5 Eccezioni

Nei programmi le funzioni hanno a volte bisogno di segnalare al loro chiamante che non sono in grado di portare a termine il loro compito perché è successo un errore o si è presentato un caso eccezionale (vedi la differenza in Java tra errori ed eccezioni). In questo caso è quindi utile avere un meccanismo per gestire le eccezioni: quando l'esecuzione del programma solleva un'eccezione, *il controllo viene trasmesso direttamente al gestore* dell'eccezione, o se tale gestore manca, si fa abortire l'esecuzione. Nei linguaggi reali quello che succede è che il sollevamento di un'eccezione provoca la chiusura degli activation records relativi alle chiamate di funzione attive, andando all'indietro nel call stack fino al punto in cui si trova (se c'è) un exception handler. In altre parole, le eccezioni provocano un trasferimento non-locale del controllo, fino all'exception handler più vicino nel call stack. Arricchiamo il linguaggio con un meccanismo di sollevamento/gestione delle eccezioni analogo a quello dei linguaggi mainstream.

Sintassi Aggiungiamo un termine $\mathsf{throw}\, M$ che indica il sollevamento di un'eccezione che porta con sé dell'informazione aggiuntiva, rappresentata dal termine M, che può essere usata dall'exception handler per gestire il caso eccezionale. Lasciamo per ora generico il termine trasportato dall'eccezione, e diciamo che ha un tipo particolare T_{exn}. Il secondo termine da aggiungere è della forma $\mathsf{try}\, M\, \mathsf{catch}\, N$, il cui significato consiste nel valutare il termine M e, nel caso questa valutazione sollevi un'eccezione con valore v, sarà valutato N a cui verrà passato v come argomento. Il termine N sarà quindi il gestore dell'eccezione, e sarà una funzione che prende come parametro il valore trasportato dall'eccezione stessa.

Termini M	$::=$	$x \mid c$	variabili e costanti
	$\mid$	$\mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\, M\, \mathsf{then}\, M\, \mathsf{else}\, M$	operazioni intere e condizionale
	$\mid$	$\mathsf{fn}\, x.M \mid MM$	funzioni
	$\mid$	$\mathsf{throw}\, M$	solleva un'eccezione
	$\mid$	$\mathsf{try}\, M\, \mathsf{catch}\, M$	gestisce un'eccezione

Ad esempio si consideri i termini:

- $M = \mathsf{fn}\, x.(\mathsf{if}\, pari(x)\, \mathsf{then}\, x/2\, \mathsf{else}\, \mathsf{throw}\, x)$
- $\mathsf{try}\, (M\ \ 3)\, \mathsf{catch}\, \mathsf{fn}\, y.y + y$
- $\mathsf{try}\ \ (\mathsf{fn}\, y.y - 2\ \ (M\ \ 5))\, \mathsf{catch}\, \mathsf{fn}\, z.print(z)$
- $\mathsf{try}\ \ (\mathsf{fn}\, y.y - 2\ \ (M\ \ \mathsf{throw}\, 2))\, \mathsf{catch}\, \mathsf{fn}\, z.print(z)$
- $\mathsf{try}\, (\mathsf{fn}\, y.y - 2\ \ (\mathsf{try}\, (M\ \ 5)\, \mathsf{catch}\, \mathsf{fn}\, z.z{+}z))\, \mathsf{catch}\, \mathsf{fn}\, z.print(z)$
- $\mathsf{try}\, (\mathsf{fn}\, x.x\ \ \ \mathsf{throw}\, 1)\ (\mathsf{try}\, (M\ \ 5)\, \mathsf{catch}\, \mathsf{fn}\, z.z{+}z)\, \mathsf{catch}\, \mathsf{fn}\, z.print(z)$

Semanica operazionale Le regole di semantica operazionale che si aggiungono descrivono il sollevamento delle eccezioni e il comportamento dell'exception handler. In un'espressione $\mathsf{try}\, M\, \mathsf{catch}\, N$ viene valutato il termine M (regola (TRY)), se la valutazione finisce correttamente in un valore, allora l'handler viene scartato (regola (TYR VAL)), se invece la valutazione di M produce il sollevamento di un'eccezione, la regola (TRY HANDLE) attiva l'exception handler passando come parametro il valore trasportato dall'eccezione. Nota che la regola (TRY HANDLE) richiede che prima di attivare l'handler, il termine trasportato dall'eccezione debba essere completamente valutato ad un valore, cosa possibile grazie alle regole (RAISE 1) e (RAISE 2). La seconda regola si applica nel caso una (seconda) eccezione sia sollevata mentre si sta valutando l'informazione propagata da un'eccezione sollevata in precedenza. Infine le regole (RAISE APP 1/2) propagano l'eccezione attraverso le applicazioni di funzioni fino ad incontrare l'handler. Le regole indicano infatti che se si solleva un'eccezione mentre si sta valutando una chiamata di funzione (la funzione o il suo argomento), tale valutazione va subito interrotta e va restituita l'eccezione

