Progetto GNCS 2012

Specifiche insiemistiche eseguibili e loro verifica formale

Unità: Università di Catania:
Cantone, Longo, Nicolosi-Asmundo
Università di Parma:
Chiarabini, Rossi
Università di Trieste:
Casagrande, Omodeo

Responsabile: Gianfranco Rossi, Università di Parma

Indice presentazione:

- Contesto e base di partenza
- Obiettivi
- Lavoro svolto e risultati ottenuti

Specifiche insiemistiche eseguibili

Insiemi: potente astrazione su dati e operazioni

Immersi in un linguaggio logico del prim'ordine
permettono di **modellare in modo naturale** vasta gamma di soluzioni a problemi complessi

Es.: nozioni di **grafo** e di **stringa** e tutte le usuali operazioni su questi [*Tomescu, 2012*]

Es.: vari linguaggi della **logica descrittiva**
traducibili in un linguaggio logico-insiemistico [Cantone, Longo, Pisasale, 2010] [Cantone, Longo, Nicolosi~Asmundo, 2011]

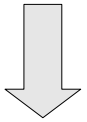
Specifiche insiemistiche eseguibili - cont.

Linguaggio logico-insiemistico:

capacità di **esprimere formula** che modella un dato problema

+

strumenti per **verificare se** tale formula sia **soddisfacibile** o no (e ottenere le soluzioni al problema, quando ne esistano): opportuni **risolutori** (solver) incorporati nel linguaggio



specifiche **eseguibili**

linguaggi di **programmazione dichiarativa**

Un esempio

Problema: *trovare il massimo m di una collezione S di numeri interi*

Soluzione: con linguaggio di progr. convenzionale – es. C++

```
int m = S[0];
for (int i=1; i<n; i++)
   if (S[i] > m)
       m = S[i];
```

Specifiche insiemistiche eseguibili - cont.

Soluzione: con linguaggio **logico-insiemistico**

invece di cercare procedimento algoritmico, si cerca di modellare il problema in termini di insiemi e operazioni su essi ==> formula logico-insiemistica

Es. (in linguaggio {log})

```
M in S & forall(Y in S, M >= Y)
```

Esecuzione:

se `S = {3,5,2,1}`, la formula è provata soddisfacibile con la sostituzione: `M = 5`

Dalla “Teoria Computabile degli Insiemi” ai “Vincoli Insiemistici”

Linguaggio logico e espressioni insiemistiche ammesse:
più o meno “complesse”:

- con quantificatori universali o solo esistenziali, annidati, semplici o ristretti
- operazioni su insiemi elementari o complesse
- insiemi di soli interi, o elementi di vario tipo
- insiemi piatti o annidati, a diversi livelli
- insiemi con tutti gli elementi noti o parzialmente specificati
- insiemi dati per enumerazione o definiti per proprietà

ecc.

Caratteristiche del linguaggio logico e delle espressioni insiemistiche determinano **“complessità” dei** relativi **risolutori**:
linguaggio più “ricco” ⇨ più complesso, e tipicamente inefficiente, risolutore associato

Dalla “Teoria Computabile degli Insiemi”
ai
“Vincoli Insiemistici”

Insiemi e formule logico-insiemistiche	*Aree di ricerca*	*Linguaggi e risolutori*
“complesse”	Computable Set Theory	`Referee/ AEtnaNova`
	(Constraint Logic) Programming with sets	`SETL, {log}, JSetL`
“semplici”	Set constraints	`Conjunto, Cardinal, ...`

Teoria Computabile degli Insiemi (CST)

Formule relativamente complesse in ambiti sintatticamente delimitati della teoria generale ZF per le quali la soddisfacibilità risulti decidibile ([Schwartz, Cantone, Omodeo 2011], [Cantone, Omodeo, Policriti 2001], [Cantone, Ferro, Omodeo 1989]).

Nei casi di linguaggi più espressivi: risolutori caratterizzati da **complessità computazionale spesso proibitiva**.

Alcuni **frammenti** insiemistici decidibili, quali MLSS e sue estensioni, ammettono risolutori relativamente efficienti. Implementati con successo nel nucleo inferenziale del sistema di verifica di dimostrazioni `Referee/AEtnaNova`

Interessati soprattutto ai **risultati di decidibilità** piuttosto che a computare (in modo efficiente) le possibili soluzioni.

