Progetto di Ingegneria del Software

Espressioni Condizionali

Matteo Croce 82246

1. Espressioni Condizionali

Si considera un semplice linguaggio per la formulazione di espressioni condizionali, ossia espressioni booleane i cui operandi sono generati da un confronto di due espressioni aritmetiche, con un operatore di relazione. Le espressioni aritmetiche si intendono costruite su numeri senza segno, manipolati dagli operatori addizione, sottrazione, moltiplicazione, divisione (quoziente), resto della divisione intera, elevamento a potenza. Nelle espressioni aritmetiche, l'elevamento a potenza è il più prioritario, seguono gli operatori moltiplicativi (moltiplicazione e divisione) ed infine quelli additivi (somma e sottrazione). Nelle espressioni booleane, NOT è l'operatore più prioritario, seguono le AND, quindi le OR. Gli operatori di relazione sono più prioritari degli operatori booleani.

Esempi di espressioni condizionali:

a<=b && c>1 || a==4 che è equivalente a: (a<=b && c>1) || a==4
a%b==0 && (a>2 || b-12>0)

L'obiettivo del progetto, basato sui pattern Composite, Builder (ricorsivo) e Visitor, è quello di acquisire da input un'espressione condizionale, farne l'analisi sintattica e costruire l'albero sintattico (composite) corrispondente. A partire dall'albero occorre valutare il risultato della condizione sulla base di un *contesto* o ambiente (fornito esternamente mediante un file properties), che stabilisce i valori delle variabili intere usate nelle espressioni aritmetiche (se una variabile non è presente nel contesto, si assume che il suo valore sia 0). Occorre altresì predisporre le operazioni di stampa in ordine simmetrico e postfisso del composite di una condizione.

Una particolarità del progetto è che l'operazione di valutazione deve applicare correttamente le condizioni di corto circuito sugli operatori AND e OR. Ad es. (corto circuito AND): b>0 && a/b>2, se b è <=0 tutta la AND deve valutarsi a false senza valutare il secondo "pezzo" che invece potrebbe dar luogo ad un'eccezione per divisione per 0. Similmente (corto circuito OR): b==0 || a/b>2 se b è 0, tutta la OR deve valutarsi a true senza valutare il esplicitamente il secondo confronto della OR. Attenzione che come in Java, le condizioni di corto circuito si basano sull'ordine di elencazione dei confronti sinistro e destro dell'operatore AND o OR.

1.1 Grammatica

<cond> → <termb> [OR <termb>]*
<termb> → <factb>[AND <factb>]*
<factb> → <expr> RELOP <expr> | NOT <factb> | OPAR <cond> CPAR
<expr> → [PLUS | MINUS] <term> [(PLUS <term>) | (MINUS <term>)]*
<term> → <termp> [(MULT <termp>) | (DIV <termp>) | (REM <termp>)]*
<termp> → <fact> [POWER <fact>]*
<fact> → ID | NUM | OPAR1 <expr> CPAR1

1.2 Simboli terminali

ID → ("A" | ... | "Z" | "a" | ...| "z") [("A"| ... | "Z" | "a" | ...| "z" | "0" | ... | "9")]*
NUM → ("0" | ... | "9") [("0" | ... | "9")]*
OPAR → "("
CPAR → ")"

OPAR1 → "["
CPAR1 → "]"
RELOP → EQ | NEQ | GT | GE | LT | LE
EQ → "= ="
NEQ → "!="
GT → ">"
GE → ">="
LT → "<"
LE → "<="
POWER → "^"
DIV → "/"
REM → "%"
MULT → "*"
MINUS → "−"
PLUS → "+"
AND → “and” o “&&”
OR → “or” o “||”
NOT → “not” o “!”

Sotto espressioni aritmetiche in parentesi utilizzano le parentesi quadre. Sotto condizioni in parentesi usano le parentesi tonde.

Analisi dei requisiti

Si deve sviluppare un sistema software per l'analisi di espressioni numeriche. L'espressione viene scomposta in sottoespressioni che possono essere di quattro tipi:

- Espressioni Relazionali (X < Y, X != Y, ...)
- Espressioni Logiche (X && Y, X || Y, ...)
- Espressioni Algebriche (X + Y, X / Y, ...)
- Numeri (1, 2, a, b, ...)

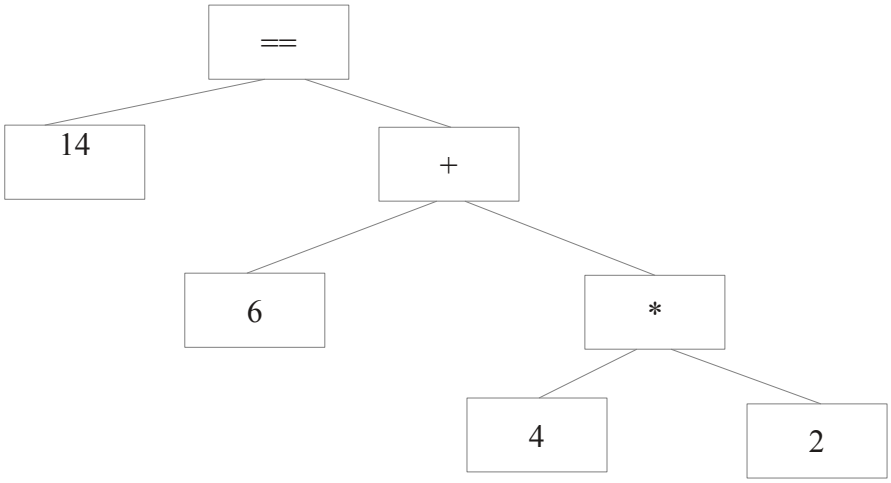
L'espressione viene rappresentata in un albero tenendo conto della precedenza degli operatori e delle parentesi, per esempio nell'espressione:

$$14 == 6 + 4 * 2$$

viene valutata prima la moltiplicazione, poi la somma e infine l'uguaglianza.

