Graph of Distributed Data Structures

Jakub Radek, Przemysław Roman

January 2023

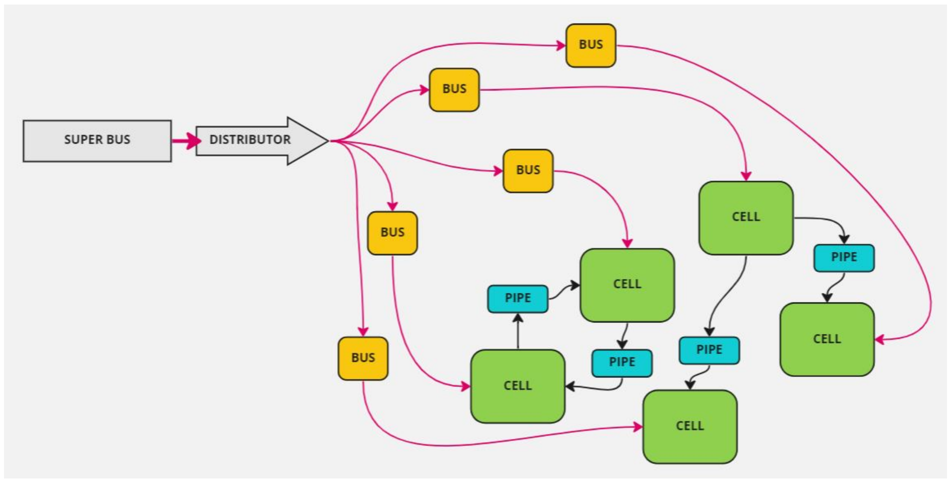
Contents

1 How to get started

```
git clone git@github.com:proman3419/Graph-of-Distributed-Data-Structures.git
cd Graph-of-Distributed-Data-Structures/goddslang
mvn package
alias godds="java -jar target/goddslang-1.0-SNAPSHOT-jar-with-dependencies.jar"
godds "src\main\resources\code snippets\example01_hello_world.godds"
```

2 Model

A standard program written in GoDDS can be featured as:



2.1 Super bus

It's used to input data to the program, implemented as a queue; in code #INPUT.

2.2 Distributor

It's used to distirbute data from the super bus to separate buses for each cell; in code #INPUT_CELLS. How does it work? Consider the following header:

```
#CELLS_COUNT 4
#CELLS_GRAPH
    0011
    1011
    0000
    1010
#END
#INPUT 10 16 23 110 21 45
#INPUT_CELLS 0 2 3 1 0
```

The key notes:

- there are 4 cells (with indexes 0, 1, 2, 3)
- super bus' initial state is [10, 16, 23, 110, 21, 45]
- the distributor's content is [0, 2, 3, 1, 0]

Steps of the algorithm:

1. 10 lands in cell 0. bus → [10]
2. 16 lands in cell 2. bus → [16]
3. 23 lands in cell 3. bus → [23]
4. 110 lands in cell 1. bus → [110]
5. 21 lands in cell 0. bus → [10, 21]
6. 45 lands in cell 0. bus → [10, 21, 45]

As we can see the value from the super bus at the ith index is added to the bus
with id INPUT_CELLS[i % len(INPUT_CELLS)].

2.3 Bus

Input source for a cell, implemented as a queue, after its initialization by distributing superbus contents no data can be written to it.
Each cell can access only the bus that's dedicated for it, reading values in order they were inserted.
Retrieving a value removes it from the bus, if no value exists operation is skipped.

2.4 Pipe

Mutable bus that allows a one-way communication between two cells.
Each cell can have multiple pipes but each pipe has to have a unique destination.
Two cells can both have a pipe connecting them to each other, which creates a two-way connection.

2.5 Cell

Entity that has two registers (R0, R1) which can store integer values.
The R0 register is the active one - most of the functions work with and/or modify its value.
The R1 register is meant for storing data.
Each cell may have code which it executes.

