РАЗРАБОТКА ГЕНЕРАТОРА MPI-ПРОГРАММ

Подготовил студент 4 курса ФИТ

Пирожков Андрей

Руководитель: Перепёлкин Владислав Александрович

ВВЕДЕНИЕ

- Генерация разных вариантов MPI-программ из фрагментов кода и заготовленных спецификаций без runtime системы – это один из способов конструирования фрагментированных программ

ЦЕЛЬ И ЗАДАЧИ

Цель: разработать программу, позволяющую получить работающий вариант MPI-программы из заготовленных спецификаций и фрагментов кода.

Задачи:

- Разобраться в исходном коде и выявить в чём причина некорректной работы алгоритма в некоторых ситуациях
- Разработка собственного способа построения последовательной программы
- Добавление средств для построения вариантов программ на MPI.

2.1

ЧТО ПОСТУПАЕТ НА ВХОД ПРОГРАММЫ?

- Реализация си-шных функций – фрагменты кода
- Файл спецификаций, перечисляющий какие переменные и какие операции должны использоваться в сгенерированной программе
- Файл спецификаций, описывающий подробно используемы операции (функции)

```c
#include <math.h>

void c_calc_area(double e1, double e2, double an, double *ar) {
    *ar = e1 * e2 * sin(an) * 0.5f;
}

void c_mul2(double x, double *y) {
    *y = x*2;
}

void c_mul(double x, double y, double *z) {
    *z = x*y;
}
```

2.2

ЧТО ПОСТУПАЕТ НА ВХОД ПРОГРАММЫ?

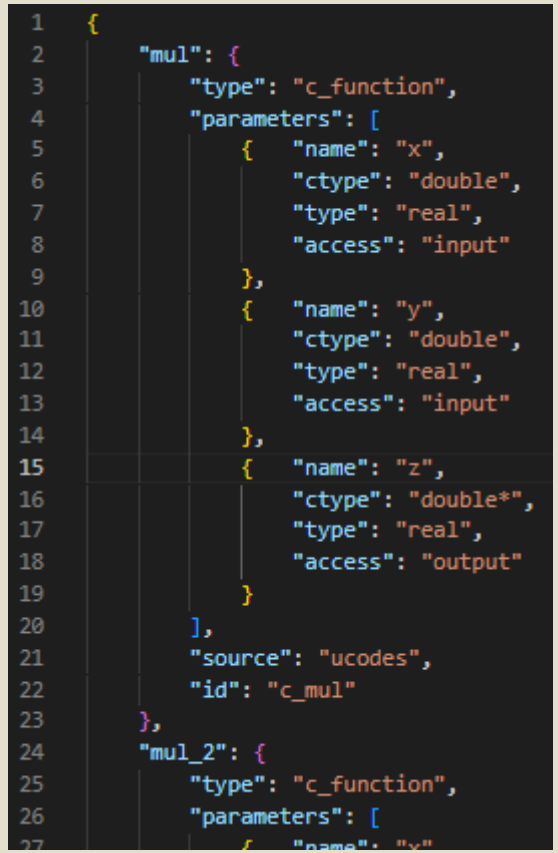
- Реализация си-шных функций – фрагменты кода
- Файл спецификаций, перечисляющий какие переменные и какие операции должны использоваться в сгенерированной программе
- Файл спецификаций, описывающий подробно используемы операции (функции)

```
{
    "using": ["funcs.json"],
    "interface": {
        "type": "cli1",
        "inputs": ["x", "y", "alpha"],
        "output": "S2"
    },
    "operations": {
        "calc_S": {
            "code": "calc_area",
            "inputs": ["x", "y", "alpha"],
            "outputs": ["S"],
            "arguments": {
                "edge1": "x",
                "edge2": "y",
                "angle": "alpha",
                "area": "S"
            }
        },
        "double_S": {
            "code": "mul_2",
            "inputs": ["S"],
            "outputs": ["S2"],
            "arguments": {
                "x": "S",
                "y": "S2"
            }
        }
    },
    "variables": {
        "x": "real",
        "y": "real",
        "alpha": "real",
        "S": "real",
        "S2": "real"
    }
}
```

2.3

ЧТО ПОСТУПАЕТ НА ВХОД ПРОГРАММЫ?

- Реализация си-шных функций – фрагменты кода
- Файл спецификаций, перечисляющий какие переменные и какие операции должны использоваться в сгенерированной программе
- Файл спецификаций, описывающий подробно используемы операции (функции)



```
{
    "mul": {
        "type": "c_function",
        "parameters": [
            {   "name": "x",
                "ctype": "double",
                "type": "real",
                "access": "input"
            },
            {   "name": "y",
                "ctype": "double",
                "type": "real",
                "access": "input"
            },
            {   "name": "z",
                "ctype": "double*",
                "type": "real",
                "access": "output"
            }
        ],
        "source": "ucodes",
        "id": "c_mul"
    },
    "mul_2": {
        "type": "c_function",
        "parameters": [
```

НЕКОТОРЫЕ СОСТАВЛЯЮЩИЕ ПРОГРАММЫ

- PartialProgram – строящаяся программа
- Sniplet: - определённые действия (функции)
  - ArgvSniplet – sscanf();
  - VarToStdoutSnip – if (rank == 0) printf();
  - InvokeOperationSnip – if (rank == 0) operation();
  - SendRecvSniplet – if (rank == 0) MPI_Send(); if (rank == 1) MPI_Recv();
- Port – аргументы к функции
- Mcell – ячейки памяти

ОГРАНИЧЕНИЯ ПРИ КОНСТРУИРОВАНИИ

- Вся сгенерированная программа содержится в единственной функции:
  - `int main(int argc, char **argv)`
- Пока поддерживается только вещественный тип double
- В начале объявляются все переменные, которые будут использоваться в программе
- Последним синглетом всегда является
  - `VarToStdoutSnip` - printf()

```c
#include <stdio.h>

// user codes declarations
void c_mul(double x, double y, double* z);
void c_mul2(double x, double* y);
void c_calc_area(double edge1, double edge2, double angle, double* area);

int main(int argc, char **argv) {
        double S2;
        double S;
        double x;
        double y;
        double alpha;
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        c_calc_area(x, y, alpha, &S); // calc_S
        c_mul2(S, &S2); // double_S
        printf("%lf\n", S2);
}
```

