λ

TFPIE 2022
March 16th 2022

Functional programming learning path

Lidia V. Gorodnyaya,
Dmitry A. Kondratyev
A.P. Ershov Institute of Informatics Systems,
Siberian Branch of the Russian Academy of Sciences. Novosibirsk, Russia
e-mail: gorod@iis.nsk.su, apple-66@mail.ru

The experience of teaching

Teaching programming was always a primary concern of Institute of Informatics Systems and Novosibirsk State University.

An elective topic "Functional Programming" for MMF (Faculty of Mechanics and Mathematics) and FIT (Faculty of Information Technologies) students is taught and developing since 1990.

Distance courses

2004 - **Functional Programming Fundamentals**
https://intuit.ru/studies/courses/29/29/info
2006 - **Introduction to Lisp Programming**
https://intuit.ru/studies/courses/1026/158/info.
- **Programming paradigms**
https://intuit.ru/studies/courses/1109/204/info
2019 - **A new course** "Programming Paradigm"
based on the Moodle system.
https://www.iis.nsk.su/files/book/file/FIT-Gor-PP3.pdf

Functional programming learning path. **Lections & lab practice & lab assignments**

- lab practice sessions to understand **semantic fundamentals** (on Clisp),

- **pragmatics** by problem solving examples; to free programmer from solving problems that are not fundamental for the nature of tasks,

- **implementation** of **principles** in different functional programming languages (review lectures),

- **formalization** of the **principles** of the functional programming that distinguish it from other paradigms (based on students assignments reports) .

Skills, understanding, outlook

Relationships between **principles**,

consequences,

applications in **parallel programming**,

and **practical trade-offs** in functional programming**.**

The key idea:

the correctness of programmes

is more important

than the its effectiveness**.**

Step 1 of learning path

Functional programming is one of the first paradigms aimed not so much at obtaining an efficient implementation of pre-created and well-studied algorithms, but at a complete solution of **new and research problems.**

For functional programming, **priority is given to the organization of calculations and data structures,** while memory and process menagment are pushed to the periphery.

Semantic principles & consequence

Universality - to define debug-friendly generic functions, always to produce a result

Self-application - to achieve clear, understandable and concise definitions through the use of recursion

Equal rights of parameters — to choose independent parameters with a clear distinguish between bound and free variables

Meta-programming - to use actively and edit debugged templates

Verification — to show the role of proofs and axiomatization in ensuring the correctness of programs

The autonomy of developed modules — to show the advantages of multidimensional combinatorics

Pragmatic principles & consequence

The flexibility of restrictions - can be presented as vectors processing.

Data immutability - can be shown on the mechanisms of continuous software development systems.

The rigorous results – can show its advantages when familiarized with the definition of abstract machines.

Assumption of process continuity - the description of processes whose boundaries are difficult to predict.

Reversibility of actions - debugging programs dynamically, like UNDO in editing systems.

Unary functions - for turning program fragments into uniform flows of actions as single parameter functions

Step 2 of learning path

The framework of functional programming allows one to consider **independent streams** of represented data and, if necessary, reorganize the stream space.

With the transition to reusable programs and parallel computing, **the performance of program becomes more important** than their ubiquitous correctness.

In production, functional programming typically includes **practical trade-off mechanisms** that look like special functions in the programming language.

For parallel computing

Lazy evaluation - allows to avoid the cost of executing for too rare situations.

Identity of recalculations - for debugging programs and measuring their performance.

The iteration space - in the absence of relationships between iterations (supported in the **Sisal** language).

Automatic parallelization, load balancing, multi-pin nodes can be useful for solving problems of organizing parallel computing.

Most often automatic parallelization is supported.
Support for load balancing and multi-pin nodes **is not common.**

Productivity increase & Outside world

Data type control - analysis of data types.
Loop diagrams - familiar computational control scheme.
Memoization - can radically reduce the complexity of computations.

Programmable prediction - to evaluate actual boundaries on available resources, including time.
Pseudo-functions - access to external devices **for I/O**.
Data recovery - correct alternative to data immutability, aimed in case of performance problems to transition debugged programs to **more effective means without violating the correctness** of their processing.

Experiments demonstrating functional parallel programming principles to students

Let us consider the following task: **the sum of cubes of non-negative elements whose indices are even**.

Let us note that the *map*, *reduce*, and *select* functions allows simplifying solutions of many similar problems and allow parallel execution of such solutions.

A function-parameter of *map* and *select* should depend on element index to solve the considering problem.

But function-parameter of *map* and *select* depends only on sequence element.

Consequently, an important challenge is the generalization of the *map* and *select* functions.

Students consider this task and may suggest different solutions.

Experiments demonstrating functional parallel programming principles to students

The solution of the problem is the iteration space from the **Sisal** programming language.

Students can execute Sisal programs using cloud parallel programming system (CPPS).

An iteration space can be created using Sisal triplets or the Cartesian product of triplets.

A triplet defines an arithmetic progression.

The following construct is a triple construction expression: *lower_bound*..*upper_bound*..*step*

Examples of triplets are the following constructs:

- 0..n..1 is a sequence of natural numbers up to *n*;
- 0..n-1..2 is a sequence of indices of even elements of an array of length *n*.

