Actor Lang

Release 1.1.0

Elyan Poujol (422404)

Mar 26, 2022

CONTENTS:

A small language I made to give life to the pseudocode one of my teacher used to teach us actor programming.

The language is pretty small so only basic arithmetic and comparisons can be done (and obviously standard actor programming actions). It only supports integers, booleans, strings and actor references.

CHAPTER

ONE

LICENCE

This software is released under the MIT licence. See the full licence text.

1.1 Language syntax

1.1.1 Comments

Comments can be added into code like so:

```
/* a comment */

/* A
 * multi-line
 * comment
 */
```

1.1.2 Basic types

- Integers: `10`, `-42`, `+1337`
- Strings: `"Hello, world!"`
- Booleans: `true`, `false`

1.1.3 Maths

Basic maths are supported so you can do arithmetic or boolean logic:

```
1 + 1
b * b - 4 * a * c
(1 + 2) * 3
true || false
true || false && myVar
x % 2 == 0
```

1.1.4 String concatenation

Simply «add» any value to a string to concatenate it:

```
"Hello" + "world!" --> "Helloworld!"
"2 + 2 is " + 4    --> "2 + 2 is 4"
"This is " + false --> "This is false"
```

1.1.5 Conditional execution

```
if (x == y) {
    ...
}

if (x < y) {
    ...
} else {
    ...
}
```

1.1.6 Loops

```
for (i in 0..10) {
    ...
}
```

Iterates in a range from 0 to 10 (inclusive).

You can also iterate on decreasing values:

```
for (i in 10..0) {
    ...
}
```

Iterates in a range from 10 to 0 (inclusive).

You can use «complex» expressions for the range bounds:

```
for (i in start..(a + b)) {
    ...
}
```

1.1.7 Printing data

```
display "hello";
display (1 + 2) * 3;
display true || false && true;
```

Prints three lines:

```
hello
9
true
```

To print values without new lines, you can use:

```
put "hello"
put 42
```

Prints:

```
hello42
```

1.1.8 Defining functions

Functions can be defined following this pattern:

```
fun <function name> (<arg 1>, ..., <arg N>) = <expression> | <statements>
```

Example:

```
fun mul(x, y) = x * y;

fun print(s) = {
    display s
};

fun spawnActor(x) = {
    a = create Actor(x);
    return a
}
```

Functions can be called as part of expressions or with the `call` statement:

```
fun print(value, printer) = send [value] to printer;

Printer () [value] = display value;
p = create Printer ();

v = F(x, y) * H(y) + x;
call print(v, p)
```

1.1.9 Actor behavior definition

An actor behavior definition follows this pattern:

```
<actor type> (<state var1>, <state varN>) [<message item1>, <message itemN>] =
↪<statements>
```

A behavior is executed when a message matching a pattern is received by an actor.

Example:

```
MyActor () [item1] = display item1;

MyActor (State) [item1, item2] = {
    display item1;
    display item2;
};
```

1.1.10 Tagged messages

To enable calling the right behavior of an actor with multiple behaviors of the same arity, one can tag the messages with literal values in the patterns.

Example:

```
MyActor () ["display-one"] = display 1;
MyActor () ["display-two"] = display 2;
```

These behaviors will be executed on receiving a message containing either `"display-one"` or `"display-two"`.

1.1.11 Changing behavior

An actor can change its type (and so its behavior) based on a received message:

```
Empty ()  ["set", x] = become Full (x);

Full (X) ["get", sender] = {
    send [X] to sender;
    become Empty ()
};
```

1.1.12 Sending messages

```
send [42] to anActor;

send ["hello", 1337] to anotherActor;
```

1.1.13 Actor self reference

In a behavior, the `self` variable is a reference to the actor executing the code.

1.1.14 Creating actors instances

Instantiating an actor from a given type with a given state (e.g. `MyActor (42)`) is done like so:

```
myInstantiatedActor = create MyActor (42);
```

1.2 Examples

Exemples can be found in the src/dist/examples folder.

1.3 Installation

To install `actorlang` you must unpack a distribution from a GitHub release or one you built yourself.

1.3.1 GitHub release

GitHub releases are available here.

1.3.2 From source

- First you need to build from source if not done yet: *Building from source*.
- Then unpack a distribution outputted in `<repo>/build/distributions/`.

1.4 Executing code

To execute code you must run the `actorlang` script with an Actor Lang file as an argument.

Example:

```
./actorlang ./examples/counter.actor
```

1.5 Building from source

To build the project simply run:

```
./gradlew build
```