Universidade do Minho

Eduardo Brito* and Jorge Sousa Pinto**

* edbrito@di.uminho.pt ** jsp@di.uminho.pt

A Formal Approach for the mSPARK Programming Language

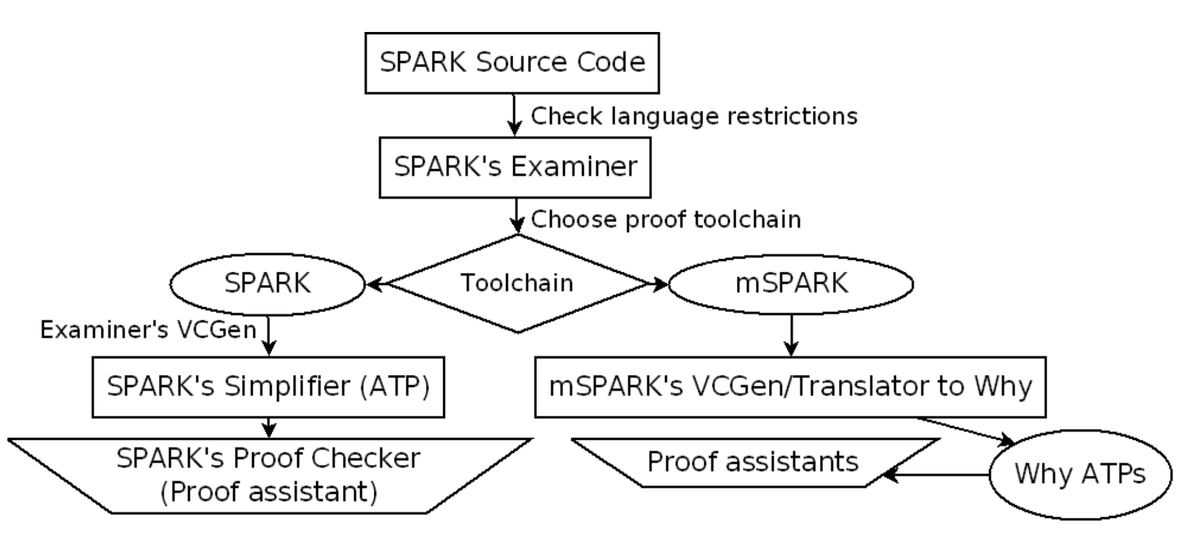
The mSPARK Language

SPARK is a known subset of the Ada programming language, with added source code annotations, and provides its own toolset to aid in the formal verification of safety and correctness of source code written with it. An example of SPARK and some of its features is shown below.

```
package Stack
--# own State: StackType;
is
   --# type StackType is abstract;

 procedure Pop;
   --# global in out State;
   --# derives State from State;
   --# pre 0 < Count_of(State);
   --# post Cap_of(State) = Cap_of(State~) and Count_of(State) =
Count_of(State~)-1 and
   --#      Substack(State, State~);
end Stack;

package body Stack
--# own State is Capacity, Ptr, Vector;
is
   type Ptrs is range 0..MaxStackSize;

   procedure Pop
   --# global in out Ptr;
   --# derives Ptr from Ptr;
   --# pre 0 < Ptr;
   --# post Ptr = Ptr~ - 1;
   is
   begin
      Ptr := Ptr - 1;
   end Pop;
end Stack;
```

Although it is used for formal verification, the underlying mathematics behind the tool that is used for formal reasoning is not described anywhere.

With the mSPARK language we propose to take a (large) subset of SPARK and formally describe it. This subset will be verifiable by the same tools as SPARK but we also propose a new toolchain for the language. A flowchart for this approach is shown below.



The Formal Approach

The first step to build a set of trustworthy tools for program verification is to have a formal description of the language. A standard way to do this is to describe the language in terms of operational semantics.

Having an operational semantics for a programming language is a very useful step for program verification. Given an operational semantics we can describe on top of it, its axiomatic semantics (in the form of a Hoare-like program logic) and check if this axiomatic semantics is sound.

After having a sound axiomatic semantics, we can then proceed to designing a verified verification condition generator. Thus, this enables a high degree of formalism and rigour.

Our Goal

With this approach we intend on having a complete formal description for a language and its program verification toolset, both of which will be given to the community for further scrutiny and enhancement.

We believe that this benefits a wide range of people, from academia to industry, by using a real programming language that is already being used in the industry in the development of safety- and mission-critical systems (though we are addressing a subset of the language, it is a large subset).

Another deliverable that we plan to have is an interpreter for the language. By using operational semantics (in the big-step style), having an interpreter is easily achieved. Although this is not a novel feature, we believe it is an interesting one to have.

Ackowledgements

This work is being partially funded by a research grant of the FCT-funded RESCUE project (PTDC/EIA/65862/2006).

References

Peter V. Homeier, David F. Martin 1994 Trustworthy Tools for Trustworthy Programs: A Verified Verification Condition Generator

Barnes, J. 2003 High Integrity Software: the SPARK Approach to Safety and Security. Addison-Wesley Longman Publishing Co., Inc.

Reynolds, J. C. 2009 Theories of Programming Languages. 1st. Cambridge University Press.

Jean-Christophe Filliâtre and Claude Marché 2007 The Why/Krakatoa/Caduceus platform for deductive program verification.

Hoare, C. A. 1983. An axiomatic basis for computer programming. Commun. ACM 26, 1 (Jan. 1983), 53-56.

O'Neil, Ian 1994 The formal semantics of SPARK83

CCTC
Centro de Ciências e Tecnologias da Computação
cctc.uminho.pt