Programming Languages: Project 1

This document describes the first project for the course Programming Languages (2022/23).

This file might be updated before the deadline (last changed 12 May 2023). Any changes that occur will be announced in Slack.

Quick Summary

- Goals:
  1. Extend a simple imperative language with a `break` statement
  2. Define the semantics of the extended language in two different ways
  3. Formally prove some relevant properties
- Deadline: **26 May 2023**
- To be done in groups of three students
- Submission is via Fénix (see instructions below)
- If you have any questions, please do not hesitate to contact the teaching team. You are encouraged to ask questions in the course's Slack (channel `#projects`)!

Tasks

0. Download the Project Pack from Fénix

You must work on the provided project pack (`P1-PL.zip`). The pack contains files that were taken/adapted from the Software Foundations book (Volume 1):

- `Maps.v`: defines total and partial maps
- `Imp.v`: defines the simple imperative language that will be extended
- `Interpreter.v`: this is where a step-indexed evaluator and proofs using it should be defined
- `RelationalEvaluation.v`: this is where the operational semantics should be defined (as a relation)
- `AdditionalProperties.v`: additional properties that relate the step-indexed and relational semantics
- `README.md`: file that you should fill in with your group's information and contributions (follow the template given)
- `INSTRUCTIONS.md`: this file

You should work on the files `Imp.v`, `Interpreter.v`, `RelationalEvaluation.v`, and `AdditionalProperties.v`. The files are marked with TODO comments to identify the places where you are supposed to work. You are also required to add information to the `README.md` file.

You can use the Makefile to compile the project (but note that it will only compile after you complete some of the tasks). If you add new files, you should edit the file `_CoqProject` and regenerate the Makefile (see the official documentation)

1. Extend Imp

You are required to extend the Imp abstract syntax with a `break` statement.

The book chapter on Simple Imperative Programs contains helpful information (see exercise `break_imp`).

Tasks You should edit the file `Imp.v` and perform the following tasks:

1. Extend the datatype `com` with the new construct `break`
2. Define a new notation for the new construct
3. Define examples `p1` and `p2` as specified below:

Program p1:

```
X := 1;
Y := 0;
while true do
  if X=0 then break else Y := Y+1; X := X-1 end
end
```

Program p2:

```
X := 1;
Y := 0;
while ~(X = 0) do
  Y := Y+1; X := X-1
end
```

2. A Step-Indexed Evaluator

2.1. Implementation Implement the step-indexed evaluator `ceval_step` in the file `Interpreter.v` so that it evaluates a program `c` given as parameter. In particular, you are required to use the following type for `ceval_step`:

```
Fixpoint ceval_step (st : state) (c : com) (i : nat): option (state*result)
```

The interpreter receives as parameters an initial state `st`, a program `c`, and an index `i` that limits the number of execution steps.

The type of the result is provided (defined in `RelationalEvaluation.v`):

```
Inductive result : Type :=
  | SContinue
  | SBreak.
```

The book chapter on Simple Imperative Programs contains helpful information about the `break` semantics (see exercise `break_imp`). The comments in the file `RelationalEvaluation.v` are also helpful (and taken from the book).

2.2. Properties You are required to prove the three properties stated without proof in the file `Interpreter.v`: `equivalence1`, `inequivalence1`, and `p1_equivalent_p2`. **Add a succint explanation in your own words of why `equivalence1` and `inequivalence1` are valid.**

The ImpCEvalFun chapter might guide you on how to implement the interpreter and how to structure your own proofs (also relevant for the file `AdditionalProperties.v`).

3. Relational Evaluation and Additional Properties

You are required to define a relational semantics for the extended Imp language (the `ceval` relation). The semantics is similar to the relational semantics shown in the Imp chapter, but here we deal with programs that can have `break` statements.

Read and follow the comments and explanations in `RelationalEvaluation.v`. As mentioned above, the book chapter on Simple Imperative Programs contains the same information (see exercise `break_imp`).

3.1. Proving properties of the relational semantics Prove the six properties stated without proof in the file `RelationalEvaluation.v`. Note that your semantics needs to satisfy these properties: if any of these properties becomes unprovable, you should revise your definition of `ceval`.

Add a succint comment before each property explaining the property in your own words.

3.2. Proving additional properties Prove all the properties stated without proof in the file `AdditionalProperties.v`.

Add a succint comment before each property explaining the property in your own words. Moreover, you are required to write the last proof using natural language (see TODO in the file).

4. Extras

You are encouraged to extend your work with more features. In terms of grades, the extensions might only be considered if everything else was attempted.

Here are some suggestions for extra features:

1. Improve the step-indexed evaluator so that: i) when it fails, instead of just returning `None`, it returns an appropriate error message; ii) when it succeeds, it shows the resulting state, but also the number of "steps" taken.

2. We have seen simple transformations and optimizations in the lectures that can also be applied to Imp. Implement a few optimizations that you find interesting and prove them correct with respect to the semantics defined.
3. Create a standalone interpreter for your extension of Imp by importing and expanding/adapting the chapters on Parsing and Extraction.

Submission

The project is due on the **26th of May, 2023**. You should follow the following steps:

- Submit only one file per group. Make sure your submitted file is named `P1-PL-GNN-2022.zip`, where `NN` is the group number. Always use two digits (e.g., Group 8's submitted file should be named `P1-PL-G08-2022.zip`).
- `PL-P1-GNN-2022.zip` is a zip file containing the solution and a `README.md` file where all group members and contributions are identified.
- Upload the file to Fénix before the deadline.

Assessment

To assess your submission, the following grid will be used:

Task	Marks (max)
README file properly filled in	0,25
Task 1 (Imp.v)	
Extend `com`	0,5
New notation	0,5
Examples `p1` and `p2`	0,5
Task 2 (Interpreter.v)	
Implementation of step-indexed evaluator	3
Proof of `equivalence1`	1
Proof of `inequivalence1`	1
Proof of `p1_equivalent_p2`	1
Task 3 (RelationalEvaluation.v)	
Definition of `ceval`	3
Proof of `break_ignore`	0,75
Proof of `while_continue`	0,75
Proof of `while_stops_on_break`	0,75
Proof of `seq_continue`	0,75
Proof of `seq_stops_on_break`	0,75
Proof of `while_break_true`	1,5
Task 3 (AdditionalProperties.v)	
Proof of `ceval_step_more`	1
Proof of `ceval_step__ceval`	1

Task	Marks (max)
Proof of `ceval__ceval_step`	1,5
Informal proof of `ceval_deterministic'`	0,5

If any of the above items is only partially developed, the grade will be given accordingly. If you are unable to finish a proof, you can hand in partially developed proofs by using `admit` or `Admitted`.

You are encouraged to comment your submission, so that we can understand your decisions. You might get additional points for that (e.g., if you describe in a comment exactly what needs to be done, even though a proof is incomplete).

Other Forms of Evaluation

After submission, you may be asked to present individually your work or to develop the solution of a problem similar to the one used in the project. This decision is solely taken by the teaching team.

Fraud Detection and Plagiarism

The submission of the project assumes the commitment of honour that the project was solely executed by the members of the group that are referenced in the files/documents submitted for assessment. Failure to stand up to this commitment, i.e., the appropriation of work done by other groups or someone else, either voluntarily or involuntarily, will have as consequence the immediate failure of all students involved (including those who facilitated the occurrence).