Ragionamento automatico

I semestre, Anno accademico 2018-19

Progetto: Implementazione di chiusura di congruenza per la soddisfacibilità equazionale

Assegnato il 29 novembre, da consegnare entro le 14:00 del 4 febbraio

Assegnamento

1. Implementare in un linguaggio di programmazione a scelta l'*algoritmo di chiusura di congruenza su grafo diretto aciclico* per decidere la soddisfacibilità di un insieme di letterali nel frammento senza quantificatori della teoria dell'eguaglianza.

   - Come visto in classe, non è necessario verificare la presenza di una contraddizione solo alla fine del calcolo della chiusura di congruenza. In pratica, si vuole scoprire una contraddizione al più presto possibile. Un modo per farlo è dotare ogni termine `t` di una lista di *termini proibiti* o *lista proibita*. Supponiamo che `s` sia nella lista proibita di `t`. Non appena una chiamata `merge t s` è pronta per essere lanciata, significa che c'è una contraddizione. Poichè l'algoritmo non genera eguaglianze negate, la lista proibita di `t` contiene `s` se l'ingresso contiene `t != s`.
   - Nella versione di `union` sul Bradley-Manna, quale dei due rappresentanti diventi il rappresentante della nuova classe è una scelta arbitraria. In un'implementazione è opportuno definire un criterio di scelta: per esempio, stabilire che il rappresentante della nuova classe sia il rappresentante della più numerosa tra le due classi che vengono unite. L'intuizione è che in questo modo più pochi termini avranno catene più lunghe di chiamate a `find` per arrivare al loro rappresentante.
   - Nell'algoritmo sul Bradley-Manna, si arriva al rappresentante di un termine mediante una catena di chiamate a `find`, perchè quando si uniscono due classi, si modifica solo il campo `find` del rappresentante. Un'alternativa è implementare l'algoritmo in modo tale che tutti i termini di una classe abbiano nel loro campo `find` l'identificatore del rappresentante della classe. In tal modo l'operazione `find` diventa elementare, ma `union` deve essere modificata per aggiornare il campo `find` di tutti i termini della classe il cui rappresentante viene modificato. Se si procede così ha ancora più senso scegliere come rappresentante della nuova classe prodotta da `union` il rappresentante della più numerosa tra le due classi.

2. Svolgere esperimenti con l'implementazione provando insiemi di letterali da:

- Esempi ed esercizi da tutti i libri consigliati per il corso, e dalla bibliografia in calce, anche trasformando in insiemi di letterali equazionali quelli scritti in logiche più generali:
  - Se il problema include simboli di predicato diversi dall'eguaglianza, si implementi la trasformazione spiegata in classe per eliminarli;
  - Se il problema include simboli di funzione o costante definiti (e.g., le operazioni dell'aritmetica, numeri), li si rimpiazzino con simboli di funzione o costante liberi (e.g., f, g, h, ..., a, b, c);
  - Se il problema è dato in forma normale congiunta (i.e., un insieme di clausole), si implementi la trasformazione da forma normale congiunta a forma normale disgiunta e si applichi l'algoritmo di chiusura di congruenza a ogni disgiunto;
  - Se il problema include variabili quantificate universalmente, le si rimpiazzino con costanti o le si trattino come variabili libere;

  chiaramente non tutte queste trasformazioni preservano il problema, per cui si risolve un problema diverso da quello dato, ma intanto si ottengono più ingressi accettabili per l'algoritmo implementato nel progetto.
- La classe `QF-UF` sul sito `http://www.smt-lib.org/`: contiene formule nel frammento senza quantificatori (`QF`) della teoria dell'eguaglianza con simboli di funzione non interpretati (`UF`), dalle quali potrebbe essere possibile estrarre insiemi di letterali.

3. Scrivere una relazione (max 6 pagine a 11pt o 12pt) che presenti
   - Il solutore, con enfasi sulle principali scelte implementative (e.g., strutture di dati, euristiche);
   - I risultati degli esperimenti, in una o più tabelle, con dati quali tempo d'esecuzione, risposta (soddisfacibile/insoddisfacibile), o altro che si trovi interessante, avendo cura di riportare le fonti dei problemi;
   - Osservazioni sulle prestazioni, impatto delle euristiche, e altro che si ritenga rilevante.

La scelta del linguaggio di programmazione è libera. Il programma deve essere eseguibile su Linux. L'implementazione deve avere un'interfaccia che permetta all'utente di dare in ingresso un insieme di letterali e ricevere in uscita la risposta: si può creare una semplice interfaccia grafica oppure standard input e standard output da comando di linea.

Da consegnare

1. Una stampa della relazione, da lasciare nella cassetta delle lettere della docente (**Ricordare di proteggere l'ambiente stampando fronte-retro!**);

2. Un pacchetto contenente codice sorgente, eseguibile, problemi usati per gli esperimenti (includendo la fonte del problema), e un documento README con istruzioni per eseguire il programma, da consegnare per posta elettronica. Si mettano i documenti in una cartella denominata NomeCognomeNumeroMatricola e la si comprima (e.g., `tar czvf - NomeCartella > NomeCartella.tgz` comprime e `tar xzpvf NomeCartella.tgz` decomprime).

Bibliografia

- Libri di testo:

  1. Aaron R. Bradley, Zohar Manna. *The Calculus of Computation. Decision Procedures with Applications to Verification.* Springer, 2007, ISBN 978-3-642-09347-0.
  2. Daniel Kroening, Ofer Strichman: *Decision Procedures. An Algorithmic Point of View*, Springer, 2008, ISBN: 978-3-540-74104-6.

- Articoli:

  1. Leo Bachmair, Ashish Tiwari and Laurent Vigneron. Abstract congruence closure. Journal of Automated Reasoning 31(2):129–168, 2003.
  2. David L. Detlef, Greg Nelson and James B. Saxe. Simplify: a theorem prover for program checking. Journal of the ACM 52(3):365–473, 2005.
  3. Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. Journal of the ACM 27(4):758–771, 1980.
  4. Dexter Kozen. Complexity of finitely presented algebras. Technical Report TR-76-294, Department of Computer Science, Cornell University, 1976.
  5. Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. Journal of the ACM 27(2):356–364, 1980.
  6. Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. Information and Computation 205:557–580, 2007.
  7. Robert E. Shostak. An algorithm for reasoning about equality. Communications of the ACM 21(7):583–585, 1978.