amb egg

The non-deterministic backtracking ambivalence operator
Extension for Chicken Scheme
Version 1.2.0

Thomas Chust

Table of Contents

1 About this egg

1.1 Version history

`1.2.0`	Non-determinism now optional, better controllability of scope
`1.1.1`	Fixed bug in amb.setup [Thanks to Kon Lovett]
`1.1.0`	Renamed `bag-of` to `amb-collect` and added `amb-assert`
`1.0.0`	Initial release

1.2 Usage

Load this egg like so:

```
(require-extension amb)
```

2 Documentation

The `amb` operator is a nice toy and sometimes a useful tool for lightweight logic programming. Its implementation is also a good exercise in the handling of continuations.

2.1 Normal interface

`amb` [macro]

```
(amb expr ...) => <top>
```

In the form `(amb)` the expression always fails.

If the expression has any parameters, the first one of them is evaluated and the result is returned. If a subsequent occurrence of `amb` fails, though, backtracking may cause the second of the given expressions to be selected for evaluation, then the third and so forth until the whole program does not fail if at all possible.

The backtracking mechanism is described along with the `amb-failure-continuation` parameter below.

`amb/random` [macro]

```
(amb/random expr ...) => <top>
```

Works like `amb` but the parameters are not selected in sequence but randomly. None of them is selected more than once, though.

`amb-find` [macro]

```
(amb-find expr #!optional failure-value) => <top>
```

Evaluates `expr` returning its value if successful (possibly after backtracking).

If `expr` cannot be evaluated successfully and the expression tree is exhausted, `failure-value` is evaluated and the result is returned instead. If no `failure-value` is specified, an exception occurs. See the `amb-failure-continuation` parameter below for a description of the exception.

`amb-collect` [macro]

```
(amb-collect expr) => <list>
```

Evaluates `expr` and performs backtracking repeatedly until all possible values for it have been accumulated in a list, which is returned.

`amb-assert` [macro]

```
(amb-assert ok?) => <void>
```

Evaluates `ok?` and fails if it is `#f`.

2.2 Additional stuff exported by amb-base

`amb-failure-continuation` [parameter]

Seen in a global context, the `amb` operator transforms the whole program that contains it into a depth first search for return values from `amb` forms that will not cause failure.

This is realized using a backtracking system that invokes previously stored continuations whenever an `amb` expression fails. The `amb-failure-continuation` parameter is the status variable for this system.

At the start of the program, or when no further backtracking options are available, this is set to a procedure of no arguments that throws an exception of the composite `(exn amb)` kind (except when a `amb-collect` statement is being processed, where the parameter will point to a procedure signalling `amb-collect` that there are no more backtracking options available).

In all other cases this parameter is set to a procedure of no arguments that causes backtracking to the next possible alternative in the "tree".

If you want to restrict the scope of backtracking to something smaller than the whole past program, use `amb-find` or `amb-collect` which restore this parameter to its original value when they are done evaluating the expressions they were given.

`amb-thunks` [procedure]

```
(amb-thunks <list of procedures>) => <top>
```

The backend of `amb`. `amb` wraps all its parameters into thunks and passes a list of them into this procedure, `amb/random` shuffles the list first.

`amb-find-thunk` [procedure]

```
(amb-find-thunk <procedure> #!optional <procedure>) => <top>
```

The backend of `amb-find`. `amb-find` wraps its parameters into thunks and passes them into this procedure.

`amb-collect-thunk` [procedure]

```
(amb-collect-thunk <procedure>) => <list>
```

The backend of `amb-collect`. `amb-collect` wraps its parameter into a thunk and passes it into this procedure.

3 Examples

The following code is a rewrite of an example from the book "Teach Yourself Scheme in Fixnum Days" by Dorai Sitaram. The book gives the following problem setting:

> The Kalotans are a tribe with a peculiar quirk. Their males always tell the truth. Their females never make two consecutive true statements, or two consecutive untrue statements.
>
> An anthropologist (let's call him Worf) has begun to study them. Worf does not yet know the Kalotan language. One day, he meets a Kalotan (heterosexual) couple and their child Kibi. Worf asks Kibi: "Are you a boy?" Kibi answers in Kalotan, which of course Worf doesn't understand.
>
> Worf turns to the parents (who know English) for explanation. One of them says: "Kibi said: 'I am a boy.'" The other adds: "Kibi is a girl. Kibi lied."
>
> Solve for the sex of the parents and Kibi.

So here is the solution:

```
;;;; amb-demo.scm
;;;; A solution for the Kalotan puzzle using amb

(require-extension amb)

(define (xor a? b?)
  (if (and a? b?) #f (or a? b?)))

(define (solve-kalotan-puzzle)
  (let ((parent1 (amb 'm 'f))
        (parent2 (amb 'm 'f))
        (kibi (amb 'm 'f))
        (kibi-self-desc (amb 'm 'f))
        (kibi-lied? (amb #t #f)))
    (amb-assert
     (not (eq? parent1 parent2)))
    (if kibi-lied?
        (amb-assert
         (xor
          (and (eqv? kibi-self-desc 'm)
               (eqv? kibi 'f))
          (and (eqv? kibi-self-desc 'f)
               (eqv? kibi 'm)))))
    (if (not kibi-lied?)
        (amb-assert
         (xor
          (and (eqv? kibi-self-desc 'm)
               (eqv? kibi 'm))
          (and (eqv? kibi-self-desc 'f)
               (eqv? kibi 'f)))))
    (if (eqv? parent1 'm)
        (amb-assert
         (and
          (eqv? kibi-self-desc 'm)
          (xor
           (and (eqv? kibi 'f)
                (eqv? kibi-lied? #f))
           (and (eqv? kibi 'm)
                (eqv? kibi-lied? #t))))))
    (if (eqv? parent1 'f)
        (amb-assert
         (and
          (eqv? kibi 'f)
          (eqv? kibi-lied? #t))))
    (list parent1 parent2 kibi)))

(write (amb-collect (solve-kalotan-puzzle)))
(newline)
```

4 License

Index