Algebraic Effects

Gli Algebraic Effects si basano su un concetto fondamentale: l'istruzione `throw(E)` trasferisce il controllo a distanza alla clausola `catch` che gestisce l'effetto `E`. Il codice remoto può successivamente restituire il controllo alla `throw` fornendo un valore `V` che diventa il risultato dell'istruzione `throw` stessa.

Un esempio in pseudo codice Erlang potrebbe essere:

```
2 * try 3 + (5 + throw 7)
    catch
        0 -> 22                     % come un'eccezione normale
        N -> resume N+1             % resume nuova keyword
    end
```

Nel caso dell'utilizzo di resume, il risultato sarebbe `32`.

Come viene implementato questo meccanismo di "ritorno"?
Nelle eccezioni tradizionali, la `throw` esegue un ciclo `while` sullo stack e per ogni record di attivazione/record try catch che non gestisce l'eccezione effettua `Stack.pop()`.

Negli algebraic effects (nell'implementazione più semplice) la `throw`, invece di eseguire un semplice `Stack.pop()`, effettua:

```
RA = Stack.pop();
Detached.push(RA);
```

dove Detached rappresenta un secondo stack, ordinato in senso inverso, nel quale vengono temporaneamente memorizzati i frame di stack distaccati.

L'istruzione `resume` esegue la seguente operazione:

```
while(!Detach.is_empty()) {
    RA = Detach.pop();
    Stack.push(RA);
}
```

assegnando il risultato della `resume` come valore di ritorno della `throw`.

Se invece nel ramo `catch` non viene utilizzata l'istruzione `resume`:

```
while(!Detach.is_empty())
    Detach.pop()
```

In un linguaggio con supporto per gli effetti algebrici viene introdotto un nuovo tipo di dato astratto denominato **fibra** (**fiber**) che rappresenta un frammento di stack distaccato. Questo è un tipo di dato di prima classe sul quale è possibile invocare l'operazione `resume()` per reinstallarlo in cima allo stack corrente. Il ramo `catch` cattura questa fibra.

Esaminiamo ora un esempio di implementazione di uno scheduler, in pseudo codice Erlang:

```
yield() -> throw(yield).            % yield è un effetto
fork(G) -> throw(fork).             % fork è un effetto

code_to_fiber(F) ->
 try
  throw(stop),F
 catch
  stop, K -> K
 end.

% Main è il primo thead da eseguire
% Queue la coda dei thread sospesi
scheduler(Main, Queue) ->
 try
  resume(Main, ok)
 catch
  yield, K ->                       % K è la fibra, i pattern sono sempre
   case Queue of                    % pattern_eccezione + pattern K
    [] -> scheduler(K, [])
    [F|L] -> scheduler(F, append(L,[K]))
   end ;
  fork(G), K ->
    scheduler(K, append(Queue,[code_to_fiber(G)]))
 end.

scheduler(Main) ->
 scheduler(code_to_fiber(Main),[]).
```

Gli effetti algebrici consentono quindi di gestire in maniera non locale gli errori come se fossero gestiti localmente, di implementare scheduler a livello utente (user-space), e altri meccanismi avanzati.

Un'implementazione efficiente degli effetti algebrici trasforma lo stack in uno stack di fibre, dove ogni fibra è a sua volta uno stack.
In questo modo, le operazioni di distacco e riattacco di una fibra hanno un costo computazionale di $\mathbf{O(1)}$, sebbene ciò comporti che lo stack non sia più allocato in modo contiguo in memoria.