Пример сгенерированной последовательной программы

ПРИМЕРЫ

Основная задача – это написание программы, которая бы смогла конструировать разные варианты реализации MPI-программы

Разные варианты возможны:

- Выбор одного фрагмента кода, при многовариантности
- Выполнение фрагментов кода на разных узлах

```
int main(int argc, char **argv) {
        double S2;
        double S;
        double x;
        double y;
        double alpha;
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        c_calc_area(x, y, alpha, &S); // calc_S
        c_mul2(S, &S2); // double_S
        printf("%lf\n", S2);
}
```

```
int main(int argc, char **argv) {
        double S2;
        double S;
        double two;
        double x;
        double y;
        double alpha;
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        c_calc_area(x, y, alpha, &S); // calc_S
        sscanf(argv[4], "%lf", &two);
        c_mul(S, two, &S2); // mul_two
        printf("%lf\n", S2);
}
```

5.2

ПРИМЕРЫ

Основная задача – это написание программы, которая бы смогла конструировать разные варианты реализации MPI-программы

Разные варианты возможны:

- Выбор одного фрагмента кода, при многовариантности
- Выполнение фрагментов кода на разных узлах

```
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        if (rank == 1) c_calc_area(x, y, alpha, &S); // calc_S
        if (rank == 1) c_mul2(S, &S2); // double_S
        if (rank == 0) printf("%lf\n", S2);
        if (rank == 1) MPI_Send(&S2, 1, MPI_DOUBLE, 0, 309, MPI_COMM_WORLD);
        if (rank == 0) MPI_Recv(&S2, 1, MPI_DOUBLE, 1, 309, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        MPI_Finalize();
}
```

```
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        if (rank == 0) c_calc_area(x, y, alpha, &S); // calc_S
        sscanf(argv[4], "%lf", &two);
        if (rank == 1) c_mul(S, two, &S2); // mul_two
        if (rank == 0) printf("%lf\n", S2);
        if (rank == 1) MPI_Send(&S2, 1, MPI_DOUBLE, 0, 330, MPI_COMM_WORLD);
        if (rank == 0) MPI_Recv(&S2, 1, MPI_DOUBLE, 1, 330, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        MPI_Finalize();
}
```

6.1

КАК КОНСТРУИРУЮТСЯ ПРОГРАММЫ

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – 2-мя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell

6.2

КАК КОНСТРУИРУЮТСЯ ПРОГРАММЫ

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – 2-мя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell

```
      snip[0]
  VarToStdoutSnip

snip_id: 0
      port[0]
var_id: S2
binding: null
param_spec: input
```

6.3

КАК КОНСТРУИРУЮТСЯ ПРОГРАММЫ

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – 2-мя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell

snip[0]
VarToStdoutSnip

snip_id: 0
port[0]
var_id: **S2**
binding: null
param_spec: input

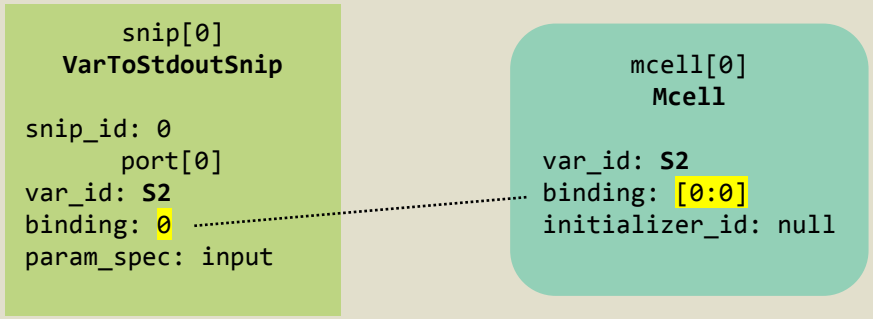
mcell[0]
Mcell

var_id: **S2**
binding: null
initializer_id: null

6.4

КАК КОНСТРУИРУЮТСЯ ПРОГРАММЫ

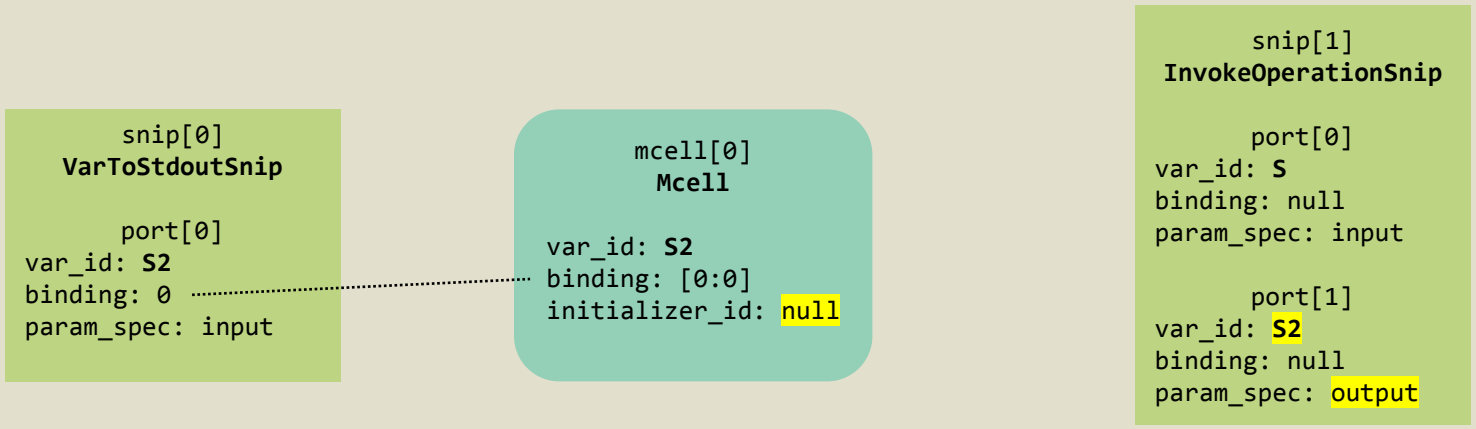
- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – 2-мя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



