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

Calcolo dei sequenti

1 Introduzione

Il calcolo dei sequenti è un sistema formale logico introdotto da Gentzen nel 1935 che può essere usato per specificare la ricerca di prove (*proof search*) in logica:

Calcolo dei sequenti

Il calcolo dei sequenti è un sistema formale logico introdotto da Gentzen nel 1935 che può essere usato per specificare la ricerca di prove (*proof search*) in logica:

- il sequente $\Delta \vdash \Gamma$, di solito interpretato come

$$\delta_1 \wedge \cdots \wedge \delta_n \rightarrow \gamma_1 \vee \cdots \vee \gamma_m$$

Calcolo dei sequenti

Il calcolo dei sequenti è un sistema formale logico introdotto da Gentzen nel 1935 che può essere usato per specificare la ricerca di prove (*proof search*) in logica:

- il sequente $\Delta \vdash \Gamma$, di solito interpretato come

$$\delta_1 \wedge \cdots \wedge \delta_n \rightarrow \gamma_1 \vee \cdots \vee \gamma_m$$

- la regola, che descrive come manipolare i sequenti
  - descrive la semantica di un connettivo (introduzione e eliminazione):

$$\text{intro}\wedge \ \frac{\Delta \vdash \phi', \Gamma \qquad \Delta \vdash \phi'', \Gamma}{\Delta \vdash \phi' \wedge \phi'', \Gamma} \qquad \text{elim}\wedge \ \frac{\Delta, \phi', \phi'' \vdash \Gamma}{\Delta, \phi' \wedge \phi'' \vdash \Gamma}$$

  - descrive come trattare le formule all'interno del sequente (regole strutturali).

Calcolo dei sequenti

1 Introduzione

Il calcolo dei sequenti è un sistema formale logico introdotto da Gentzen nel 1935 che può essere usato per specificare la ricerca di prove (*proof search*) in logica:

- il sequente $\Delta \vdash \Gamma$, di solito interpretato come

$$\delta_1 \wedge \dots \wedge \delta_n \to \gamma_1 \vee \dots \vee \gamma_m$$

- la regola, che descrive come manipolare i sequenti
  - descrive la semantica di un connettivo (introduzione e eliminazione):

$$\text{intro}\wedge \, \frac{\Delta \vdash \phi', \Gamma \qquad \Delta \vdash \phi'', \Gamma}{\Delta \vdash \phi' \wedge \phi'', \Gamma} \qquad \text{elim}\wedge \, \frac{\Delta, \phi', \phi'' \vdash \Gamma}{\Delta, \phi' \wedge \phi'' \vdash \Gamma}$$

  - descrive come trattare le formule all'interno del sequente (regole strutturali).
- la dimostrazione, un albero avente come radice il sequente da dimostrare e ottenuto concatenando regole.

Proof search

Un **theorem prover** è un programma che, dato un sequente, prova a costruirne la dimostrazione. Nello specifico nella versione bottom-up:

1. si sceglie un *goal*, inizialmente il sequente iniziale;
2. lo si scompone applicando una regola ad una delle formule nel goal;
3. si aggiungono i nuovi goal al working-set;
4. si ripete dal passo 1 fino a che ci sono goal irrisolti.

Focusing

1 Introduzione

Nella maggior parte dei casi ci saranno più regole applicabili ad un sequente, ma non tutte le scelte sono uguali:

- certe sono *invertibili* e possono essere applicate in qualunque ordine senza perdere completezza (non-determinismo don't-care),
- altre richiedono scelte e creano punti di backtracking (non-determinismo don't-know).

Il **focusing** è una tecnica che suddivide la dimostrazione in due fasi che si alternano:

- asincrona: si applicano eagerly solo regole invertibili (non-determinismo don't-care);
- sincrona: ci si concentra su una formula e si continuano ad applicare regole non invertibili (non-determinismo don't-know).

