Solidity in 2022

Solidity Summit 2022, DEVConnect, Amsterdam

Christian Reitwiessner
@chriseth / @ethchris
chris@ethereum.org

Slides: https://chriseth.github.io/notes/talks/summit_2022/

Core Team

Leonardo Alt (leonardoalt)

Alexander Arlt (aarlt)

Mathias Baumann (marenz)

Alex Beregszaszi (axic)

Franziska Heintel (franzihei)

Daniel Kirchner (ekpyron)

Harikrishnan Mulackal (hrkrshnn)

Christian Parpart (christianparpart)

Christian Reitwiessner (chriseth)

Nishant Sachdeva (nishant-sachdeva)

Yashas Samaga (YashasSamaga)

Bhargava Shastry (bshastry)

Kamil Śliwak (cameel)

Damian Wechman (wechman)

General

- Language Server (`solc --lsp`)
- Documentation Translation
  - French, Indonesian, Persian finished
  - Chinese, Korean, Japanese being worked on
  - Portuguese, Spanish, Russian, Vietnamese started
- User Survey
- Underhanded Contest

The Solidity Yul IR Pipeline

Considered stable since 0.8.13 (2022-03-16)

```
solc --via-ir --bin --asm file.sol
solc --ir file.sol
solc --ir-optimized --optimize file.sol
```

- flexibility + auditability
- cross-function optimization
- comparative or even better gas costs

Please give us feedback!

Use with optimizer! Still a bit slow - working on it!

Errors and Encoding

```
error Unauthorized(uint authLevelRequired, uint authLevelPresent);

interface I {
    function fun(uint value, bytes memory input) external;
}

contract C {
    address target;
    bytes callDataStored;
    function f(bytes calldata input) public payable {
        if (!authorized(msg.sender))
            revert Unauthorized(10, authLevel(msg.sender));
        bytes memory forward = bytes.concat("msg", input[4:]);
        // Perform type checking, abi-encoding and
        // prefix function selector.
        callDataStored = abi.encodeCall(I.fun, (msg.value, forward));
    }
    ...
```

Immutable

```
contract Token is ERC20 {
    string public immutable name = "MyToken";
    uint public immutable totalSupply = 10**80;
    address immutable owner;

    // Not yet possible, but soon!
    ERC20[] immutable subTokens;

    constructor(ERC20[] memory _subtokens) {
        owner = msg.sender;
        subtokens = _subtokens;
    }

    function faucet(uint subTokenIndex) public {
        require(msg.sender == owner);
        subTokens[subTokenIndex].transfer(msg.sender, 10);
    }
}
```

Inline arrays

Very soon!

```
uint[] x = [1, 2, 3];
uint[3] y = [4, 5, 6];
uint[][3] z = [[1, 2], g(), []];
```

Much simpler with "viaIR" - no need to worry about stack layout constraints, it "just works"

```
// uint[] x = [1, 2, 3]
let t_1 := 1 // these will be moved by the optimizer
let t_2 := 2
let t_3 := 3
let x := allocate(0x20 + 0x60)
mstore(x, 3)
mstore(add(x, 0x20), t_1)
mstore(add(x, 0x40), t_2)
mstore(add(x, 0x60), t_3)
```

User-Defined Value Types (1)

No compiler support to avoid parameter ordering confusion:

```
function approve(uint tokenID, uint amount, uint startDate) public ...
// ...
c.approve(amount, startDate, tokenID);
```

Better:

```
function approve(TokenID tokenID, DAIAmount amount, Date startDate) public ...
// ...
c.approve(amount, startDate, tokenID); // !!! type error
```

But is it also cheap?

User-Defined Value Types (2)

```
// Introduces new type without any properties or implicit conversions,
// based on an underlying built-in value type.
// T.wrap / T.unwrap to convert from/to underlying type.
// Uses the underlying type in the ABI.
type TokenID is uint;
type Fixed is uint128; // no implicit conversion from literals!
uint128 constant FixedMultiplier = 10**18;
function uintToFixed(uint128 a) pure returns (Fixed) {
    return Fixed.wrap(a * FixedMultiplier);
}
...
```

User-Defined Value Types (3)

```
type Fixed is uint128;
...
// Add functions to the new type globally (only for types defined in the same file).
// No need to repeat "using".
using {add, mul} for Fixed global;
function add(Fixed a, Fixed b) pure returns (Fixed) {
    return Fixed.wrap(Fixed.unwrap(a) + Fixed.unwrap(b));
}
function mul(Fixed a, Fixed b) pure returns (Fixed) {
    uint result = (uint(Fixed.unwrap(a)) * uint(Fixed.unwrap(b))) /
        uint(FixedMultiplier);
    require(result <= type(uint128).max);
    return Fixed.wrap(uint128(result));
}
// In a different file:
function square(Fixed x) pure returns (Fixed) {
    return x.mul(x);
}
```