2.5.1 States

- Running - the cell is executing code
- Finished - the cell has finished executing code

2.5.2 Options

- Daemon - cell terminates if there are no more running non-Daemon cells
- Inactive - the cell doesn't run code (it's used only for storing it)

Options are prefixed with %, and are set after cell definition as shown in the example below:

```
#CELLS_COUNT 1
#CELLS_GRAPH
    0
#END
    #CELL 0 Cell0 %Inactive
    #END
#END_ALL
```

2.6 Code execution

Program runs in steps, for each step each cell calls one function.
Id of a cell determines its order, the smaller it is the higher the priority.
In other words: cell with id X will perform a step before a cell with id (X + 1).
When all cells have changed their states to Finished the program ends.

2.7 Call stack

The cells obey the following rules regarding jumps:

- If the label of a jump refers to the same cell which owns this call then the point from the cell had jumped isn't kept on the stack
- If the label of a jump refers to a neighboring cell then the point from the cell had jumped is kept on the stack

Check this example to better understand the difference.
The call stack's max depth is determined by the available RAM.

3 Sample program

3.1 Code

```
#CELLS_COUNT 3
#CELLS_GRAPH
    010
    101
    010
#END
#INPUT 5 72 101 108 108 111   // The first value is n = message's length
                              // followed by n values - the message
#INPUT_CELLS 0                // Write to cell 0
    #CELL 0 Client1
        READ_BUS              // Read n
        SWAP                  // Store it

        LABEL send_loop
            READ_BUS          // Read a character of the message
            WRITE_CELL Server // Send it to the server
            SWAP
            SUB 1             // Mark the character as sent
            IFEZ $            // Check if the message has been sent
            SWAP
            JUMP send_loop

        TERMINATE
    #END

    #CELL 1 Server %Daemon
        READ_CELL Client1     // Get a character
        WRITE_CELL Client2    // Forward the character
        JUMP ^
    #END

    #CELL 2 Client2 %Daemon
        READ_CELL Server      // Get a character
        PRINT "Received " 1
        PRINTNL_CHAR R0 0     // Print the character
        JUMP ^
    #END
#END_ALL
```

3.2 Preamble

```
#CELLS_COUNT VAL
#CELLS_GRAPH
    G[0,0]              G[0,1]              ...   G[0,CELLS_COUNT-1]
    .
    .
    .
    G[CELLS_COUNT,0]    G[CELLS_COUNT,1]    ...   G[CELLS_COUNT,CELLS_COUNT-1]
#END
```

The first line defines cells count VAL followed by a VAL x VAL matrix representing the cells' graph as an adjacency matrix.
Consider a value on the intersection of row X and column Y which denotes existence of a pipe X $\rightarrow$ Y:

- 1 - existent
- 0 - nonexistent

```
#INPUT VAL(s)
#INPUT_CELLS VAL(s)
```

Optional element, defines values contained in the superbus, order in which they will be read and distributes this content into each cell's bus according to input.
VAL in INPUT_CELLS defines id of a cell which will receive VAL set on corresponding index in INPUT.
Bus loops around if there are more cell ids given in INPUT_CELLS than values in INPUT.

3.3 Cell code

```
#CELL ID LABEL OPTIONS
    (Any combination of keywords creating cell code)
#END
```

Cell starts with #CELL keyword requiring ID, LABEL, optional OPTIONS, and ends with #END. Cell code can be written between #CELL and #END keywords. Cell ids have to be a number from permutation of numbers from the set: $\{0, 1, ..., <\text{CELLS\_COUNT}>-1\}$.
Cell label has to be unique when compared to other already existing cells.

3.4 Code end

```
#END_ALL
```

After definitions of all cell codes, #END_ALL has to be used, signifying the end of executable code.