Importante proprietà di **completezza** del risolutore

Programmazione logica con insiemi

Restringendo opportunamente il linguaggio logico-insiemistico utilizzato ==> risolutori più efficienti

Es. ***{log}*** (basato su *CLP(SET)*) [Dovier, Omodeo, Pontelli, Rossi 96]: linguaggio di **programmazione logica a vincoli** con insiemi

- forme insiemistiche relativamente generali
- forme semplici di quantificazione (esistenziale e universale ristretta)
- risolutore completo, con calcolo esplicito di tutte le soluzioni

Es. formule ammissibili:

```
1 ∈ S & union(S,R,T) & 1 ∉ T   --->  false
{1|R} = {2|S}   --->   R = {2|N}, S = {1|N}
```

Complessità computazionale più contenuta rispetto risolutori generali sviluppati in ambito CST, anche se, in generale, i problemi decisionali trattati rimangono NP-completi.

Programmazione dichiarativa con insiemi

Libreria JSetL [Rossi, Panegai, Poleo 2007]

stesso linguaggio logico-insiemistico di base di *{log}*, ma ospitato all'interno di un linguaggio object-oriented convenzionale (= **Java**)

Es.: *trovare il massimo m di una collezione S di numeri interi*

```
public static LVar max(Set s) throws Failure {
      LVar m = new LVar();
      LVar y = new LVar();

      Solver.add(m.in(s));
      Solver.forall(y, s, m.ge(y));

      Solver.solve();
      return m;
}
```

Vincoli insiemistici

Ulteriori semplificazioni su tipo formule e su forma insiemi ammissibili
+
rinuncia alla completezza del risolutore
=
linguaggi logico-insiemistici meno espressivi,
ma con risolutori molto **più efficienti**

"**Finite Set Constraints**" - es. *Conjunto* [Gervet '97]

Solo insiemi (piatti) di interi, completamente specificati
Solo quantificatori esistenziali ed operatori insiemistici di base
Ciascuna variabile insiemistica ha associato un **dominio finito**
= intervallo di insiemi; es. `{}..{1,2,3}`

Risolutori molto efficienti
utilizzati in varie applicazioni concrete:
problemi di allocazione risorse, di configurazione, di diagnostica, ...

Vincoli insiemistici - cont.

Disponibili risolutori efficienti per linguaggi con vincoli insiemistici
Es.

- in ambito CLP: ECLiPSe
- librerie per linguaggi object-oriented:
  ILOG, Choco, JaCoP, Koalog

Recentemente, comunità Java:
documento di specifica (**JSR-331**) per la definizione di un'interfaccia (API) standard per la programmazione con vincoli, che include vincoli insiemistici

Proposte orientate unicamente alla **soluzione efficiente** di problemi di soddisfacimento di vincoli (**CSP**),
piuttosto che al supporto più in generale di programmazione (dichiarativa) con insiemi

Obiettivi del progetto

Sviluppo di un ambiente che offra un linguaggio logico-insiemistico:

espressivo, da poter essere usato nei più svariati contesti per la codifica (dichiarativa) di problemi

+

efficiente, permetta di ottenere, quando serve, prestazioni computazionali adeguate

Approccio esemplificato da JSetL:
linguaggio insiemistico inserito all'interno di un linguaggio di programmazione convenzionale

Sotto-obiettivi

1. cercare di ottenere la **maggior efficienza** possibile dal risolutore dei vincoli insiemistici (anche rinunciando a completezza)

2. **“arricchire” il linguaggio** logico-insiemistico a disposizione.
A livello più teorico: ricerca su metodi d'inferenza (in particolare algoritmi di decisione) operanti su formule insiemistiche

3. verificare adeguatezza JSetL a **specifica JSR-331** per API std per la programmazione a vincoli definita dalla comunità Java

4. applicazioni: esprimibilità di logiche modali e descrittive (--> semantic web); specifica e verifica di correttezza (in `Referee`) di algoritmi operanti su grafi espressi in termini insiemistici; supporto alla generazione di test per linguaggi di specifica basati su insiemi come Z e B

Lavoro svolto, risultati

JSetL

Integrazione in JSetL di un risolutore di vincoli su **domini finiti di interi** e su **domini finiti di insiemi di interi (**seguendo l'approccio di [Gervet '97] e [Azevedo 2007]).