sollevata. Analogamente, le regole (RAISE IFELSE) e (RAISE SUM) propagano le eccezioni nei rimantenti contesti del liguaggio.

(TRY)
$$\frac{M \longrightarrow M'}{\mathsf{try}\ M\ \mathsf{catch}\ N \longrightarrow \mathsf{try}\ M'\ \mathsf{catch}\ N}$$

(TRY HANDLE)
$$\frac{}{\mathsf{try}\ \mathsf{throw}\ v\ \mathsf{catch}\ M \longrightarrow M\ v}$$

(TRY VAL)
$$\frac{}{\mathsf{try}\ v\ \mathsf{catch}\ M \longrightarrow v}$$

(RAISE 1)
$$\frac{M \longrightarrow M'}{\mathsf{throw}\ M \longrightarrow \mathsf{throw}\ M'}$$

(RAISE 2)
$$\frac{}{\mathsf{throw}\ (\mathsf{throw}\ v) \longrightarrow \mathsf{throw}\ v}$$

(RAISE APP 1)
$$\frac{}{(\mathsf{throw}\ v)\ M \longrightarrow \mathsf{throw}\ v}$$

(RAISE APP 2)
$$\frac{}{v_1\ (\mathsf{throw}\ v_2) \longrightarrow \mathsf{throw}\ v_2}$$

(RAISE IFELSE)
$$\frac{}{\mathsf{if}\ \mathsf{throw}\ v\ \mathsf{then}\ M\ \mathsf{else}\ N \longrightarrow \mathsf{throw}\ v}$$

(RAISE SUM 1)
$$\frac{}{\mathsf{throw}\ v + M \longrightarrow \mathsf{throw}\ v}$$

(RAISE SUM 2)
$$\frac{}{v_1 + \mathsf{throw}\ v_2 \longrightarrow \mathsf{throw}\ v_2}$$

(RAISE MINUS 1)
$$\frac{}{\mathsf{throw}\ v - M \longrightarrow \mathsf{throw}\ v}$$

(RAISE MINUS 2)
$$\frac{}{v_1 - \mathsf{throw}\ v_2 \longrightarrow \mathsf{throw}\ v_2}$$

EXERCISE 5.1. Si dia la semantica operazionale dei termini indicati sopra, osservando come il sollevamento delle eccezioni comporti un salto non locale del flusso di controllo. □

EXERCISE 5.2. Si definisca la semantica operazionale per il liguaggio esteso con i costrutti visti in precedenza: unit, records e varianti. □

Si osservi che il termine $\mathsf{throw}\ v$ non è stato aggiunto alla sintassi dei valori, anche se si tratta di una forma normale, che cioè non riduce oltre. Ciò è voluto e serve per rendere deterministica la valutazione di termini come $(\mathsf{fn}\ x{:}\mathsf{Nat}.0)\ \mathsf{throw}\ v$, che altrimenti potrebbe, nondeterministicamente, ridurre a 0 (applicando l'assioma dell'applicazione di funzioni) oppure a $\mathsf{throw}\ v$ (applicando la regola (RAISE APP 2). Con le regole che abbiamo date non c'è invece ambiguità: poichè il termine $\mathsf{throw}\ v$ non è un valore, l'unica regola applicabile è (RAISE APP 2).

Typing Si osservi che, poiché il lingugaggio ammette un modo per gestire le eccezioni sollevate, è giusto trovare un modo per rendere ben tipato anche un termine la cui esecuzione solleva un'eccezione. Il teorema di safety dovrà poi essere invece riformulato per tenere conto anche delle eccezioni sollevate e non gestite. Che tipo dunque assegnare al termine $\mathsf{throw}\ M$? Poiché vogliamo poter sollevare un'eccezione in qualsiasi contesto, il termine $\mathsf{throw}\ M$ deve poter essere tipato con ogni tipo, indipendentemente dal tipo di M. Ad esempio, si considerino questi termini:

$$M_1 = (\mathsf{fn}\ x{:}\mathsf{Bool}.x)\ \mathsf{throw}\ 0 \qquad M_2 = \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ 2\ \mathsf{else}\ \mathsf{throw}\ 0$$
$$M_3 = \mathsf{try}\ ((\mathsf{fn}\ x{:}\mathsf{Bool}.\mathsf{throw}\ 0)\ \mathsf{false}) + 5\ \ \mathsf{catch}\ \mathsf{fn}\ y.y \qquad M_4 = \mathsf{try}\ (\mathsf{throw}\ 0\ \ \mathsf{true})\ \mathsf{catch}\ \mathsf{fn}\ y.y$$