4 Used nomenclature

```
<_REG_NAME_> can be one of:
    <R0>
    <R1>
```

Refers to a register's name.

```
<_REG_VALUE_>
```

Refers to a value stored in a register.

```
<_ARG_> can be one of:
    <_REG_NAME_>
    <VAL>
```

Refers to a math function argument where VAL can be any integer.

```
<_EXTENDED_LABEL_> can be one of:
    ^                     // Start of the local cell
    $                     // End of the local cell
    <LABEL>               // Label in the local cell
    <CELL_LABEL>@^        // Start of a specific cell
    <CELL_LABEL>@$        // End of a specific cell
    <CELL_LABEL>@<LABEL>  // Label in a specific cell
```

Refers to an argument for jump function, either local label for intracell jumps or label of different cell and label local to it's content for intercell jumps.

```
<_PRINT_ARG_> can be one of:
    <_REG_NAME_>
    "Any string"
```

Refers to an argument for print functions.

5 Functions

5.1 Supers

```
    #CELLS_COUNT VAL
```

Defines cell count for the program.

```
    #CELLS_GRAPH
    G[0,0]            G[0,1]            ...   G[0,CELLS_COUNT-1]
    .
    .
    .
    G[CELLS_COUNT,0]  G[CELLS_COUNT,1]  ...   G[CELLS_COUNT,CELLS_COUNT-1]
    #END
```

Defines adjecency graph representing neighbourhood of all cells.

```
    #INPUT VAL(s)
    #INPUT_CELLS VAL(s)
```

Optional element, defines bus of values and order in which they will be read, VAL in input cells defines id of cell which will receive VAL from on corresponding index in input. Bus loops around if there are more cell ids given in input cells than values in input.

```
    #CELL ID LABEL OPTIONS
```

Defines start of executable code for cell with ID, LABEL and optional OPTIONS.

```
    #END
```

Defines end of executable code for cell.

```
    #END_ALL
```

Defines end for executable code for entire program.

5.2 Modifying registry values

```
    ADD <_ARG_>
```

Adds the value <VAL> or <_REG_VAL_> to the value stored in R0 then saves the result in R0.
In short: R0 = R0 + <_ARG_>

```
    SUB <_ARG_>
```

Subtracts the value <VAL> or <_REG_VAL_> from the value stored in R0 then saves the result in R0.
In short: R0 = R0 − <_ARG_>

```
MUL <_ARG_>
```

Multiplies the value stored in R0 by <VAL> or <_REG_VAL_> then saves the result in R0.
In short: R0 = R0 ∗ <_ARG_>

```
DIV <_ARG_>
```

Divides the value stored in R0 by <VAL> or <_REG_VAL_> then saves the result (as integer) in R0.
In short: R0 = ⌊R0 / <_ARG_>⌋

```
MOD <_ARG_>
```

Executes modulo operation using the value stored in R0 and <VAL> or <_REG_VAL_> then saves the result in R0.
In short: R0 = R0 % <_ARG_>

```
ABS
```

Overwrites R0's value with its absolute value.
In short: R0 = |R0|

```
SET <_REG_NAME_> <VAL>
```

Sets the value of register <_REG_NAME_> to <VAL>.
In short: <_REG_NAME_> = <VAL>

```
COMP <VAL>
```

Compares the value stored in R0 to <VAL>, then saves the result (as integer) in R0.
The value saved is either -1, 0 or 1 depending on the values of R0 and <VAL>.
In short:
<VAL> < R0 → R0 = -1
<VAL> ≡ R0 → R0 = 0
<VAL> > R0 → R0 = 1

```
SWAP
```

Swaps the values stored in R0 and R1.
In short: R0 ↔ R1

```
COPY
```

Copies the value stored in R0 to R1.
In short: R0 → R1

5.3 Modifying order of calls

```
LABEL <NAME>
```

Defines a label that can be referenced by <NAME> when using the JUMP function.

```
JUMP <_EXTENDED_LABEL_>
```

Jump to the label <_EXTENDED_LABEL_> then continue execution from the point after its name. Jump can be either an intracell one, if no <CELL_LABEL> is given, or an intercell one if full <_EXTENDED_LABEL_> is used.

```
IFEZ <_EXTENDED_LABEL_>
```