6.5

КАК КОНСТРУИРУЮТСЯ ПРОГРАММЫ

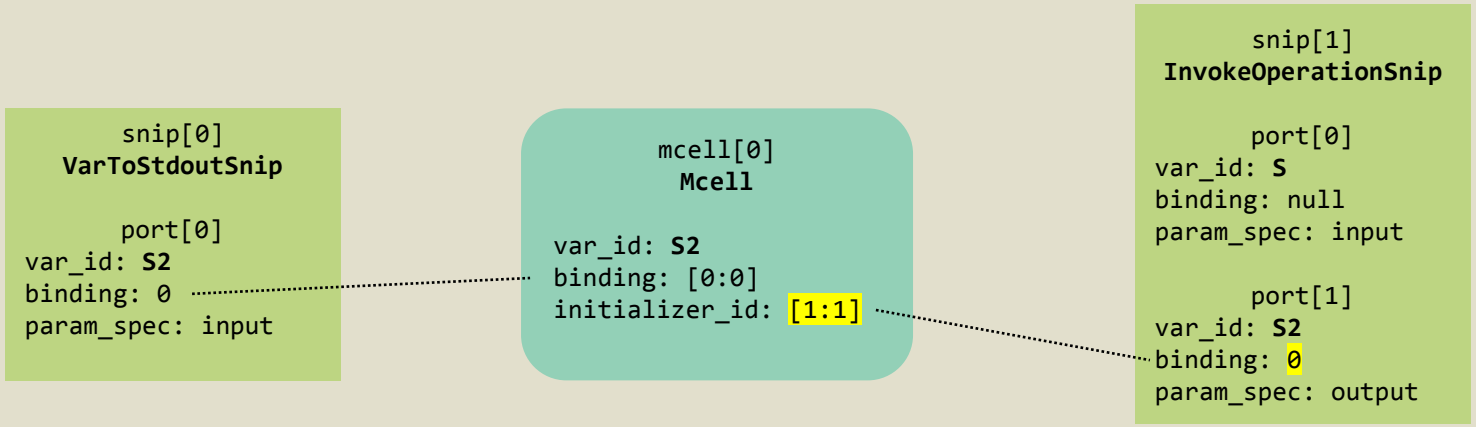
- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – 2-мя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



6.6

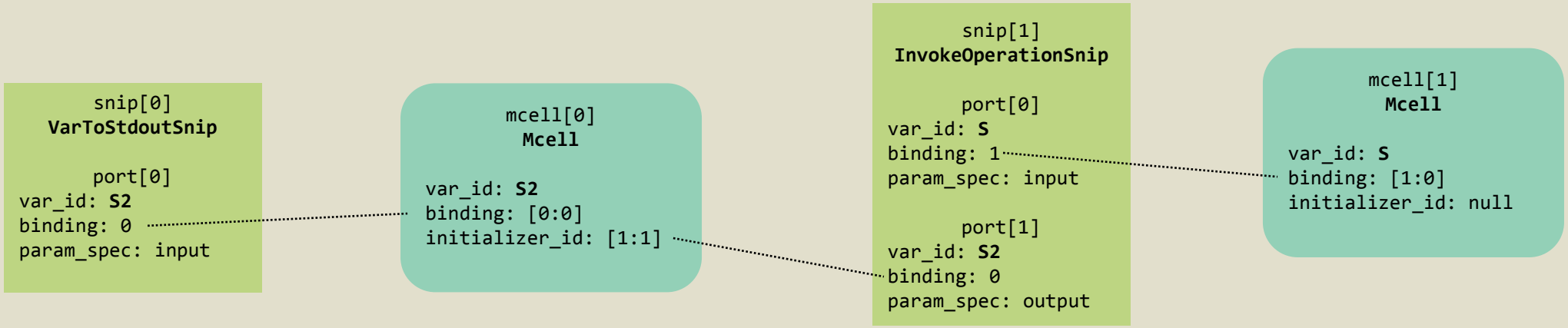
КАК КОНСТРУИРУЮТСЯ ПРОГРАММЫ

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – 2-мя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



