Implementazione di Linguaggi 2

Federico Bernardi

- Type checking 2° parte
  - Equivalenza di type expressions
  - Type conversion

Implementazione di Linguaggi 2

Testi di riferimento:

- Compilers, Principles, Technique, and Tools
- Lucidi Prof. Ancona

Equivalenza di Type expressions

- I linguaggi di programmazione usano diversi modi e definizioni per stabilire che due tipi sono equivalenti.
- I due approcci principali sono:
  - Strutturale (Fortrand, C, C++, ...)
  - Per nome (Modula, Oberon, Ada)

Equivalenza Strutturale

Due espressioni sono strutturalmente equivalenti se sono lo stesso tipo di base o sono ottenute applicando lo stesso costruttore a tipi strutturalmente equivalenti.

In altre parole due espressioni sono strutturalmente equivalenti se e solo se sono identiche.

Equivalenza Strutturale

Algoritmo per definire in maniera ricorsiva se due tipi sono strutturalmente equivalenti:

```
Function sequiv(s,t): boolean
Begin
    if s and t are the same type then
    return true
    else if s = array(s1,s2) and t = array(t1,t2) then
    return sequiv(s1,t1) and sequiv(s2,t2)
    else if s = s1 x s2 and t = t1 x t2 then
     return sequiv(s1,t1) and sequiv(s2,t2)
    else if s = pointer (s1) and t = pointer(t1) then
     return sequiv(s1,t1)
    else if s = s1 → s2 and t = t1 → t2 then
     return sequiv(s1,t1) and sequiv(s2,t2)
    else
    return false
end
```

Codifica di una Type Expression

- Spesso è possibile rappresentare type expression in modo codificato (stringhe di bit o interi binari) in modo che l'equivalenza possa essere verificata efficientemente anche con il confronto di bit.
- Vediamo un esempio preso dal compilatore di C scritto da Ritchie.

Codifica di una Type Expression

- Vogliamo confrontare:
  - char
  - freturns(char)
  - pointer(freturns(char))
  - array(pointer(freturns(char)))

- Tutte queste espressioni possono essere rappresentate con una sequenza di bit usando un semplice schema.

Codifica di una Type Expression

Usiamo due bit per codificare il costruttore

Type Constructor	Encoding
pointer	01
array	10
Freturns	11

Codifica di una Type Expression

Usiamo quattro bit per codificare i tipi base

Type Constructor	Encoding
boolean	0000
char	0001
integer	0010
real	0011

Codifica di una Type Expression

Le espressioni di tipo precedentemente descritte si possono codificare in questo modo:

Type Constructor	Encoding
char	000000 0001
freturns(char)	000011 0001
pointer(freturns(char))	000111 0001
array(pointer(freturns(char)))	100111 0001

Codifica di una Type Expression

- Due differenti sequenze di bit non possono rappresentare lo stesso tipo perché i tipi base o i costruttori delle type expression sono differenti.

- Questo sistema è più efficiente dei quello che verifica l’equivalenza ricorsivamente.

Equivalenza per nome

- Il problema dei nomi:

```
Type   link = ^cell;
Var    next : link;
       last : link;
       p    : ^cell;
       q,r  : ^cell;
```

- Le variabili next, last, p, q, r sono tutte dello stesso identico tipo?
- Dipende!!! Per esempio in Iso-Pascal NO!

Equivalenza per nome

- Quando si ammettono nomi nelle Type Expression, l'equivalenza delle espressioni dipende da come vengono trattati i nomi.

Abbiamo due alternative:

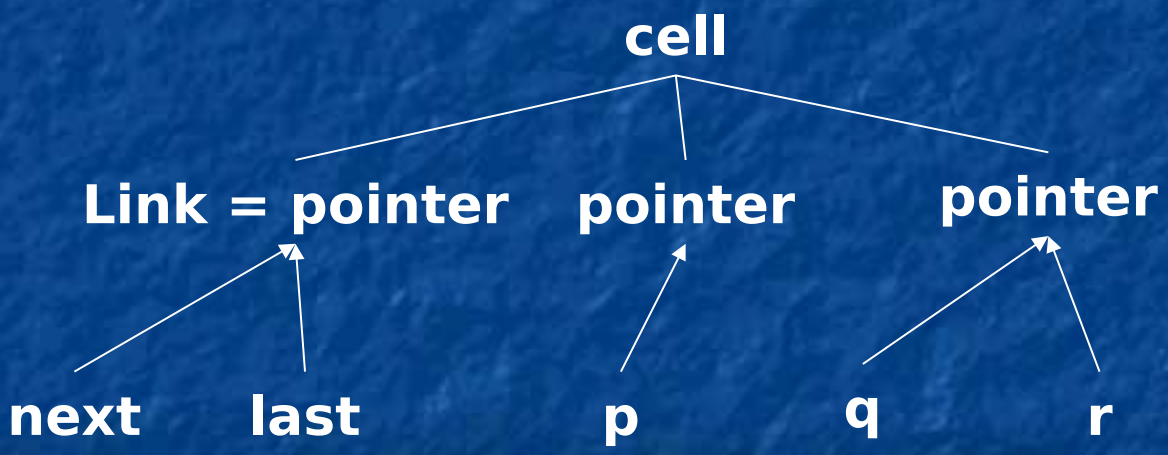
variabile	type expression	Equiv. Strutt.	Equiv. nome	Pascal
next	link	Stesso	Stesso	Tipo
last	link		Tipo	
p	pointer(cell)			Tipo
q	pointer(cell)	Tipo	Stesso	Tipo
r	pointer(cell)		Tipo	

Equivalenza per nome

- Il Pascal associa lo stesso nome per difetto a tutte le dichiarazioni di variabile associate allo stesso tipo.
- I tipi vengono descritti, nell' implementazione, da un type graph.
- Ogni volta che viene scandito un nuovo tipo costruttore o tipo base viene aggiunto un nuovo nodo.
- Ogni volta che viene scandito un nome di tipo viene creata una nuova foglia tenendo traccia della type expression cui si riferisce.