Affinché siano termini ben tipati, nel primo caso $\mathsf{throw}\ 0$ deve avere tipo Bool, nel secondo e nel terzo caso deve avere tipo Nat mentre nell'ultimo caso deve avere tipo $\mathsf{Bool} \to T$. Dunque, poiché il termine $\mathsf{throw}\ M$ può apparire in qualsiasi contesto, dobbiamo essere in grado di dargli un tipo qualsiasi. La cosa più semplice è appunto una regola che assegna qualsiasi tipo T, anche se ciò fa perdere la proprietà per cui ogni termine del linguaggio ha un tipo unico.

Si noti che in Java l'espressione `throw new Exception()` non ha un tipo preciso. Si consideri ad esempio il metodo

```
C foo ()
{ if(...) return new C();
  else throw new Exception();
}
```

Il typing del costrutto if/then/else richiederebbe anche per `throw new Exception()` il tipo C. In Java invece le eccezioni sono trattate in modo speciale dal sistema di tipi. Per le eccezioni controllate, i.e., le sottoclassi di `Exception`, il type system controlla che le eccezioni sollevate da un metodo appaiano nel prototipo del metodo, i.e. `C foo() throws Exception`. Per le eccezioni non controllate, la specifica del linguaggio Java dice "non-checked exceptions are exempted from compile-time checking".

$$\text{(RAISE)}\ \frac{\Gamma \vdash M : T_{exn}}{\Gamma \vdash \mathsf{throw}\, M : T} \qquad \text{(TRY)}\ \frac{\Gamma \vdash M : T \quad \Gamma \vdash N : T_{exn} \to T}{\Gamma \vdash \mathsf{try}\, M\, \mathsf{catch}\, N : T}$$

Per quanto riguarda la scelta del tipo T_{exn}, due semplici possibilità sono $T_{exn} = \mathsf{Nat}$ come in C, oppure, come in ML, T_{exn} potrebbe rappresentare un *(extensible) variant type* ad esempio:

```
<divideByZero:Unit,  overflow:Unit, fileNotFound:String, ... >
```

oppure, come in Java, $T_{exn} =$`Throwable`, cioè il tipo di una "superclasse" comune a tutte le eccezioni definibili dall'utente.

Per quanto riguarda le due principali proprietà del sistema di tipi, il Teorema di Preservazione resta invariato proprio grazie alla regola di tipo (RAISE). Il Teorema di Progressione, e dunque anche il Teorema di Safety, vanno invece adattati visto che abbiamo un termine ben tipato che non è un valore, e che non si valuta oltre.

Theorem 5.3 (Progressione)**.** *Sia M un termine chiuso e ben tipato, i.e. $\emptyset \vdash M : T$, allora o M è un valore, oppure $\exists M'.\ M \longrightarrow M'$ oppure $M = \mathsf{throw}\, v$ per qualche valore v.* □

Theorem 5.4 (Safety)**.** *Sia M un termine chiuso e ben tipato, allora M non evolve ad un termine stuck, cioè sia $\emptyset \vdash M : T$ e $M \longrightarrow^* M'$ con $M' \not\longrightarrow$, allora $\exists v$ tale che $M' = v$ oppure $M' = \mathsf{throw}\, v$.* □

Quindi il teorema di safety dice che un programma ben tipato può evolvere in un'eccezione non gestita. Ciò accade ad esempio anche nei programmi ben tipati di Java, che possono sollevare un'eccezione runtime.

EXERCISE 5.5. Dimostrare che quattro termini M_1, M_2, M_3, M_4 definiti sopra rispettano il teorema di safety □

EXERCISE 5.6. Dimostrare il seguente enunciato oppure trovarne un controesempio: Sia $\mathsf{try}\, M\, \mathsf{catch}\, N$ un termine chiuso e ben tipato, allora sicuramente evolve ad un valore finale v. □