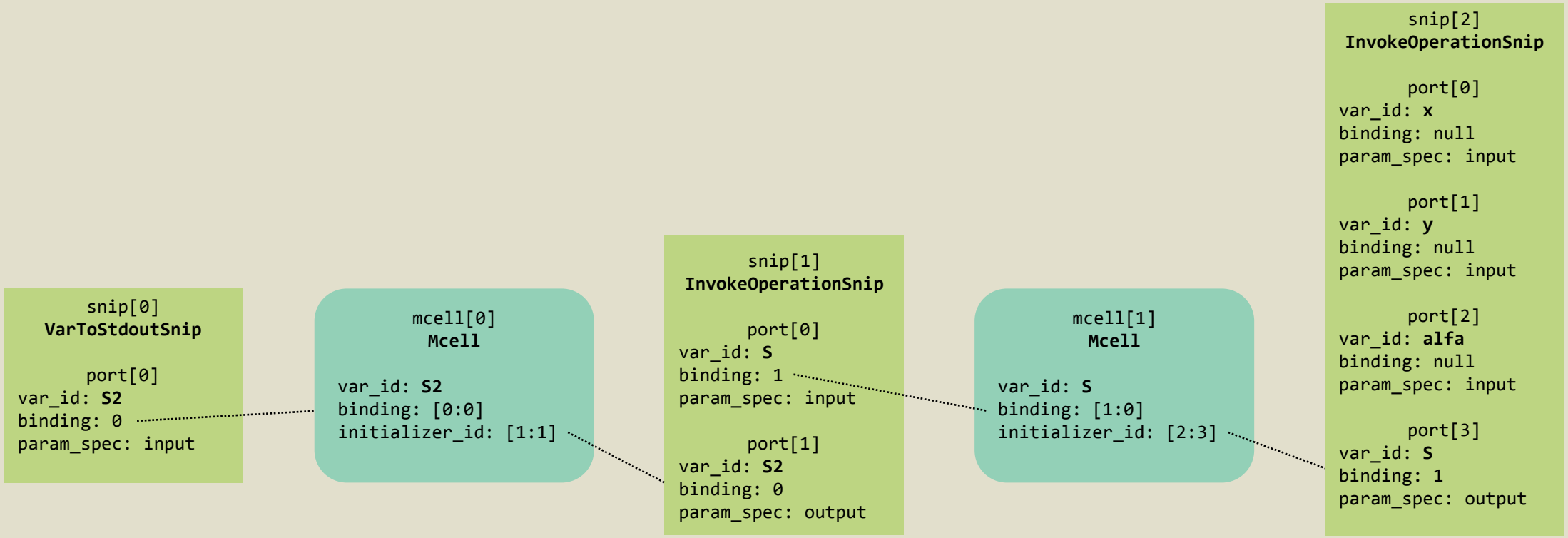
6.7

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – двумя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



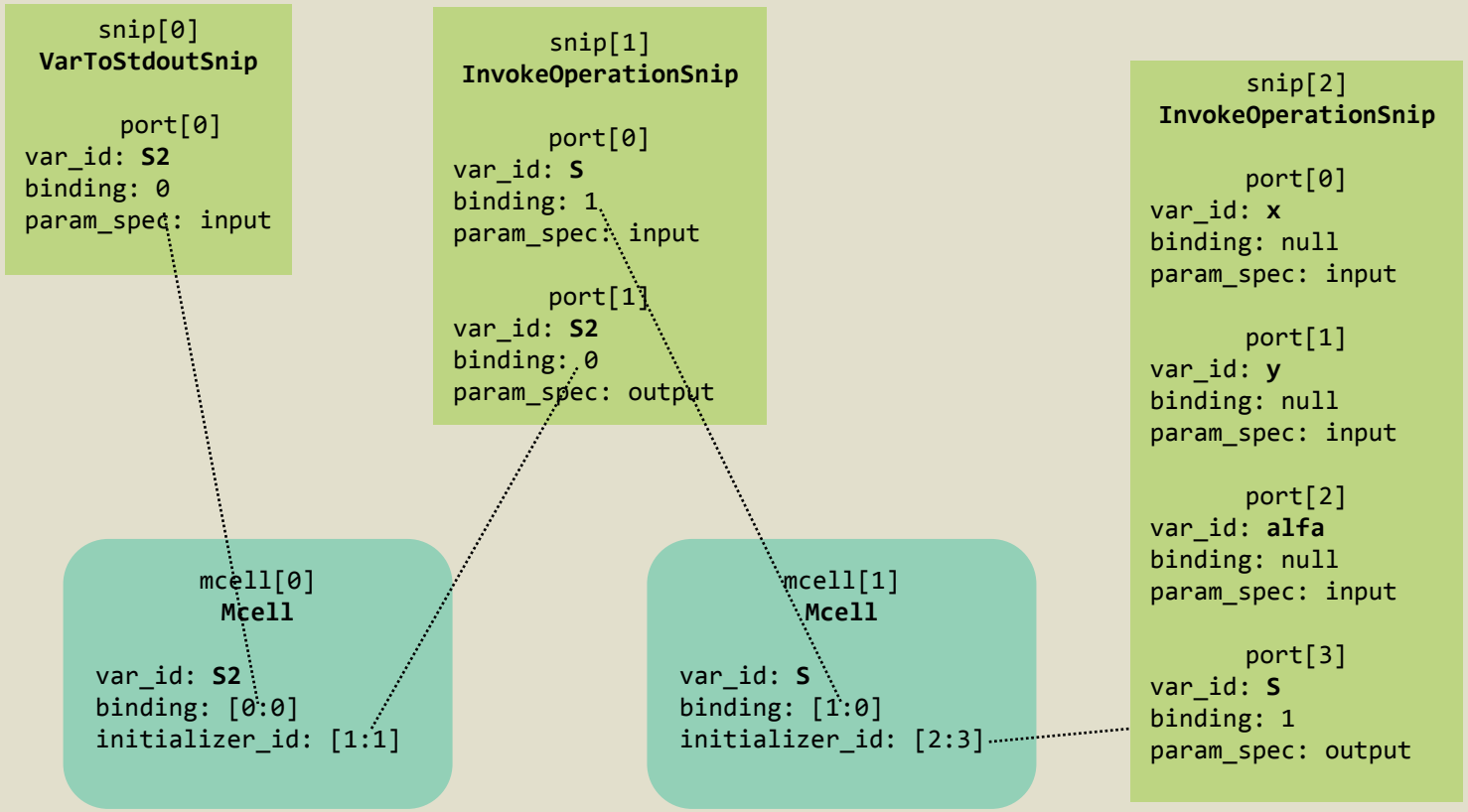
6.8

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – двумя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



