Bollettino
Unione Matematica Italiana

Sezione A – La Matematica nella Società e nella Cultura

Baldan Dante

Integrazione dell'interpretazione astratta in un compilatore Prolog basato sulla WAM. Appendice: Un passo verso una metodologia per lo sviluppo di programmi Mercury: una semantica dichiarativa per Mercury

Bollettino dell'Unione Matematica Italiana, Serie 8, Vol. **3-A**—*La Matematica nella Società e nella Cultura* (2000), n.1S, p. 17–19.

Unione Matematica Italiana

<http://www.bdim.eu/item?id=BUMI_2000_8_3A_1S_17_0>

Bollettino dell'Unione Matematica Italiana, Unione Matematica Italiana, 2000.

Bollettino U. M. I.
La Matematica nella Società e nella Cultura
Serie VIII, Vol. III-A, Supplemento ad Aprile 2000, 17-19

Integrazione dell'interpretazione astratta in un compilatore Prolog basato sulla WAM. Appendice: Un passo verso una metodologia per lo sviluppo di programmi Mercury: una semantica dichiarativa per Mercury.

Baldan Dante

1. – Integrazione dell'interpretazione astratta in un compilatore Prolog basato sulla WAM.

Abbiamo studiato la progettazione di un buon compilatore Prolog che produca codice efficiente per programmi Prolog sulla base dell'informazione prodotta da una analisi statica di tali programmi. Abbiamo focalizzato la nostra attenzione su ottimizzazioni di basso livello per la WAM [5] basate su analisi descritte nell'ambito del quadro dell'Interpretazione Astratta [1].

I contributi principali di questa tesi sono i seguenti:

1. Abbiamo dimostrato che in molti casi interessanti è possibile produrre un compilatore ottimizzato con modifiche locali e semplici di un compilatore Prolog già esistente. Questo è chiaramente più semplice che costruire un compilatore interamente nuovo. Questo metodo generale è illustrato descrivendo come il compilatore SICStus è stato modificato in modo tale che, utilizzando l'informazione sulle variabili non-inizializzate e sulle variabili sicure, sia possibile generare codice migliore rispetto ad una compilazione che ignori tale informazione. In questo modo, con poco sforzo, abbiamo prodotto un compilatore ottimizzato che mantiene tutte le caratteristiche desiderabili del compilatore SICStus originale.

2. Presentiamo, all'interno del quadro classico dell'Interpretazione Astratta dei Cousot, sia l'analisi delle variabili non-inizializzate sia l'analisi delle variabili sicure. Abbiamo definito una semantica generica di punto fisso da cui, attraverso opportune interpretazioni, abbiamo ottenuto l'analisi delle variabili non-inizializzate e sicure:

- l'informazione calcolata dall'analisi delle variabili non inizializzate è utile per estendere la WAM con le variabili non-inizializzate il cui principale vantaggio è il risparmio di inizializzazioni e di de-inizializzazioni di locazioni di memoria. La novità è che l'analisi delle variabili non-inizializzate è formalizzata nel quadro dell'Interpretazione Astratta dei Cousots. I risultati sperimentali ci permettono di

dire che la nostra analisi è precisa quanto quella presentata da Van Roy [4] e quanto l'analisi monovariante presentata da Lindgren [3];

- l'analisi delle variabili sicure calcola informazione che permette di sostituire in modo corretto alcune istruzioni WAM con codice WAM più efficiente. Più precisamente, l'analisi delle variabili sicure determina un insieme di variabili le cui locazioni di memoria possono essere deallocate senza nessun controllo da parte della WAM. L'analisi delle variabili sicure non è mai stata proposta in letteratura.

3. Le due analisi descritte sopra sono state implementate usando il ben noto Analizzatore Generico di Interpretazione Astratta per Prolog GAIA [2]. Il software disponibile consiste in un unico programma che esegue entrambi le analisi.

4. L'analizzatore menzionato al punto 3 è stato integrato nel Compilatore ed Emulatore SICStus 2.1. Il sistema ottenuto è un'estensione di SICStus che, attraverso l'informazione calcolata dall'analizzatore, produce codice ottimizzato per programmi Prolog.

