Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 253 (2010) 71–81
www.elsevier.com/locate/entcs

Reversible Computation and Reversible Programming Languages

Tetsuo Yokoyama [1]

Department of Software Engineering, Nanzan University
Seirei-cho 27, Seto city, Aichi 489-0863, Japan

Abstract

A reversible programming language supports deterministic forward and backward computation. This tutorial focuses on a high-level reversible programming language Janus. In common with other programming paradigms, reversible programming has its own programming methodology. Janus is simple, yet powerful, and its constructs can serve as a model for designing reversible languages in general.

Keywords: Reversible computing, Reversible programming languages

1 Introduction

Conventional computing models such as Turing machines and random access machines (RAMs) destroy information at each computational step. The symbol written on the tape in the previous state will be overwritten by the new symbol, and the value written on the registers will be updated into the new one. At the first sight, we tend to think the destruction of information is necessary to computation. However, it was shown by Landauer that any irreversible computation can be simulated by reversible computation by adding the extra storage to remember the history of computation [16]. Moreover, this garbage information can be erased by its inverse computation [2]. Thus, in theory we can simulate any irreversible computation with reversible computation provided that a given storage is infinite.

When a conventional computation is physically performed information destruction has a physical cost in the form of heat dissipation. Conversely, if no bit is erased during computaion, in theory there is no lower bound of heat dissipation for the computation. Therefore, the research of reversible computing has some potential

[1] Email: tetsuo@se.nanzan-u.ac.jp

[2] This work is partly supported by EPSRC grant EP/G039550/1, JST CREST and Nanzan University Pache Research Subsidy I-A-2 for the 2009 academic year.

doi:10.1016/j.entcs.2010.02.007

applications such as the low-power CMOS and quantum computing. Note that any quantum computing is necessary to be reversible.

This tutorial focuses on a high-level reversible programming language Janus. In common with other programming paradigms, reversible programming has its own programming methodology. We define the language and give its syntax and operational semantics.

2 The Reversible Language Janus

The imperative language Janus appears to be the first reversible structured programming language: it was invented by Lutz and Derby [17], but remained unpublished for two decades. The language presented here extends our original formalization [32] and has been presented in [30]. Janus is simple, yet powerful, and its constructs can serve as a model for designing reversible languages in general. The main difference from conventional programming languages is that all assignments and control constructs are purely reversible, and the language's inverse semantics can be accessed by uncalling procedures (i.e., executing them backward).

2.1 Example Program: Fibonacci Pairs

To provide a flavor of reversible programming, we show a Janus procedure for computing *Fibonacci pairs*. Given an integer `n`, the procedure `fib` computes the (`n+1`)-th and (`n+2`)-th Fibonacci number. For example, the Fibonacci pair for $\texttt{n} = 4$ is $(5, 8)$. Returning a pair of Fibonacci numbers makes the otherwise non-injective Fibonacci function injective. Variables `n`, `x1`, `x2` are initially set to zero. Parameter passing is pass-by-reference.

```
procedure fib(int x1,int x2,int n)
   if n=0 then x1 += 1
               x2 += 1
          else n -= 1
               call fib(x1,x2,n)
               x1 += x2
               x1 <=> x2
   fi x1=x2

procedure fib_fwd(int x1,int x2,int n)
   n += 4
   call fib(x1,x2,n)     // forward execution

procedure fib_bwd(int x1,int x2,int n)
   x1 += 5
   x2 += 8
   uncall fib(x1,x2,n)   // backward execution
```