Aggiunta possibilità di trattare **intervalli** di interi, di insiemi e **multi-intervalli** di interi (es. `[1..5]+{10}+[20..30]`)

Nuova versione di JSetL disponibile all'indirizzo:
`cmt.math.unipr.it/jsetl.html`
Manuale JSetL disponibile come Rapporto di Ricerca del DMI di Parma

Lavoro svolto, risultati
JSetL - cont

JSetL utilizzato come uno dei solver scelti per implementare l'interfaccia standard definita nelle **specifiche JSR-331**

Lavoro svolto **in collaborazione con J. Feldman**, University College Cork

Implementazione JSR-331 basata su JSetL disponibile all'interno del gruppo di lavoro Java

Superato tutti i **test obbligatori**

Attualmente in corso altri test più complessi e completi

Programmazione (dichiarativa) **non-deterministica**

come sfruttare il non-determinismo insito nelle **operazioni insiemistiche** e nel **risolutore di vincoli** JSetL

per fornire una **semplice soluzione** a problemi naturalmente esprimibili in termini non-deterministici

come ad esempio **problemi su grafi** o realizzazione di semplici analizzatori sintattici (DCG)

==> G.Rossi, F.Bergenti: *“Nondeterministic Programming in Java with JSetL”*, in preparazione

Procedure di decisione per frammenti della teoria degli insiemi

Procedura di decisione per il **frammento** $\mathbf{3LQS^R}$
(variabili di tre livelli e quantificazione ristretta sulle variabili di livello 0 e di livello 1)
$3LQS^R$ in grado di esprimere diversi frammenti della sillogistica stratificata, un certo numero di costrutti della teoria degli insiemi come ad esempio il powerset e il prodotto cartesiano non ordinato nonché la logica modale S5.
==> [Cantone, Nicolosi-Asmundo 2012]

Procedura di decisione per il **frammento** $\mathbf{4LQS^R}$
(variabili di quattro livelli)
$4LQS^R$ in grado di esprimere relazioni binarie e loro proprietà (simmetria e transitività) e logiche modali normali quali K45
==> [Cantone, Nicolosi-Asmundo 2012]

Lavoro svolto, risultati

Procedure di decisione per frammenti della teoria degli insiemi - - cont

Procedura di decisione per il sottolinguaggio quantificato della teoria degli insiemi denominato $\forall^{\pi}_{0,2}$

due sorte di variabili: insiemi generici e mappe (insiemi di coppie ordinate), con relative operazioni
Procedura di decisione con tempo esponenziale non-det.

Identificato sottoinsieme con procedura di decisione con tempo polinomiale non-det.
(con applicazioni in rappresentazione della conoscenza)

==> [Cantone, Longo 2012], sottomessa anche a volume speciale di *Theoretical Computer Science*

Definizione di una tecnica di **Skolemizzazione globale**

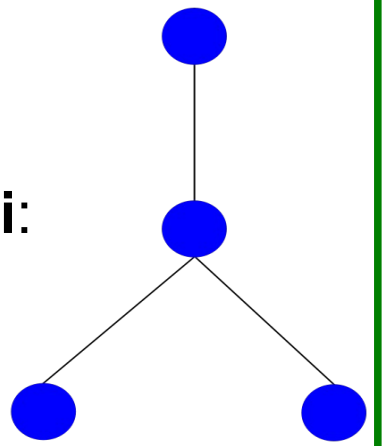
Ottimizzazione di un metodo per eliminazione quantificatori nella logica del I ordine introdotto da Davis e Fechter '91 (= definizione implicita dei quantificatori attraverso opportuni termini di Skolem):

permette di definire i quantificatori con termini di Skolem meno profondi e di ottenere pertanto dimostrazioni più brevi

==> [Cantone, Nicolosi-Asmundo, Omodeo 2012]

Lavoro svolto, risultati - cont

Sperimentazione con il **proof-checker AEtnaNova/Referee**



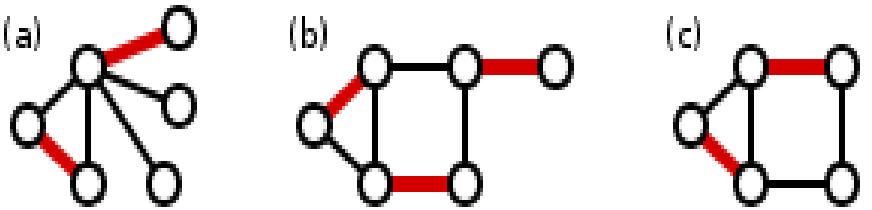
"**claw-free graph**": grafi che non contengono "claw"

"claw-free graph": rappresentabili in **termini insiemistici**: grafi con vertici degli insiemi e archi coppie non orientate di vertici dei quali uno appartenga all'altro.