Alcuni dei risultati citati sono già stati presentati al SAS'97 e al Post-ILPS'97 Workshop su Parallelism and Implementation Technology for (Constraint) Logic Programming Languages.

2. – Un passo verso una metodologia per lo sviluppo di programmi Mercury: Una semantica dichiarativa per Mercury.

Lo scopo della seconda parte della tesi è la definizione di una semantica dichiarativa per Mercury da usare per la costruzione sistematica di programmi Mercury.

Il disegno di metodologie per la costruzione di programmi prese origine dal fatto che la semantica dichiarativa dei programmi Prolog differisce da quella procedurale. Parecchi tentativi sono stati fatti per migliorare o per arricchire il linguaggio Prolog (IC-Prolog, EPILOG, METALOG, MU-Prolog, e Prolog II).

Un punto di vista opposto può essere adottato. Benché il disegno di migliori programmi logici sia un problema fondamentale, è anche importante presentare metodologie per la costruzione di programmi logici per linguaggi realmente ed ampiamente usati. Tale approccio è stato seguito nella definizione della motodologia di Deville per la costruzione sistematica di programmi logici che produce programmi Prolog efficienti da una specifica informale. La metodologia di Deville è generica e solo nell'ultimo stadio prende esplicitamente in considerazione il linguaggio Prolog. La maggior parte della proposta di Deville può dunque essere applicata ad altri linguaggi logici, tra cui Mercury.

La costruzione di programmi Mercury richiede una forma di ragionamento esplicito, cioè un programma Mercury non è specifica e programma allo stesso tempo, ma è solo un programma. Dunque, vi è la necessità di una semantica di-

chiarativa per Mercury, che non è stata ancora definita e che è il principale contributo della seconda parte della tesi.

Gli altri contributi sono:

1. Proponiamo una sintassi astratta, chiamata *AS*, che descrive un sovra-insieme dei programmi Mercury accettabili. *AS* fornisce la parte libera da contesto della descrizioe del linguaggio, senza considerare le regole dei tipi.

2. Al fine di esprimere le regole dei tipi, introduciamo una sintassi astratta, chiamata *TAS*, che è una versione estesa di *AS*, in cui tutti i termini sono annotati con il, cosidetto, tipo apparente. I tipi apparenti sono convenienti per esprimere la relazione tra il tipo di un termine e i tipi dei sottotermini. Questa relazione è poi definita attraverso un insieme di regole di type-checking, che ci permettono di verificare se un programma in *TAS* è corretto (lo chiameremo accettabile).

3. Infine, per stabilire che un programma è accettabile, forniamo un insieme di regole di inferenza dei tipi che permettono di tradurre un programma in *AS* in un programma in *TAS*. Il programma originale è accettabile se e solo se la traduzione ha successo. In questo caso, il programma ottenuto è accettabile nel nostro senso tecnico e le sue annotazioni di tipo mostrano esplicitamente l'informazione lasciata implicita nel programma originario.

Alcuni dei risultati citati sono stati pubblicati in LOPSTR'98.

BIBLIOGRAFIA

[1] COUSOT P. e COUSOT R., *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, ACM Press, New York, (January 1977) 238-252.

[2] LE CHARLIER B. e VAN HENTENRYCK P., *Experimental evaluation of a generic abstract interpretation algorithm for PROLOG*, ACM Transactions on Programming Languages and Systems, **16(1)**, (January 1994), 35-101.

[3] LINDGREN T. *Polyvariant detection of uninitialized arguments of prolog predicates*, Journal of Logic Programming, **28(3)**, (September 1996), 217-229.

[4] VAN ROY P., *Can Logic Programming Execute as Fast as Imperative Programming*, PhD thesis, Computer Science Division, University of California Berkeley, (December 1990).

[5] WARREN D. H. D., *An Abstract PROLOG Instruction Set*, Technical Report 309, Artificial Intelligence Center, Computer Science and Technology Division, SRI International, Menlo Park, CA, (October 1983).

Dipartimento di Matematica Pura ed Applicata
Università degli Studi di Padova, Via Belzoni 7 - 35131 Padova
e-mail: dante@math.unipd.it
Dottorato in Matematica Computazionale ed Informatica Matematica
(sede amministrativa: Padova) - Ciclo X
Direttore di ricerca: Prof. Gilberto Filè, Università di Padova