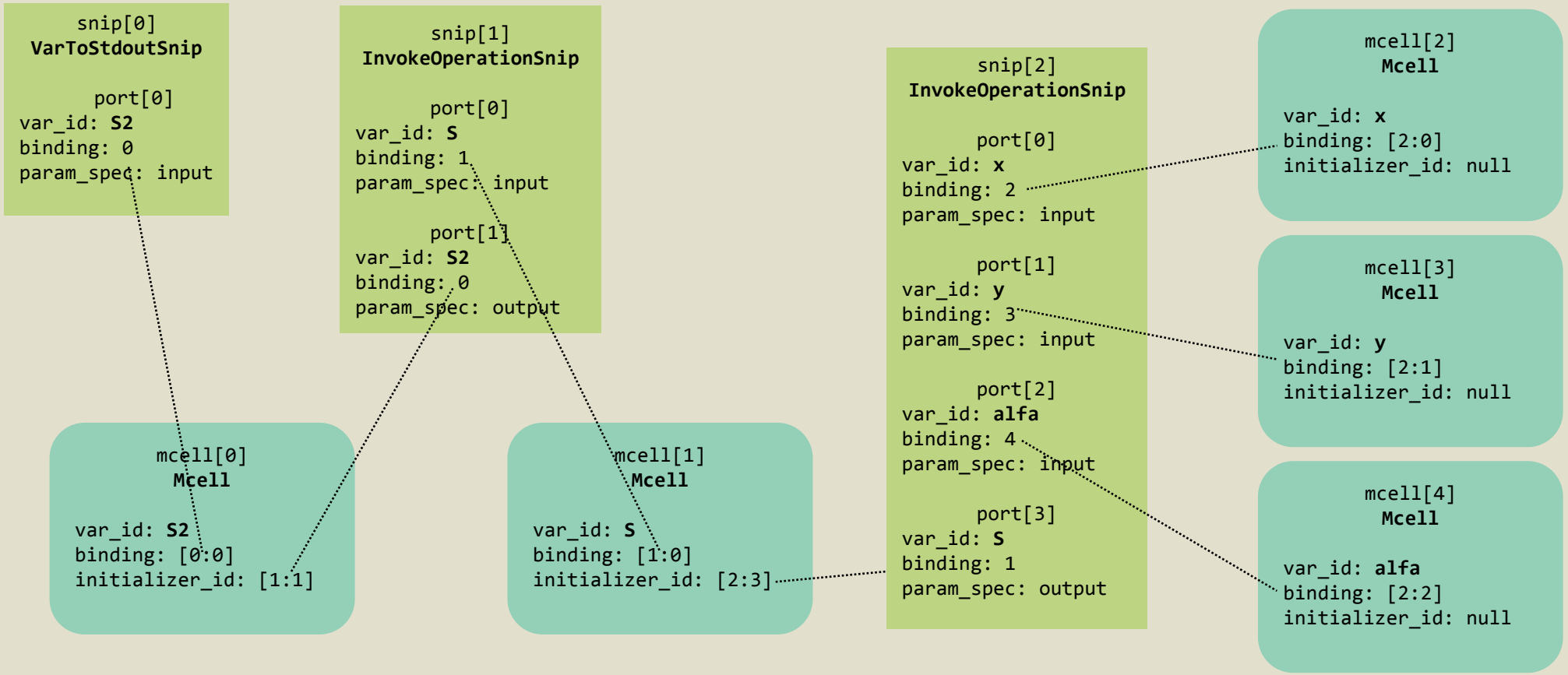
6.9

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – двумя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



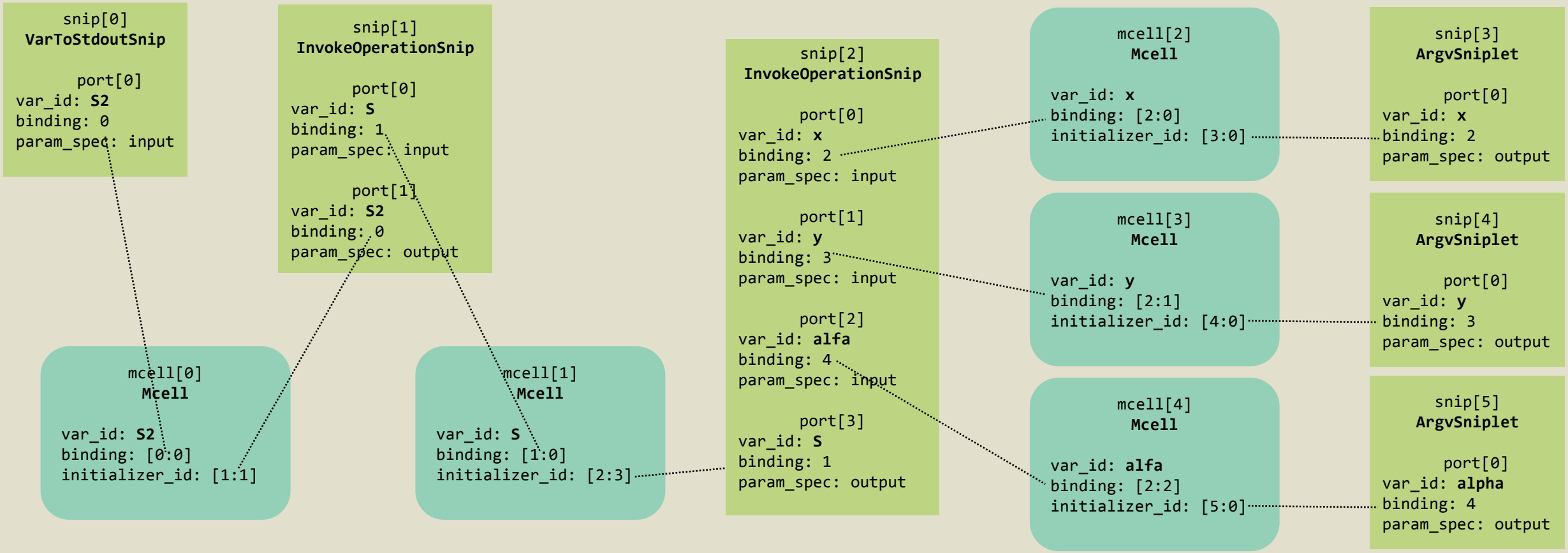
6.10

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – двумя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