Equivalenza per nome

- Con questa rappresentazione due type expression sono equivalenti se esse sono rappresentate dallo stesso nodo del grafo.



associazioni tra variabili e nodi nel grafo

Trattamento dei cicli nelle definizioni di tipo

- Strutture di dati basi come liste lincate e alberi sono definiti spesso ricorsivamente.
- Queste strutture di dati sono solitamente implementate usando record che contengono puntatori a record simili

Trattamento dei cicli nelle definizioni di tipo

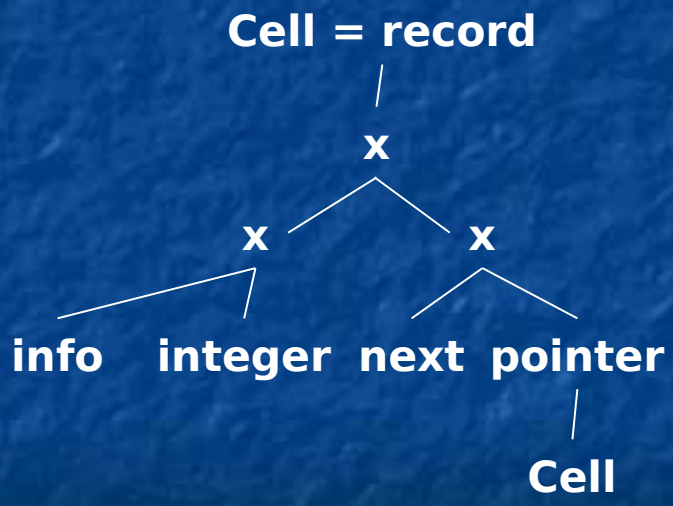
- Consideriamo un esempio con una lista lincata contenente un valore intero ed un puntatore alla cella successiva della lista.
- Pichiarazione del tipo alla Pascal:

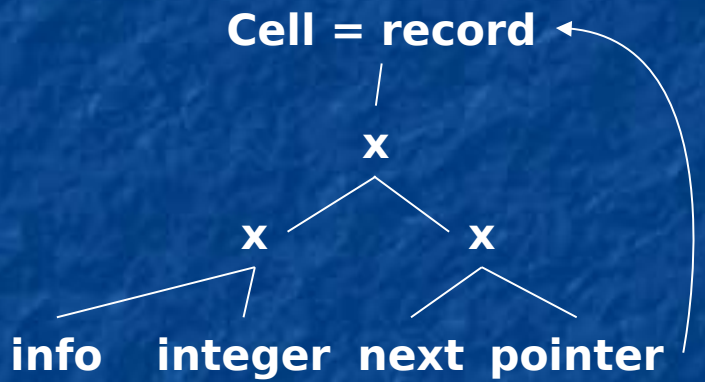
```
Type  link = ^cell;
      cell = record
                  info : integer;
                  next : link;
             end;
```

- Link è definito usando cell e cell è definito usando link (definizione ricirsiva).

Trattamento dei cicli nelle definizioni di tipo

- I nomi di tipo definiti ricorsivamente possono essere sostituiti se introduciamo un ciclo nel grafo dei tipi.





Usando un cicli possiamo eliminare la menzione a cell

Trattamento dei cicli nelle definizioni di tipo

- Il C risolve i cicli nel grafo dei tipi usando l'equivalenza strutturale per tutti i tipi eccetto che per i record. Una dichirazione di cell i C sarebbe:

```
Struct cell{
   int info;
   struct cell *next;
}
```

- Il C usa una rappresentazione aciclica come nella prima figura.
- C e Pascal richiedono che i nomi dei tipi vengano dichiarati pima di essere usati, con l'eccezione di puntatori a tipi non dichiarati.

Type conversion

- Consideriamo un'espressione come x+i dove x è un reale ed ì è un intero.
- Poiché la loro rappresentazione interna è differente, sono differenti anche gli algoritmi per le operazioni aritmetiche.
- Il compilatore sa che è necessaria una conversione di tipo, usualmente la conversione è verso l'elemento alla sinistra

Type conversion

- Il type checker può essere usato allo scopo introducendo una notazione postfissa come la seguente:

  x i inttoreal real+

- Quando la conversione viene automaticamente fatta dal compilatore si chiama “Coercions”.

Overloading

- E’ possibile, in molti linguaggi, attribuire ad un’ entità significati diversi a seconda del contesto in cui compare.

Ad esempio A+B può significare:

- Somma tra interi
- Somma tra reali
- Somma tra complessi
- Somma di matrici
- Somma di insiemi
- .....

Overloading

- L’overload si dice risolvibile quando è possibile dedurre un unico significato di un simbolo in un’istanza di un programma.
- NON è sempre possibile risolvere l’overload dall’analisi del contesto.