Experiments demonstrating functional parallel programming principles to students

The following construct defines a Sisal **loop controlled by a range**:

for var in triplet do
returns reduction expr
end for

- *var* is a variable;
- *triplet* is a triplet;
- *expr* is a reducible expression (it may depend on *var*);
- *reduction* is a reduction (for example, *sum of*, *array of*).

The **semantics** of this loop may be described as follows:

- Each iteration corresponds to each triplet element (*var* corresponds to the value of the triplet element).
- The value of a single iteration is the value of the reducible expression.
- The value of a loop is the value of a reduction.

Experiments demonstrating functional parallel programming principles to students

The solution of the considering problem is the following function:

```
function sum_elements_even_indices(
                     a: array of integers
                     n: integer returns integer)
  for i in 0..n-1..2 do
  returns sum of
            if (a[i] >= 0)
               a[i]*a[i]*a[i]
               else 0
            end if
  end for
end function
```

This task demonstrates comparation of *map-reduce-select* approach and Sisal loops to students.

Experiments demonstrating functional parallel programming principles to students

Students can apply deductive verification approach to Sisal programs using the C-lighVer system

We suggest students to use symbolic method of verification of definite iterations for Sisal program deductive verification.

Symbolic method of verification of definite iterations is applied to special kinds of loops (definite iterations).

Body of a definite iteration is executed once for each element of data sequence.

Symbolic method of verification of definite iterations allows defining inference rules for these loops without invariants.

Symbolic replacement of definite iterations by **recursive functions (function *rep*)** is the base of this method.

We use ACL2 as theorem prover in the C-lightVer system.

Experiments demonstrating functional parallel programming principles to students

Let us consider the precondition of considering function written in the language of ACL2 system:

```
(and (integerp n) (< 0 n) (equal (length a) n)
  (integer-listp a))
```

The postcondition is **`(= result (reduce-sum-even-indices n a))`**.

Replacement of *result* term in postcondition by application of *rep* results in the following verification condition:

```
(implies
  (and (integerp n) (< 0 n) (equal (length a) n)
  (integer-listp a))
  (equal
   (rep (triplet 0 (- n 1) 2)) a)
   (reduce-sum-even-indices n a)))
```

ACL2 successfully proved it by induction on *n*.

This verification task demonstrates correspondence between iterations and recursion to students.

Step 3 of learning path

With the **rapidly expanding space** of programming languages and the development of a **multitude of programming paradigms**, it is necessary to familiarize students with **the trends in functional programming**, its prospects, its place among other paradigms.

Mostly these are historically significant and popular languages, including **Pure Lisp, Forth, Clisp, Sisal, Haskell, F#, Clojure, Scala, C#** and others.

A description of the **actual functional programming techniques** with an overview of functional programming languages and the **relationship of functional programming with other paradigms**.

Concluding Remarks

The focus of functional programming on solving many problems of organizing parallel computing.

Some questions have not yet received a practical answer - the discipline of working with shared memory.

Strictly speaking, **functional programming can play the role** of not only a training studio for the growth of professional programmer qualifications, but also a **design department** for production programming.

Any questions?

We thank you for your attention!

Gorodnyaya, L.V. The Role of Functional Programming in the Organization of Parallel Computing /CEUR Workshop Proceedings, 2021, 3066, p. 46–5

Gorodnyaya, L. Strategic Paradigms of Programming, Which Was Initiated and Supported by Academician Andrey Petrovich Ershov /Selected Papers 2020 5th International Conference on the History of Computers and Informatics in the Soviet Union, Russian Federation and in the Countries of Mutual Economic Assistance Council, SoRuCom 2020, 2020, p. 1–11

Gorodnyaya, L. Method of paradigmatic analysis of programming languages and systems /CEUR Workshop Proceedings, 2020, 2543, p. 149–158

Gorodnyaya, L.V. On the presentation of the results of the analysis of programming languages and systems /CEUR Workshop Proceedings, 2018, 2260, p. 152–166

Kasyanov V. Sisal 3.2: functional language for scientific parallel programming. Enterprise Information Systems. 2013. Volume 7. Issue 2. pp. 227-236.

Kasyanov V.N., Kasyanova E.V., Malishev A.A. Support tools for functional programming distance learning and teaching. Journal of Physics: Conference Series. 2021. Volume 2099. Article ID 012052.

Kasyanov V.N., Kasyanova E.V., Kondratyev D.A. Formal verification of Cloud Sisal programs. Journal of Physics: Conference Series. 2020. Volume 1603. Article ID 012020.

Distance courses (in Russian)

2004 - **Functional Programming Fundamentals**
https://intuit.ru/studies/courses/29/29/info
2006 - **Introduction to Lisp Programming**
https://intuit.ru/studies/courses/1026/158/info
- **Programming paradigms**
https://intuit.ru/studies/courses/1109/204/info
2015 — The educational and methodical **teaching aid** course "Programming
Paradigm" https://www.iis.nsk.su/files/book/file/FIT-Gor-PP3.pdf
2019 - The **new course** "Programming Paradigms" for FIT NSU
based on the Moodle system
https://el.nsu.ru/course/view.php?id=1200 (available at NSU)