Proof Search in Propositional Linear Logic via Boolean Constraints Satisfaction

Laurea Triennale in Informatica

Martino D'Adda (964827)

16 Luglio 2024

UNIVERSITÀ DEGLI STUDI DI MILANO

Indice

Calcolo dei sequenti

1 Introduzione

Il calcolo dei sequenti (G. Ghentzen, 1934) è un formalismo per rappresentare un calcolo logico e strutturarne dimostrazioni. I concetti principali sono due:

- il sequente $\Delta \vdash \Gamma$, che esprime una implicazione tra la congiunzione di Δ e la disgiunzione di Γ;
- la regola, che descrive come si possono comporre i sequenti,

$$\wedge \frac{\overbrace{\Delta \vdash \phi', \Gamma}^{\Pi'} \qquad \overbrace{\Delta \vdash \phi'', \Gamma}^{\Pi''}}{\underbrace{\Delta \vdash \phi' \wedge \phi'', \Gamma}_{\Pi}}$$

in questo caso se valgono Π' e Π'', allora vale Π.

Calcolo dei sequenti cont'd

Una dimostrazione consiste in un albero avente come radice il sequente da dimostrare e ottenuto concatendo regole.

$$\dfrac{\dfrac{A\ \dfrac{}{a, b \vdash b} \quad A\ \dfrac{}{a, b \vdash a}}{\wedge_R\ a, b \vdash b \wedge a}}{\wedge_L\ a \wedge b \vdash b \wedge a}$$

Un theorem prover è un programma che, dato un sequente, prova a costruirne la dimosrazione applicando le regole di sopra. Nello specifico un theorem prover è detto bottom-up se parte dal sequente da dimostrare, e lo continua a scomporre applicando le regole "al contrario".

Logica lineare

1 Introduzione

La **logica lineare** (J.-Y. Girard, 1987) è una logica la cui caratteristica principale è che è generalmente proibita la duplicazione o l'eliminazione di formule durante il processo di dimostrazione, operazioni che altresì sono spesso lasciate implicito. Questa scelta porta ad alcune proprietà particolari, tra cui la duplicazione dei connettivi della logica classica secondo due interpretazioni diverse, una detta moltiplicativa ed l'altra additiva.

$$\begin{array}{cccc} \wedge & & \otimes & \& \\ \vee & & \parr & \oplus \\ \top & \rightarrow & 1 & \top \\ \bot & & \bot & 0 \end{array}$$

Splitting

Durante il proof searching bottom-up uno dei problemi pincipali è quello dello **splitting**.

$$\otimes \frac{\vdash \Delta', \phi' \qquad \vdash \Delta'', \phi''}{\vdash \Delta', \Delta'', \phi' \otimes \phi''}$$

La decomposizione del tensore (regola di qui sopra) comporta il partizionamento di Δ in Δ' e Δ'' per continuare la dimostrazione. Questa operazione può richiedere un numero esponenziale di tentativi.
Nel 2001 D.Pym e J.Harland propongono un metodo per gestire lo splitting utilizzando vincoli booleani generati a partire da espressioni associate a ogni singola formula.

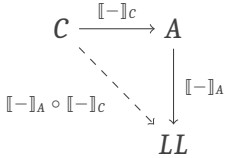
Indice

2 Il calcolo

Focusing e normalizzazione

2 Il calcolo

Partendo dal calcolo citato sopra di D.Pym e J.Harland, abbiamo applicato due classiche modifiche nell'ambito del proof searching (i.e. normalizzazione e focusing). Di questo nuovo calcolo – che chiameremo $\mathcal{C}$ – abbiamo mostrato la correttezza esibendo una traduzione verso un calcolo intermedio (A) tale che il seguente diagramma commuta:



con LL il calolo classico della logica lineare e $[\![-]\!]$ le traduzioni.

Implementazione

2 Il calcolo

Per il calcolo di sopra:

- è stata fatta un'implementazione in SWI-Prolog;

Implementazione

2 Il calcolo

Per il calcolo di sopra:

- è stata fatta un'implementazione in SWI-Prolog;
- è stato scritto un generatore di test in OCaml;

Implementazione

2 Il calcolo

Per il calcolo di sopra:

- è stata fatta un'implementazione in SWI-Prolog;
- è stato scritto un generatore di test in OCaml;
- è stato architettato un sistema per il testing e il benchmarking in Python.

Implementazione

2 Il calcolo

Per il calcolo di sopra:

- è stata fatta un'implementazione in SWI-Prolog;
- è stato scritto un generatore di test in OCaml;
- è stato architettato un sistema per il testing e il benchmarking in Python.

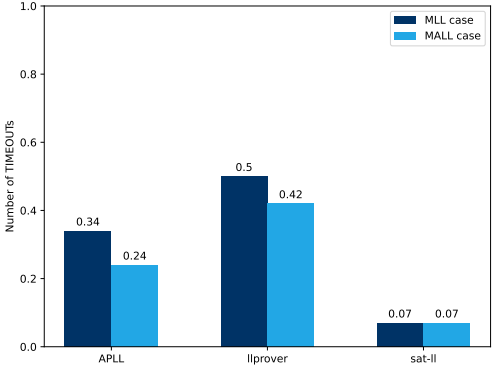
Infine l'infrastruttura per tutti i programmi di sopra è stata gestita con Nix.

Indice

3 Conclusione

Risultati – caso moltiplicativo

Una volta implementato il prover lo si è confrontato con altri due prover basati a grandi linee sullo stesso algoritmo (i.e. bottom-up focused proof search): llprover (1997) e APLL (2016). Nel caso moltiplicativo, quello principalemente soggetto al problema dello splitting, i risultati sono buoni:



Risultato – caso generale

Nel caso dei test esponenziali utilizziamo due dataset preesistenti e vediamo che i risultati si livellano:

prover	timeouts	failures	successes	success rate	avg. (succ.)	avg. (tot.)
APLL	0	17	71	≈ 0.80	0.035 s	0.326 s
llprover	20	6	62	≈ 0.70	0.981 s	2.179 s
sat-ll	5	15	68	≈ 0.77	0.443 s	0.496 s

(a) Outputs for KLE-cbv

prover	timeouts	failures	successes	success rate	avg. (succ.)	avg. (tot.)
APLL	0	16	72	≈ 0.80	0.037 s	0.055 s
llprover	20	6	62	≈ 0.70	1.709 s	3.253 s
sat-ll	4	18	66	≈ 0.75	0.130 s	0.185 s

(b) Outputs for KLE-cbn

Grazie per l'attenzione!