# Concurrency in Jason



## Introduction

This page is for people who understand the idea of programming with threads (i.e., concurrently) who would like to discuss issues of concurrency of intentions in ***Jason***.

Intentions in Jason are somewhat similar to threads in other languages in so far as they also execute concurrently, unless you have customised the intention selection function to alter the schedulling. By default, different intentions are executed in a round-robin fashion, executing only one element of the (topmost) plan body each time the intention is selected.

New intentions are created by:

- external events
  - each perceived change in the environment that the agent reacts to starts a new intention
  - another agent delegates a goal (through an *achieve* message)
- initial goals
- the !! operator
- depending on the interpreter setting, belief addtions other than by perception of the environment can also cause a new intention to be created

(tbd add and explain examples)

## Atomic

An atomic intention cannot be stopped in the middle. An intention is atomic if it has a plan with an *atomic* annotation.

(tbd add and explain examples)

## Suspend and Resume

Programmers can suspend and resume intentions by means of specific internal actions (`.suspend`, `.resume`); an intention can suspend/resume other intentions.

Intentions are automatically put in suspended mode whilst it is waiting for some action to be done in the environment; it resumes when the action feedback from the environment is received.

Intentions are automatically put in suspended mode when an `askOne` or `askAll` message is sent; it resumes when the agent receives the answer (or timeout).

(tbd add and explain examples)

## Wait

The internal action .wait can also be used to suspend an intention for some time:

```
+!g
   <- ...;
      .wait(1000); // suspend the intention for 1 second
      ...
```

and to suspend it until some event:

```
+!g
   <- ...;
      .wait("+b(4)"); // suspend the intention until the belief-addition event +b(4) takes place
      ...
```

see more details (e.g., elapsed time and timeout) at wait API.

(tbd add and explain examples)

## Proposals to be Discussed and Implemented

### Monitors

```
.wait(a(X))
.notify(a(10))
```

### Concurrent goals

Creation of a new plan-body operator that allows concurrent execution:

```
a1; ( !g1 || !g2; a2 || !g3); a3
```

do action `a1`; concurrently execute `!g1`, `(!g2; a2)`, and `!g3`, where `(g2;a2)` means achieve goal `g2` then in sequence do action `a2`; when all three separate subplans are finished, only then do action `a3`.
Note: the implementation requires that intentions be modelled as trees and no longer as stacks.

### Semaphore

Proposal for an internal action:

```
.synchronize( s ) { bla }
```

or .lock
(see how locks are implemented in Java 5)

## Open questions

1. are intentions really like threads? or is there something peculiar about them? (perhaps we should try to come up with a higher level mechanism)
2. how to relate/manage intentions at a higher level: hinder, resume (some intentions always suspend others), cancel (some intentions abort/fail others), incompatible-with (two intentions cannot be executed together) -- perhaps a simplification of TAEMS relationships?