User-Defined Operators and Literals

Not yet, but soon:

```
type Fixed is uint128;
...
using {add as +, mul as *} for Fixed global;
function square(Fixed value) pure returns (Fixed) {
    return value * value;
}

// special "literal suffix" function
function _f(uint128 val, uint8 exp) pure returns (Fixed) {
    return Fixed.wrap(val * 10**(18 - exp));
}

function addVAT(Fixed value) pure returns (Fixed) {
    // Same as mul(value, _f(115, 2))
    return value * 1.15_f;
}
```

Standard Library

Main goal: as little "magic" in the compiler as possible

Make built-in functions importable and implement them in Solidity / inline assembly.

```
import {addmod} from "std/math";
```

"std/math":

```
function addmod(uint x, uint y, uint modulus) pure returns (uint result) {
    require(modulus != 0);
    assembly { result := addmod(x, y, modulus) }
}
```

No cluttering of global namespace and ability to look up behaviour.

Generics

Very useful for stdlib and user-defined types.

Powerful type-checking instead of pseudo-generics using `uint`.

Still needs more research, but Rust's generics look good.

```
function max<T: Comparable>(T x, T y) pure returns (T) {
    return x > y ? x : y;
}
struct ResizableArray<T> {
    uint capacity;
    T[] data;
}
function append<T>(ResizableArray<T> memory self, T memory item) {
    if (self.capacity <= self.data.length)
        self.data = reallocate.<T>(self.data, self.data.length * 2;)
    self.data[self.capacity] = item;
    self.capacity++;
}
```

Data enums

```
enum User {
    New(address account),
    Validated(address account, uint validations),
    Trusted(address account),
    Banned(address account)
}

function isValidated(User memory user) pure returns (bool) {
    if (user == User.Validated && user.Validated.validations > 3)
        return true;
    return false;
}
```

Main questions: Overlap data in storage for different enum values? Dangling references to data elements?

Also: Match expressions and `switch`!

Optimizer (1)

```
for (uint i = 0; i < myArray.length; i ++) {
    f(myArray[i]);
}
```

Two redundant checks because of `i < myArray.length`:

- overflow check in `i++`
- array-out-of-bounds check in `myArray[i]`

How to identify the redundant checks?

Optimizer (2)

Attempt 1: Use existing optimizer (symbolic execution engine)

Problem: No easy way to encode facts / assertions ("i < myArray.length")

Optimizer (2)

Attempt 1: Use existing optimizer (symbolic execution engine)
Problem: No easy way to encode facts / assertions ("i < myArray.length")

Attempt 2: External SMT solver
Problem: Reproducibility / external dependency / little control / much too powerful

Optimizer (2)

Attempt 1: Use existing optimizer (symbolic execution engine)
Problem: No easy way to encode facts / assertions ("i < myArray.length")

Attempt 2: External SMT solver
Problem: Reproducibility / external dependency / little control / much too powerful

Insight by Hari: Linear rational arithmetic powerful enough, can use Simplex - maybe even just difference logic
Attempt 3: Our own SMT solver: CDCL + Simplex
Advantages: Can output proofs via Farka's Lemma, full control, can keep it very simple but powerful enough.

Memory is Expensive and Tricky

Want to improve compiler's knowledge about lifetimes and references.

Allows deallocation and and improves safety.

Still early design phase.

```
function forward(MyContract c, bytes memory data) {
    c.f(copyof data);
}
function f1(MyStruct memory x) {
    MyStruct memory y = x;
    x.data = 2; // also changes y.data!
}
function f2(MyStruct memory x) {
    MyStruct memory y = ref x;
    // Modifying x is disallowed until y is destroyed.
    // x.data = 2;
}
function f3(MyStruct memory x) {
    MyStruct memory y = copyof x;
    x.data = 2; // does not modify y
}
```

Participate in Language Design!

- https://docs.soliditylang.org/en/latest/contributing.html
- Weekly Meetings
- 1-on-1 Feedback Sessions
- User Survey
- Forum (https://forum.soliditylang.org)
- Chat (https://matrix.to/#/#ethereum_solidity-dev:gitter.im)
- Twitter: @solidity_lang