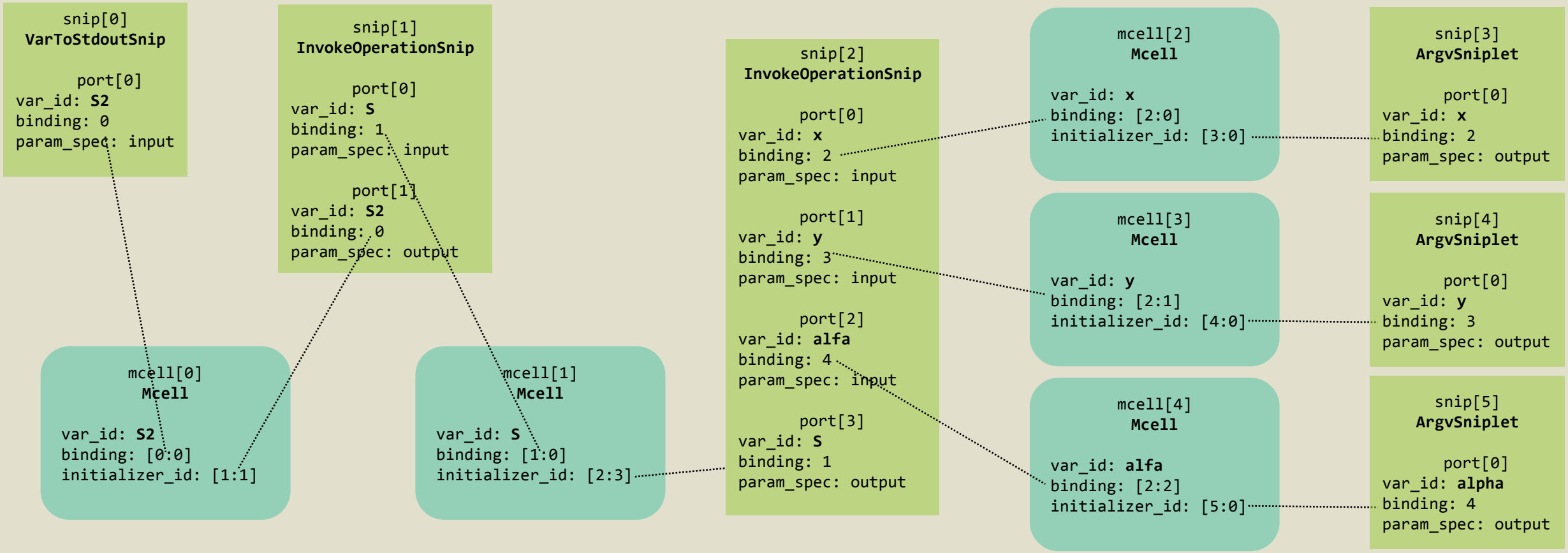
6.11

- Программа конструируется с конца, то есть с сниплета VarToStdoutSnip. Программа строится итерационно – двумя шагами:
- 1) Если есть входной свободный порт, то создаётся для него ячейка памяти (если такой нет) и делается bind
- 2) Если есть неинициализированный mcell, то добавляется нужный сниплет, где эта переменная может инициализироваться (output), затем сниплет биндится к mcell



7.1

КАК ГЕНЕРИРУЕТСЯ КОД ПО СХЕМЕ



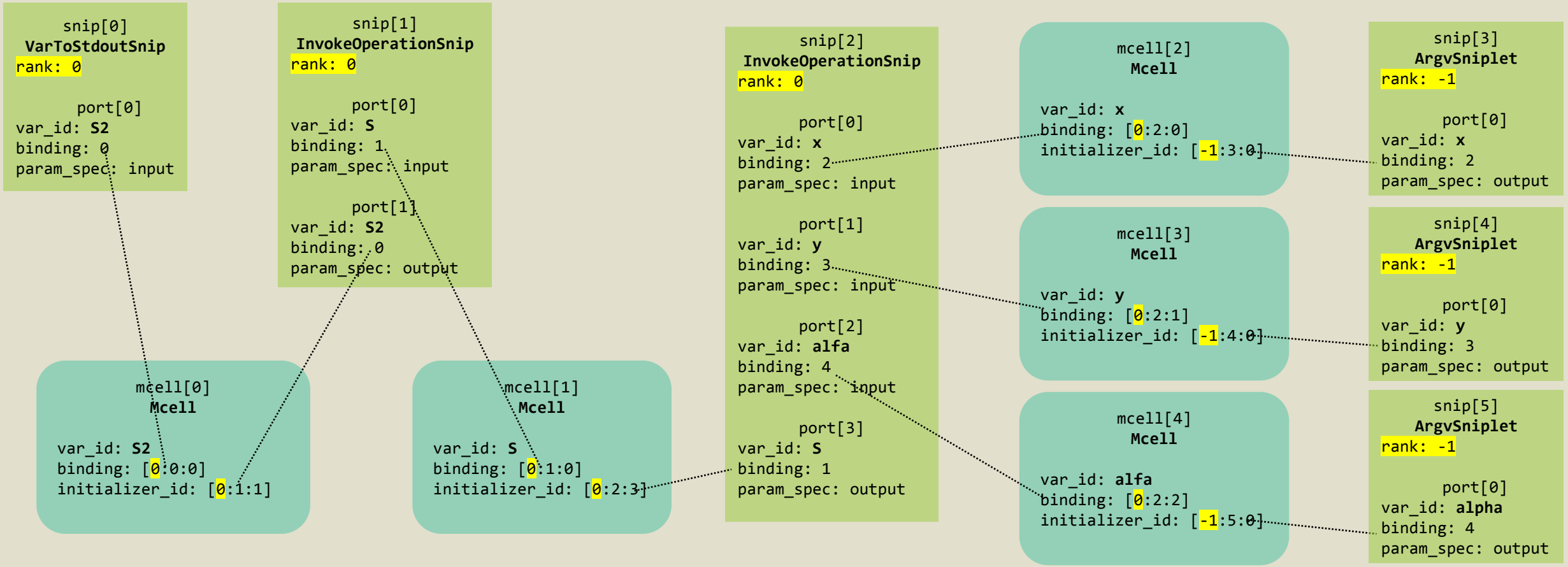
7.2

КАК ГЕНЕРИРУЕТСЯ КОД ПО СХЕМЕ

```
#include <stdio.h>

// user codes declarations
void c_mul(double x, double y, double* z);
void c_mul2(double x, double* y);
void c_calc_area(double edge1, double edge2, double angle, double* area);

int main(int argc, char **argv) {
        double S2;
        double S;
        double x;
        double y;
        double alpha;
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        c_calc_area(x, y, alpha, &S); // calc_S
        c_mul2(S, &S2); // double_S
        printf("%lf\n", S2);
}
```

