

Orc 0.5 Users Guide

William R. Cook and Javadev Misra

August 18, 2005

1 Introduction

This document describes the Orc programming system. It describes how to install the system and execute programs. It also defines the current concrete syntax of Orc that is recognized by the system. This document assumes a working knowledge of the purpose and semantics of Orc; these are described in other documents [4, 1, 3].

The Orc system is under active development; it currently does not have nearly all the features that we would like. However, the current implementation is a very clean foundation on which to build these additional features.

The source code is publicly available under GPL. The system is written in Java and includes javadoc documentation. In addition, a high-level description of the implementation is also available [2].

2 Installation

Unpack the `orc.zip` file into any directory on your system. The resulting directory structure is as follows:

<code>README.txt</code>	readme file
<code>license.txt</code>	license agreement
<code>orc</code>	command to run Orc on unix
<code>orc.bat</code>	command to run Orc on windows
<code>doc/</code>	documentation (including this document)
<code>lib/</code>	executable libraries (jar files) used by Orc
<code>test/</code>	example scripts

3 Running Orc

To run Orc, simply execute the command

```
orc filename
```

where *filename* is a file containing an Orc program. The file contains a set of definitions and one goal expression.

To enable Orc to be run from any directory, use one of the following:

- Add the directory containing the `orc` command to your path.
- Create an alias to the appropriate file (`orc` on Unix or `orc.bat` on Windows).
- Specify the path of the Orc command, as in `myorc/orc`.

Orc reads from the standard input if no filename is given. This can be used to execute Orc from within Emacs.

```
orc < filename
```

Debugging information can be printed using the `-debug` flag as the first argument.

```

expr ::= { def } goal

def ::= "def" id [ formals ] "=" expr
formals ::= "(" id { "," id } ")"

goal ::= par [ "where" binding { ";" binding ]
binding ::= id "in" par

par ::= seq { "|" seq }

seq ::= basic { ">" ["!"] [id] ">" basic }

basic ::= id | number | string | call | block
id ::= [a-zA-Z] { [a-zA-Z0-9] }
number ::= { [0-9] }
string ::= Java-style string literal

call ::= id "(" expr { "," expr } ")"

block ::= "{ " expr "}"

comment ::= "--" any text

```

Figure 1: Syntax of Orc 0.5

4 Grammar

The syntax of Orc is defined by the extended BNF grammar in Figure 1. Keywords and symbols that appear in the program are quoted. There are three keywords: “def”, “where” and “in”. The special symbols are “(”, “)”, “=”, “,”, “;”, “>”, “!”, “|”, “{” and “}”.

Names in italics are nonterminals. The form $\{ x \}$ denotes zero or more occurrences of x , while $[x]$ denotes zero or one occurrences of x .

There are several difference from the syntax of Orc in theoretical papers:

- Definitions can be given at any level, and can be nested following block structure.
- Calls can be nested inside other calls.
- There is one namespace for variables and site names. Sites can be passed and returned as values (although this is not been tested).

NOTE: These differences may be corrected as the implementation is improved.

5 Sites

The following sites can be called:

Tuples

Let: `let(e_1, e_2, \dots, e_n)`

Produces a tuple of values. Let is strict, in that it waits for all the arguments to be defined.

Item: `item(x, n)`, `item(s, n, m)`

Access the n th item of a tuple or string. Indexes are zero-based. For strings, `item` can take an extra argument to select a substring of characters from n upto m . Thus

```
let(1, 2) >x> item(x, 0)
produces 1 and
item("foo", 1, 2)
produces "o".
```

Arithmetic

Binary Operators: `add`, `sub`, `mul`, `div`

For example, `add(3, sub(9, 2))`.

Binary Comparisons: `lt`, `le`, `eq`, `ne`, `ge`, `gt`

For example, `lt(3, 9)` is true.

Logical

Constants: `true`, `false`

Unary: `not`

Binary: `and`, `or`

Control: `if(b)`

If outputs a value when b is true. It does not produce a value when b is false.

Output: `print(e1, e2, ..., en)`, `println(e1, e2, ..., en)`

Prints arguments to standard output.

Mail: `SendMail(from, to, subject, content, server)`

Sends a mail message via an SMTP server. Does not currently support servers that require login. The `to` field can be a list created by `let`. For example:

```
SendMail("wcook@cs.utexas.edu",
        let("misra@cs.utexas.edu", "wcook@cs.utexas.edu"),
        "this is a test",
        "I think it is working now",
        "mail.cs.utexas.edu")
```

Time: `Rtimer(n)`

Wait n milliseconds, then return 1.

Miscellaneous: `random(n)`

Returns a random number between 0 and n .

5.1 Nested Calls

Site calls can be nested. Thus the following is legal:

```
if(lt(x, 10)) >> ...
```

The meaning of a nested site call is:

```
M(A(), B())
```

is interpreted as

```
M(a, b) where a = A(); b = B()
```

Note: currently all calls can be nested, not just pure functional calls.

6 Planned Enhancements

Some planned enhancements, in rough priority order from most important to least.

- Allow "import" of a file into another file, i.e., textual inclusion.
- Allow commenting large regions using, say, - -
- Allow infix arithmetic and logical expressions as arguments of site calls. These should only allow side-effect free, deterministic, immediate site calls. An easy solution is to convert the expression to prefix form since you already support it.
- Receive email responses.
- Interface to web services: Plan to use Apache Axis2 to support asynchronous calls.
- Access to java objects. Add use of dot notation for records/objects, so that `x.m(args)` will call the `m` method of `x`.
- Ability to define local sites and use them. This and a few other things can be defined as short-hand forms of existing Orc behavior.
- Misc sites: Clock, Cell, Word, Persistent storage, Access to files, Input, Output. Cell and Word are blocking and non-blocking storage cells.

References

- [1] William R. Cook and Jayadev Misra. Computation orchestration: A basis for wide-area computing. to appear in the Journal on Software and System Modeling.
- [2] William R. Cook and Jayadev Misra. Implementation outline for orc. unpublished manuscript, 2005.
- [3] William R. Cook and Jayadev Misra. A structured orchestration language. unpublished manuscript, 2005.
- [4] Jayadev Misra. Computation orchestration: A basis for wide-area computing. In Manfred Broy, editor, *Proc. of the NATO Advanced Study Institute, Engineering Theories of Software Intensive Systems*, NATO ASI Series, Marktoberdorf, Germany, 2004.