Proprietà: ogni grafo claw-free ammette

- un **matching quasi-perfetto** (perfetto se num. vertici è pari)

(un matching in un grafo è un insieme di archi senza vertici comuni; è perfetto quando ciascun vertice del grafo è incidente ad esattamente un arco del matching)



- un **cammino hamiltoniano** (cammino che tocca tutti i vertici del grafo una e una sola volta)

Dimostrazione di queste proprietà sviluppata formalmente e verificata con il proof-checker

Lavoro svolto, risultati - cont

==> [Omodeo, Tomescu 2012] in monografia dedicata a J.T.Schwartz (e in corso di pubblicazione come “proof pearl” su *Journal of Automated Reasoning*)

Dimostrazione del **teorema di Stone** sulla rappresentazione di algebre di Boole come campi d'insiemi
==> articolo in corso di preparazione

In corso implementazione ex-novo di un **interprete per il linguaggio SETL**

Utilizzato anche nell'implementazione del proof-checker
AEtnaNova/Referee

Applicazione di **{log} come generatore di "test case"** per il "Test Template Framework" (TTF)

Problema: generazione dei "**casi di test**" per un programma

Approccio: model-based testing: i "casi di test" sono generati a partire dalla **specifica** del programma (non dal codice sorgente)

TTF: è un metodo di model-based testing per specifiche scritte in **Z** (linguaggio di specifica basato su logica del I ordine + set)

Data una specifica Z, TTF partiziona il suo spazio di input e genera delle "specifiche di test" = congiunzioni di predicati atomici scritti in Z.

Una qualsiasi **soluzione** di questa formula rappresenta un "**caso di test**" per il programma originario

Lavoro svolto, risultati
applicazione di {log} - cont

Applicazione di {log}: le “specifiche di test” scritte in Z vengono trasformate in **formule {log}** e quindi risolte tramite il constraint solver di {log}

{log} si è dimostrato **più efficiente e generale** di altri sistemi solitamente usati (es. ProB), anche per “specifiche di test” complesse.

Lavoro svolto **in collaborazione con Maximiliano Cristià** (Università di Rosario, Argentina)

==> articolo sottomesso a 11th *International Conference on Software Engineering and Formal Methods* (SEFM 2013), Madrid

$TransmitSD_{24}^{SP}$

$c, t : DTYPE \to \mathbb{N}$
$mem : 1 \,..\, 1024 \to BYTE$
$sdwp : \mathbb{N}$

$c\ SD = 0 \land sdwp < 3 \land 33 \,..\, 160 \neq \{\}$
$33 + (t\ SD - c\ SD) * 2 \,..\, 33 + (t\ SD - c\ SD + 1) * 2 \neq \{\}$
$33 \,..\, 160 \cap 33 + (t\ SD - c\ SD) * 2 \,..\, 33 + (t\ SD - c\ SD + 1) * 2 \neq \{\}$
$\neg\ 33 \,..\, 160 \subseteq 33 + (t\ SD - c\ SD) * 2 \,..\, 33 + (t\ SD - c\ SD + 1) * 2$
$\neg\ 33 + (t\ SD - c\ SD) * 2 \,..\, 33 + (t\ SD - c\ SD + 1) * 2 \subseteq 33 \,..\, 160$
$33 + (t\ SD - c\ SD) * 2 \,..\, 33 + (t\ SD - c\ SD + 1) * 2 \neq 33 \,..\, 160$

$TransmitSD_{24}^{TC}$

$TransmitSD_{24}^{SP}$

$c = \{sd \mapsto 0, hd \mapsto 1, md \mapsto 2\}$
$t = \{sd \mapsto 63, hd \mapsto 0, md \mapsto 1\}$
$sdwp = 0$
$mem = \{1 \mapsto G11084, 2 \mapsto G11116, \ldots \text{and 1022 more elements} \ldots\}$

Fig. 1. Typical test specification and test case in the TTF.

Programmazione con insiemi

{log}, JSetL:
linguaggi di programmazione concreta

Supporto a **programmazione dichiarativa** (“general-purpose”)

Risolutore integrato nel linguaggio di programmazione ospite
(relativamente) efficiente

Procedure di decisione per frammenti della logica relazionale

Procedura di decisione per un frammento in cui sono ammessi termini di tipo $R ; S$
dove S non contiene l'operatore di inversione ed R può contenere l'operatore di composizione solo in forma ristretta.

In grado di esprimere diverse logiche modali e descrittive

==> Risultati parziali in [Cantone, Nicolosi-Asmundo, Orlowska 2012]