]' --arg a 'with import <nixpkgs> {}; hello' --no-link
fetching path input 'path:/nix/store/nyysli8lhjf03jgyvrf7mlxrlgnqn9qp-source'
/nix/store/kwmqk7ygvhypxadsdaai27gl6qfxv7za-hello-2.12.1
```

Future language changes

Some changes to the syntax would make large chunks of existing code invalid. These need to be postponed until proper versioning and migration tooling have been figured out.

Comma separated lists (confidence: high)

Currently list items are space separated. This has two major drawbacks:

1. This is inconsistent with most other language syntax features, which use `,` or `;` as item separator.
2. Not having them requires using parentheses around function calls in lists. Those are currently easy to forget, causing confusing type issues for beginners. (This would be less of an issue if we had a type system that could catch the mistake early on …)

Function declaration (confidence: low)

- `args@` now always also contains the default values (are there use cases where one strictly needs this not to be the case? Regardless, that behavior could be manually emulated if necessary)
- The `?` for defaults becomes `?:`
- Functions can also destructure list arguments: `[name, value]: _` as a replacement for `nameValuePair` and and to make tuples a first-class citizen (together with list indexing).
  - Note however that this change conflicts with comma separated lists because having both would cause too much lookahead in the parser.

NUL bytes must be supported within strings

Status: Blocked on rewriting the garbage collector to be compatible

0-terminated strings were a mistake, and we should not make any concessions in the language to implementations who use them. Especially when they're buggy.

Paths

Comments

While we are touching the syntax, let's leave some space here to discuss code comments.

- I like having the distinction between commented out code (syntax highlighting: unobtrusive) and commenting code (syntax highlighting: vivid).

- We should leave some room for semantic doc comments, should they ever come in Nix (TODO link respective discussions)
- There is this concept of "semantic comment" that comments out entire AST nodes. This is immensely useful even though few languages have it. (Caveat: the commented out code must at least be syntactically correct.)

TODO

Revision #8
Created 2024-08-18 20:38:02 UTC by piegames
Updated 2026-02-01 13:14:00 UTC by piegames