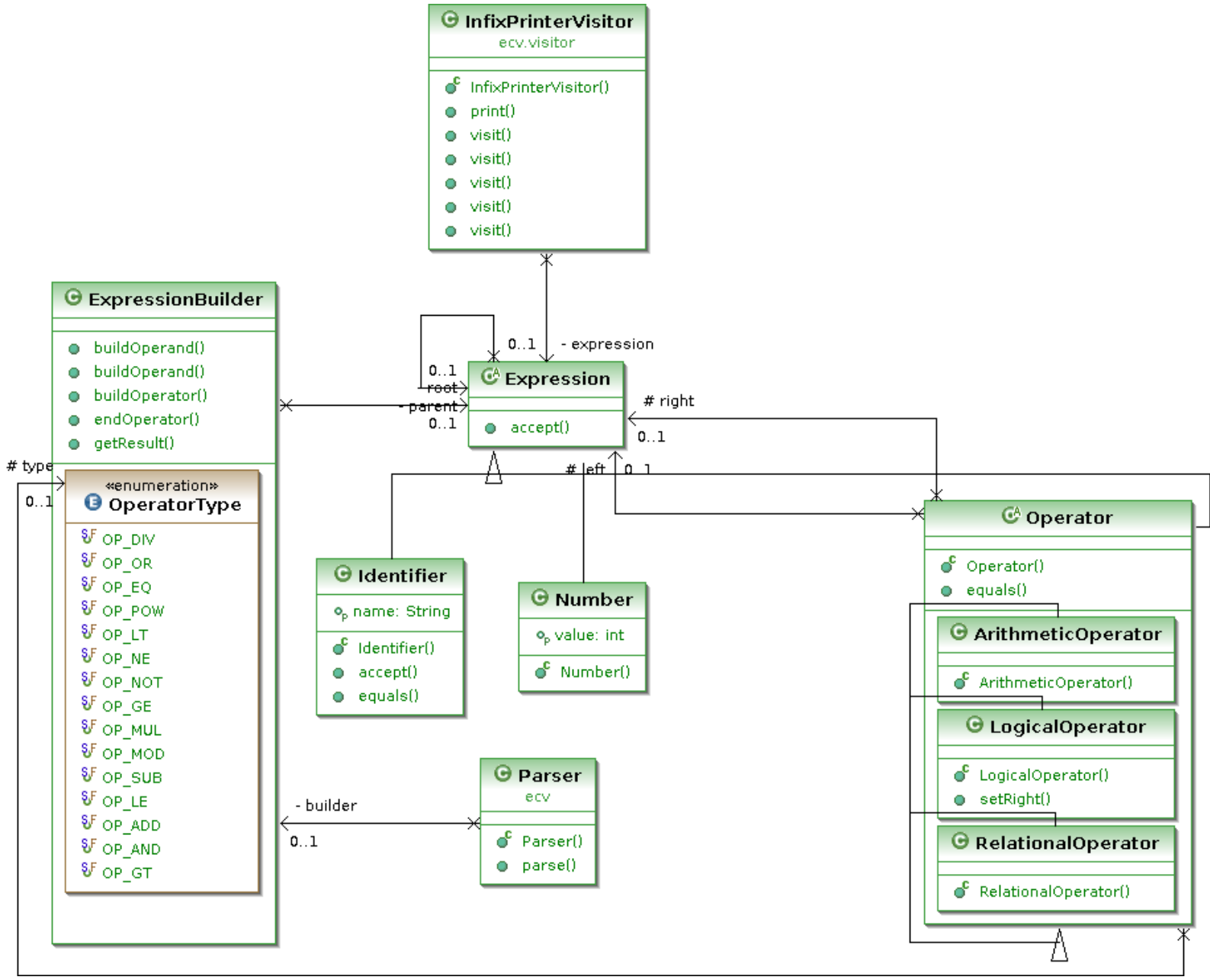
Progettazione delle classi

nel caso dell'espressione precedente viene composto un albero di questo tipo:



Dove le espressioni con minore precedenza vengono inserite in cima all'albero e vengono valutate alla fine.
Il nodo radice e` sempre un'espressione relazionale o logica, mentre i nodi foglia sono sempre numeri.
Dal momento che un'espressione aritmetica quando viene valutata da` come risultato un numero mentre un'espressione logica o relazionale da` un booleano, si e` scelto di usare il pattern Composite.
A tal punto si e` creata la classe astratta **Expression** che rappresenta una generica espressione o un numero, estesa dalle classi **Identifier Number** e **Operator** che rappresentano rispettivamente un numero, una variabile numerica, o un'espressione (aritmetica logica o relazionale).
L'espressione viene quindi analizzata e viene costruito in memoria il suddetto grafico che rappresenta l'espressione tenendo conto delle precedenze e delle parentesi.

Diagramma delle classi del Composite



A questo possiamo fare analizzare il grafo a degli oggetti diversi in grado di dare diversi tipi di analisi dell'espressione.
Per fare cio` si e` creata la classe astratta **Visitor** che rappresenta un generico “scanner” del grafo.
Viene poi subclassata per valutare l'espressione in quattro modi diversi:

- **EvaluatingVisitor** analizza la veridicita` dell'espressione
- **InfixPrinterVisitor** stampa l'espressione con la notazione infissa
- **PostfixPrinterVisitor** stampa l'espressione con la notazione postfissa o polacca inversa)
- **PaintVisitor** disegna graficamente l'albero dell'espressione

Diagramma delle classi del Visitor