8.1

КАК КОНСТРУИРУЕТСЯ MPI ПРОГРАММА



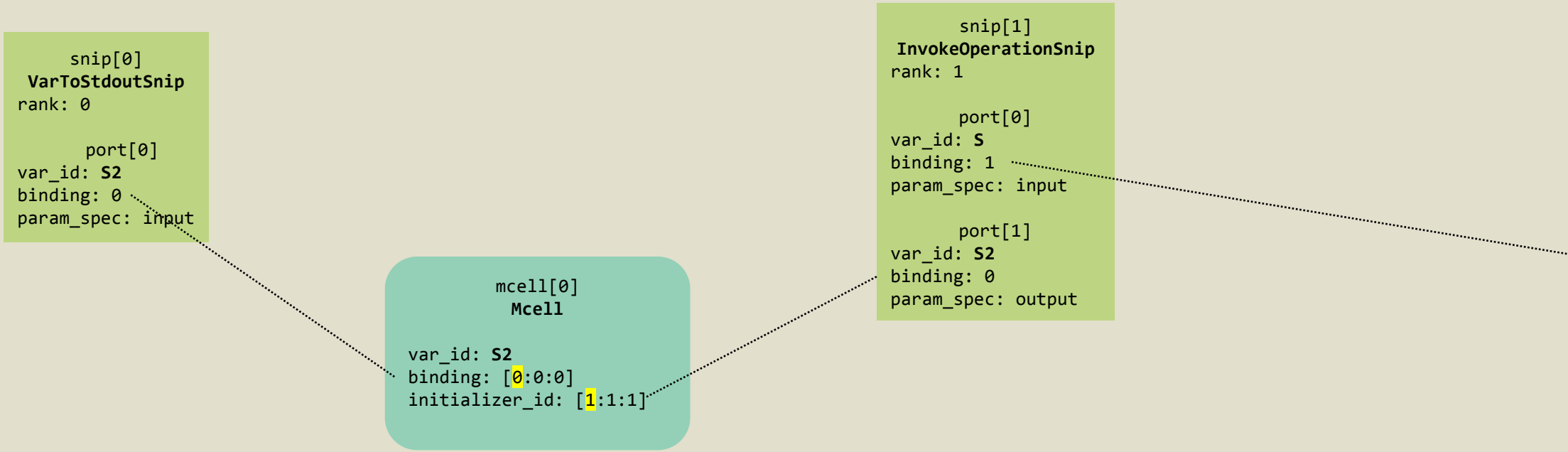
8.2

КАК КОНСТРУИРУЕТСЯ MPI ПРОГРАММА

```
#include <stdio.h>
#include "mpi.h"

// user codes declarations
void c_mul(double x, double y, double* z);
void c_mul2(double x, double* y);
void c_calc_area(double edge1, double edge2, double angle, double* area);

int main(int argc, char **argv) {
        int rank, size;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);

        double S2;
        double S;
        double x;
        double y;
        double alpha;
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        if (rank == 0) c_calc_area(x, y, alpha, &S); // calc_S
        if (rank == 0) c_mul2(S, &S2); // double_S
        if (rank == 0) printf("%lf\n", S2);

        MPI_Finalize();
}
```