If the value stored in R0 is equal to 0 then JUMP <_EXTENDED_LABEL_> will be performed.
In short: if R0 $\equiv$ 0 then JUMP <_EXTENDED_LABEL_>

```
IFNZ <_EXTENDED_LABEL_>
```

If the value stored in R0 is not equal to 0 then JUMP <_EXTENDED_LABEL_> will be performed.
In short: if R0 $\neq$ 0 then JUMP <_EXTENDED_LABEL_>

```
IFLZ <_EXTENDED_LABEL_>
```

If the value stored in R0 is less than 0 then JUMP <_EXTENDED_LABEL_> will be performed.
In short: if R0 $<$ 0 then JUMP <_EXTENDED_LABEL_>

```
IFGZ <_EXTENDED_LABEL_>
```

If the value stored in R0 is greater than 0 then JUMP <_EXTENDED_LABEL_> will be performed.
In short: if R0 $>$ 0 then JUMP <_EXTENDED_LABEL_>

```
EXIT
```

Stops execution of code from this cell. If the cell executed code from a different cell through an <_EXTENDED_LABEL_> this execution will halt and resume in earlier cell after the JUMP call.

```
TERMINATE
```

Stops execution of all code by the origin cell and changes its state to Finished.

```
PASS <VAL>
```

After encountering this keyword cell will stop it's execution for <VAL> steps.
It can be used without an argument in which case it defaults to 0.

5.4 Output

```
PRINT <_PRINT_ARG_> <FLAG>
```

In short:
if <FLAG> $\equiv$ 0 then print(<_PRINT_ARG_>)
if <FLAG> $\equiv$ 1 then print(<CELL_LABEL>: <_PRINT_ARG_>)
if <FLAG> $\equiv$ 2 then print(<CELL_LABEL>$\rightarrow$<_REG_NAME_>: <_REG_VALUE_>)

```
PRINTNL <_PRINT_ARG_> <FLAG>
```

In short:
if <FLAG> ≡ 0 then println(<_PRINT_ARG_>)
if <FLAG> ≡ 1 then println(<CELL_LABEL>: <_PRINT_ARG_>))
if <FLAG> ≡ 2 then println(<CELL_LABEL>→<_REG_NAME_>: <_REG_VALUE_>)

```
PRINT_CHAR <_REG_NAME_> <FLAG>
```

In short:
if <FLAG> ≡ 0 then print(chr(<_REG_VALUE_>))
if <FLAG> ≡ 1 then print(<CELL_LABEL>: chr(<_REG_VALUE_>))
if <FLAG> ≡ 2 then print(<CELL_LABEL>→<_REG_NAME_>: chr(<_REG_VALUE_>))

```
PRINTNL_CHAR <_REG_NAME_> <FLAG>
```

In short:
if <FLAG> ≡ 0 then println(chr(<_REG_VALUE_>))
if <FLAG> ≡ 1 then println(<CELL_LABEL>: chr(<_REG_VALUE_>))
if <FLAG> ≡ 2 then println(<CELL_LABEL>→<_REG_NAME_>: chr(<_REG_VALUE_>))

5.5 Intercell operations

```
WRITE_CELL <CELL_LABEL>
```

Writes value stored in R0 to the pipe between this cell and the one specified with <CELL_LABEL>.

```
READ_BUS
```

Reads a value from this cell's bus.

```
READ_CELL <CELL_LABEL>
```

Reads a value from the pipe between the cell specified with <CELL_LABEL> and this one, saving it to R0, if no value is found in the pipe then the cell awaits on this call.

```
COPY_CELL <CELL_LABEL>
```

Copy the value stored in R0 to R0 in cell specified with <CELL_LABEL>.

```
PRINT_LABEL_NAME <CELL_LABEL>
```

Prints label of cell specified with <CELL_LABEL>.
Pretty much useless, leftover from changing the argument from <CELL_ID>.

6 Technology stack

- ANTLR4
- Java 17
- Maven