DCC - Dynamic Creme Caramel

Ivan Simonini

20 gennaio 2010

1 Scelta del progetto

I puntatori sono una buona cosa per qualunque linguaggio. Non sono estremamente complessi da realizzare. Lo heap si rende necessario per l'utilizzo dinamico della memoria, reso possibile dai puntatori. Il conteggio dei riferimenti è un approccio più elegante e pulito rispetto alla dimenticanza permessa dal garbage colletion.

2 Ridefinizione delle strutture

Avendo a disposizione gli oggetti dell'OCaml, é comodo usarli per semplificare e nascondere - dove possibile - il modo in cui le cose funzionano veramente, perché quello strato non è di diretta competenza e sinceramente sono pić contento di avere un oggetto con memoria, scelta che riduce lo spreco di memoria (almeno credo) ed anche il numero di parametri.

3 Realizzazione dei puntatori

Oltre all'ovvia modifica al parser e all'albero di sintassi, realizzata come visto in classe durante il corso, il puntatore non é che un valore-locazione, ossia un indice di cella. Il descrittore di un puntatore, oltre al tipo finale puntato (un gType, prossimamente), la profondità del puntatore stesso, ossia il numero di CARET e il massimo numero di derefenziazioni.

Il nodo sintattico DeRef è una struttura che conta il numero di CARET che precedono un identificatore. Se questo numero è minore o uguale alla profondità del puntatore identificato (ovviamente, soltanto se esso è un puntatore), allora il metodo do_defer_value raggiunge la cella corrispondente e procede a seguire il percorso indicato.

Il nodo sintattico Ref associa - direttamente nell'ambiente - la cella di residenza di un identificatore (presente nel suo descrittore) qualora questo non sia una costante.

4 Realizzazione dello Heap

Sostanzialmente analogo allo Store, lo heap è realizzato mediante un'hashtable che associa locazioni a valori. Questa volta l'associazione non avviene in un solo passo, poichè lo Heap ha la responsabilità, come obiettivo del progetto d'esame, di conteggiare i riferimenti alla cella. Ogni descrittore - il tipo HEntry, Heap Entry - include un intero, inizializzato a 0 al momento della creazione ed opportunamente manipolato ad ogni modifica, che conta il numero di puntatori ad essa collegati.

5 Malloc e Free

Prendendo come modello il lunguaggio C, le primitive di richiesta di allocazione e di deallocazione sono state chiamate Malloc e Free.

La Malloc è una speciale espressione aritmentica che ha il compito di fare da parte destra di un'assegnazione a puntatore, restituendo il valore locazione della prima cella allocata. Essa comporta la creazione di una o più celle nello Heap il cui conteggio iniziale è pari a zero. Se il valore viene assegnato ad un puntatore adatto, questo valore diventa immediatamente 1. Le celle pendenti a zero riferimenti sono patibili di riallocazione.

La Free è un comando che si occupa distruggere una zona allocata, INDIPENDENTEMENTE da quanti riferimenti ad essa esistano attualmente.

6 Concatenazione (virtuale) di Store e Heap

Grazie alla diversificazione tra StoreLoc e HeapLoc, è possibile raggruppare virtualmente le due zone di memoria per ottenere un accesso alla cella puntata tramite un puntatore.