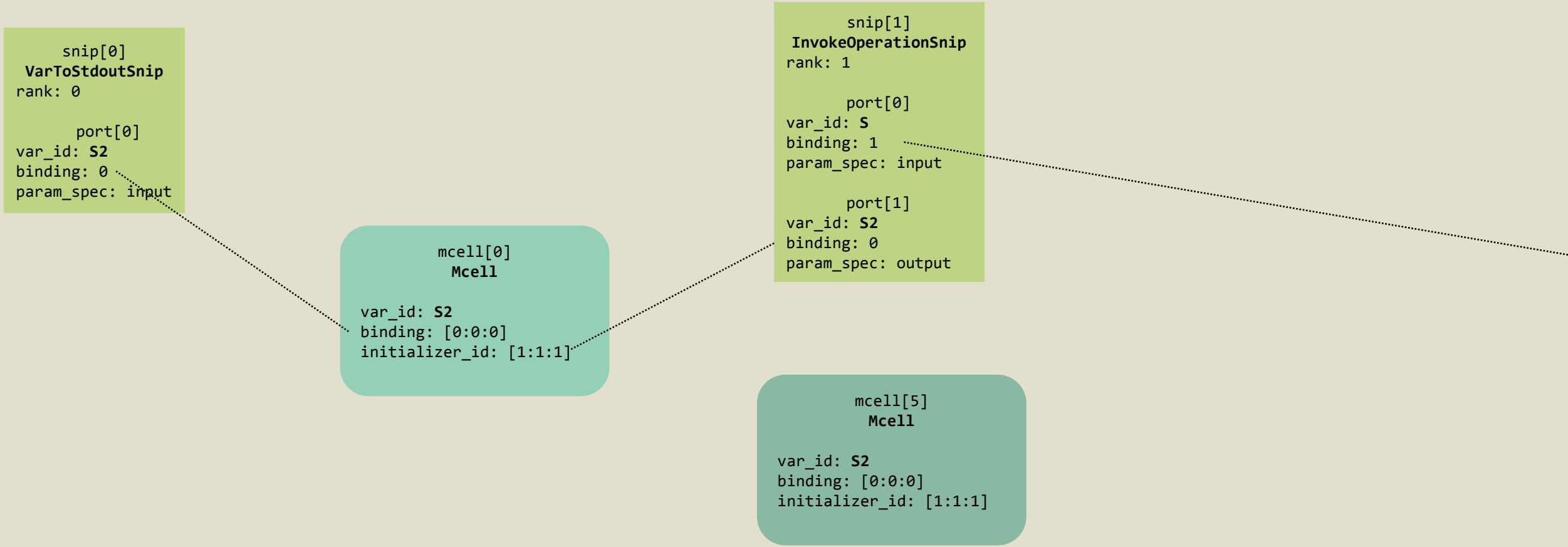
КАК КОНСТРУИРУЕТСЯ MPI ПРОГРАММА

- Сначала всегда конструируется последовательный вариант программы
- Затем решаются конфликты intitialize и bind (разные rank конфликтуют)



9.2

СНИПЛЕТ SENDRECVSNIPLET

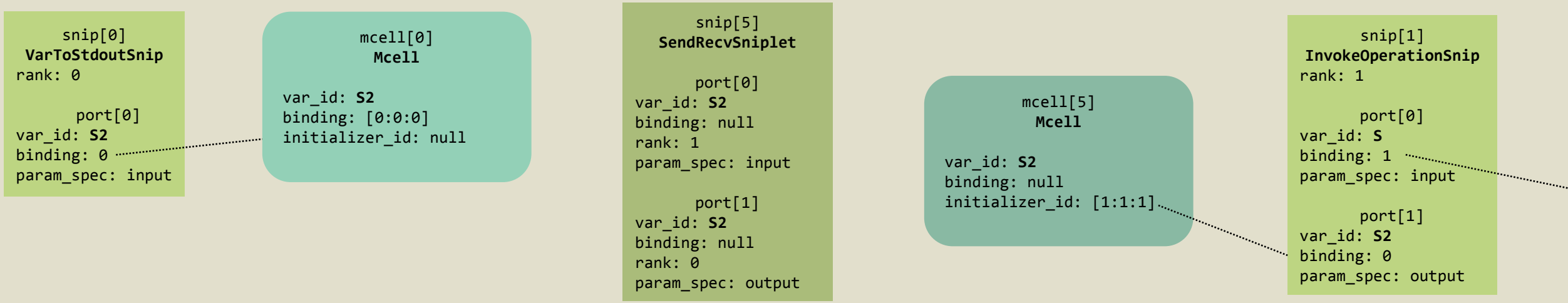


9.3

СНИПЛЕТ SENDRECVSNIPLET

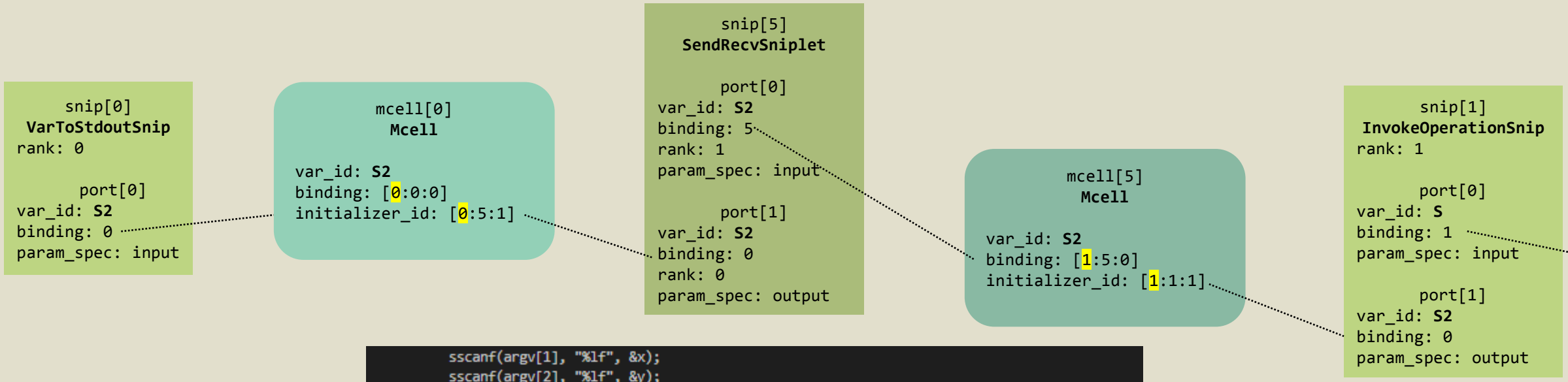
9.4

СНИПЛЕТ SENDRECVSNIPLET



9.5

СНИПЛЕТ SENDRECVSNIPLET




```
        sscanf(argv[1], "%lf", &x);
        sscanf(argv[2], "%lf", &y);
        sscanf(argv[3], "%lf", &alpha);
        if (rank == 1) c_calc_area(x, y, alpha, &S); // calc_S
        if (rank == 1) c_mul2(S, &S2); // double_S
        if (rank == 1) MPI_Send(&S2, 1, MPI_DOUBLE, 0, 309, MPI_COMM_WORLD);
        if (rank == 0) MPI_Recv(&S2, 1, MPI_DOUBLE, 1, 309, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        if (rank == 0) printf("%lf\n", S2);

        MPI_Finalize();
}
```

РЕЗУЛЬТАТЫ НА ТЕКУЩИЙ МОМЕНТ

- Исходный код был полностью разобран и доработан
- Реализован собственный способ построения последовательной программы – с конца
- Добавлено средство для конструирования программ на MPI

- Средство для построения программ MPI до конца не отлажено. В некоторых случаях не добавляется нужный сниплет. При генерации кода сниплет вставляется в не том порядке, каком следует