# **ORC** Language Project

## Orc is...

... a novel language for distributed and concurrent programming which provides uniform access to computational services, including distributed communication and data manipulation, through *sites*. Using four simple concurrency primitives, the programmer *orchestrates* the invocation of sites to achieve a goal, while managing timeouts, priorities, and failures.

## What can I use Orc for?

- As a **general purpose programming language** for concise encoding of concurrent and distributed applications.
- As a **Web scripting language** to create a Web-service mashup in a few minutes. Orc's emphasis on concurrency makes mashups much simpler to write than in other scripting languages.
- As an **executable specification language** for workflow applications and process coordination problems.

## Why Orc?

Orc is designed to solve a computational pattern inherent in many wide-area applications: acquire data from one or more remote services, perform some calculation with those data, and invoke yet other remote services with the results. We call such services *sites* and the integration of sites *orchestration*. Orchestration demands an understanding of the kinds of computations that can be performed efficiently over a wide-area network, where the delays associated with communication, unreliability of servers, and competition for resources from multiple clients are dominant concerns.

The theory behind Orc is that smooth orchestration requires only four simple combinators: parallel computation, sequencing, selective pruning, and termination detection. Together, these combinators prove powerful enough to express typical distributed communication patterns.

Consider a typical wide-area computing problem. A client contacts two airlines simultaneously for price quotes. He buys a ticket from either airline if its quoted price is no more than $300, the cheapest ticket if both quotes are above $300, and any ticket if the other airline does not provide a timely quote. The client should receive an indication if neither airline provides a timely quote. Such problems are typically programmed using elaborate manipulations of low-level threads. We regard this as an orchestration problem in which each airline is a site; we can express such orchestrations very succinctly in Orc.

To see how this is achieved and learn more about Orc's syntax and semantics, please visit our Web site at http://orc.csres.utexas.edu/

**Faculty Members:** Prof. Jayadev Misra
misra@cs.utexas.edu

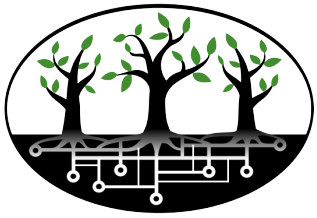
Prof. William R. Cook
wcook@cs.utexas.edu

**Graduate Researchers:** David Kitchin
dkitchin@cs.utexas.edu

John Thywissen
jthywiss@cs.utexas.edu

Amin Shali
amshali@cs.utexas.edu

Brian McCord
bmccord@cs.utexas.edu



THE UNIVERSITY OF
TEXAS
AT AUSTIN

## Dig in!

**Orc Web site:** orc.csres.utexas.edu

**In-browser demo:**
orc.csres.utexas.edu/tryorc.shtml

**Download:** orc.csres.utexas.edu/download.shtml

**Documentation:** orc.csres.utexas.edu/documentation.shtml

**Mailing list:**
groups.google.com/group/orc-lang/

**Wiki:** orc.csres.utexas.edu/wiki/

**Google Code project:**
orc.googlecode.com

# ORC SYNTAX REFERENCE

| | | |
|---|---|---|
| C ::= | boolean \| number \| string | *Constant* |
| | `signal` | |
| | `null` | |
| X ::= | identifier | *Variable* |

| | |
|---|---|
| E ::= | *Expression* |
| C | *constant value* |
| X | *variable* |
| `(` E `,` … `,` E `)` | *tuple* |
| `[` E `,` … `,` E `]` | *list* |
| `{.` key = E `,` … `,` key = E `.}` | *record* |
| `stop` | *silent expression* |
| E`(` E `,` … `,` E `)` | *call* |
| E`.`key | *member access* |
| E op \| op E \| E op E | *operator* |
| E `>P>` E | *sequential combinator* |
| E `\|` E | *parallel combinator* |
| E `<P<` E | *pruning combinator* |
| E `;` E | *otherwise combinator* |
| `lambda (` P `,` … `,` P `)` = E | *closure* |
| `if` E `then` E `else` E | *conditional* |
| D E | *scoped declaration* |

| | |
|---|---|
| D ::= | *Declaration* |
| `val` P = E | *value declaration* |
| `def` X`(` P `,` … `,` P `)` = E | *function definition* |
| `def class` X`(` P `,` … `,` P `)` = E | *class definition* |
| `import site` X = address | *site import* |
| `import class` X = classname | *class import* |
| `include` "filename" | *inclusion* |

| | |
|---|---|
| P ::= | *Pattern* |
| X | *variable* |
| C | *constant* |
| `_` | *wildcard* |
| X `(` P `,` … `,` P `)` | *call pattern* |
| `(` P `,` … `,` P `)` | *tuple pattern* |
| `[` P `,` … `,` P `]` | *list pattern* |
| `{.` key = P `,` … `,` key = P `.}` | *record pattern* |
| P `:` P | *cons pattern* |
| P `as` X | *bind sub-pattern to* X |

Where relevant, syntactic constructs are ordered by precedence, from highest to lowest. For example, among expressions, calls have higher precedence than any of the combinators, which in turn have higher precedence than conditionals.

Predefined operators:
`:= || && < > = /= :> >=`
`<: <= ~ : + - / % * **`

These are the four Orc "combinators" that give the language its ability to structure concurrent operations and combine the flow of results from these operations. The combinators are the central concept of Orc.

Imported sites and classes enable access to remote or local resources (such as Web services or Java class files) as a "site call" in Orc expressions.

The Orc prelude (standard library) defines about 50 sites, which are automatically available in Orc programs.

Comments are enclosed in `{-` and `-}` or run from `--` to the end of the line.