

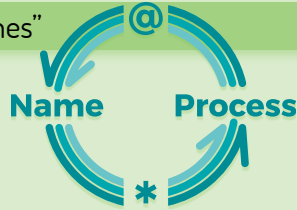


Rholang Cheat Sheet

Sends and Receives

<code>x!(P)</code>	Send process P on name x
<code>x!!(P)</code>	Persistent send
<code>for (y <- chan){P}</code>	Receive name y on chan
<code>for (@Q <- chan){P}</code>	Receive Process Q (see pattern matching)
<code>for(y <= chan){P}</code>	Persistent receive
<code>contract chan(y) = {P}</code>	Alternate persistent receive
<code>for(y <! chan){P}</code>	Peek at y on chan

"Send processes, receive names"



Unforgeable Names

<code>new x, y, z in { P }</code>	binds x, y, z in P
<code>new print(rho:io:stdout\)</code>	use system powerbox

Arithmetic

+ addition - subtraction / division * multiplication % hopefully mod coming soon

Patterns

A free variable

- `x` binds with anything, while `@x` matches to a name and binds `x` to the quoted process.

`Bool Int String Uri ByteArray` Type patterns

- `@{Bool}` matches to both `@true` and `@false`

`[Head ... Tail] Set[Subset ... Tail] { Key : Value ... Tail }`

- `[1, 2 ... x]` matches any list starting with 1, 2 and binds `x` to the remainder

`ProcessPattern /\ ProcessPattern` Logical AND

- `@{x /\ 100}` matches to `@100` and binds `x` to 100

`ProcessPattern \/ ProcessPattern` Logical OR

- `@"age"!(21 \/ 22)` matches to both `@"age"!(21)` and `@"age"!(22)`, binds nothing

`- ProcessPattern` Logical NOT

- `- Nil` matches to any process except Nil

Pattern Matching

The patterns in:

```
for{ Pattern <- Name }{ Body }
for{ Pattern <= Name }{ Body }
contract Name(Pattern){ Body }
```

Match against the processes in:

```
Name!(Process)
Name!!(Process)
```

Each `Pattern_i` in:

```
for{ Pattern_1 <- Name_1 ; ... ; Pattern_N <- Name_N }{ Body }
for{ Pattern_1 <= Name_1 ; ... ; Pattern_N <= Name_N }{ Body }
```

Matches against a `Process_i` in:

```
Name_1!(Process_1) | ... | Name_N!(Process_N)
Name_1!!(Process_1) | ... | Name_N!!(Process_N)
```

Tries to match **Process** against each **Pattern_i** until it finds a match (or doesn't):

```
match Process { Pattern_1 => { Body_1 } ... Pattern_N => { Body_N } }
```

Bundles

Cannot be destructured by pattern matching

	Can Read	Can Write
<code>bundle- {proc}</code>	YES	NO
<code>bundle+ {proc}</code>	NO	YES
<code>bundle {proc}</code>	NO	NO
<code>bundle {proc}</code>	YES	YES



RChain COOPERATIVE

For more information
<https://www.rchain.coop/learn-rholang>

Conditionals

```
if (x) { P } run process P iff x is true
else { Q } (optional) run process Q iff x is false
```

Data Structures

Strings

<code>"Hello " ++ "World"</code>	concatenation
<code>"\${greeting} World" %%" {"greeting": "Hello"}</code>	interpolation
<code>"Hello World".slice(2, 8)</code>	"llo Wo"
<code>"A402B6".hexToBytes()</code>	interpret hex string

Lists

<code>[1, 2, Nil, "Hi"]</code>	Output
<code>list.nth(2)</code>	Nil
<code>list.length()</code>	4
<code>list.slice(1, 3)</code>	[2, Nil]

Tuples

<code>(1, 2, Nil, "Hi")</code>	Output
<code>tuple.nth(2)</code>	Nil

Sets

<code>Set(1, 2, Nil, "Hi")</code>	Output
<code>set.union(Set(1, 4))</code>	Set(1, 2, 4, Nil, "Hi")
<code>set.delete(2)</code>	Set(1, Nil, "Hi")
<code>set.contains(5)</code>	false
<code>set.size()</code>	4

*Sets have no order or duplicates

Maps

<code>{"a": 1, "b": 2}</code>	Output
<code>map.union({"c": 3})</code>	{"a": 1, "b": 2, "c": 3}
<code>map.delete("b")</code>	{"a": 1}
<code>map.contains("c")</code>	false
<code>map.get("b")</code>	2
<code>map.getOrElse("d", "fail")</code>	fail
<code>map.set("b", 4)</code>	{"a": 1, "b": 4}
<code>map.keys()</code>	Set("a", "b")
<code>map.size()</code>	2

*All data structures have `toByteArray()`