Logica lineare

La **logica lineare** nasce dall'analisi delle regole strutturali e dall'abbandono di due di esse:

- *weakening*: nel sequente si possono sempre aggiungere formule;
- *contraction*: due copie della stessa formula possono essere contratte.

Nella logica che ne risulta:

- le formule possono essere viste come **risorse**;
- ogni formula deve essere usata esattamente una volta
- ogni connettivo ammette due versioni, una additiva ed una moltiplicativa, corrispondenti a due diverse interpretazioni dei connettivi classici;

Degli speciali connettivi, chiamati esponenziali, permettono di localizzare weakening e contrazione.

Gestione delle risorse

Durante il proof searching bottom-up in logica lineare un'ulteriore fonte di non-determinismo è la gestione delle risorse.

$$
\otimes_L \dfrac{\otimes_R \dfrac{A \dfrac{}{\beta \vdash \beta} \qquad A \dfrac{}{\alpha \vdash \alpha}}{\alpha, \beta \vdash \beta \otimes \alpha}}{\alpha \otimes \beta \vdash \beta \otimes \alpha}
$$

Nello specifico la scomposizione dei moltiplicativi spesso comporta un'operazione chiamata **splitting**, cioè il partizionamento del contesto.

Indice

2 Il nostro calcolo

Nel 2001 D.Pym e J.Harland propongono una gestione delle risorse basato su **vincoli booleani**. Questi

- sono generati durante la proof-search a partire da espressioni associate alle formule;
- possono essere solo di due tipi:
  - una certa formula è stata usata in questo goal, e non può essere usata altrove;
  - una certa formula non è stata usata in questo goal, e deve essere usata altrove;

In questo modo è possibile non partizionare fisicamente il contesto

$$\dfrac{\dfrac{\dfrac{}{\alpha, \beta \vdash \beta}\,A \quad \dfrac{}{\alpha, \beta \vdash \alpha}\,A}{\alpha, \beta \vdash \beta \otimes \alpha}\,\otimes_R}{\alpha \otimes \beta \vdash \beta \otimes \alpha}\,\otimes_L$$

Calcolo dei vincoli cont’d

Rendiamo il calcolo dei vincoli focused, e di questo nuovo calcolo mostriamo la correttezza esibendo una traduzione tale che il diagramma a lato commuta.
Con:

- C il nostro calcolo;
- A il calcolo focused senza constraint;
- LL il calcolo classico della logica lineare.

e $[\![-]\!]$ le rispettive traduzioni.

$$
\begin{tikzcd}
C \arrow[r, "{[\![-]\!]_C}"] \arrow[dr, dashed, "{[\![-]\!]_A \circ [\![-]\!]_C}"'] & A \arrow[d, "{[\![-]\!]_A}"] \\
 & LL
\end{tikzcd}
$$

Implementazione

2 Il nostro calcolo

Per il calcolo di sopra sono state scritte:

Implementazione

Per il calcolo di sopra sono state scritte:

- un’implementazione in SWI-Prolog;

Implementazione

2 Il nostro calcolo

Per il calcolo di sopra sono state scritte:

- un'implementazione in SWI-Prolog;
- un generatore di test in OCaml;

Implementazione

Per il calcolo di sopra sono state scritte:

- un'implementazione in SWI-Prolog;
- un generatore di test in OCaml;
- una libreria per il testing e il benchmarking in Python.

Implementazione

Per il calcolo di sopra sono state scritte:

- un'implementazione in SWI-Prolog;
- un generatore di test in OCaml;
- una libreria per il testing e il benchmarking in Python.

Infine l'infrastruttura è stata gestita con Nix.

SWI-Prolog

Nello specifico è stato scelto SWI-Prolog per:

- la sua capacità di esprimere le regole in modo dichiarativo;
- il suo supporto per il backtracking;
- le sue librerie per il constraint logic programming (CLP), che offrono una elegante interfaccia verso risolutori di vincoli booleani;
- l'unificazione che permette di gestire in automatico la propagazione degli assegnamenti delle variabili booleane.

Indice

3 Sperimentazione

Risultati

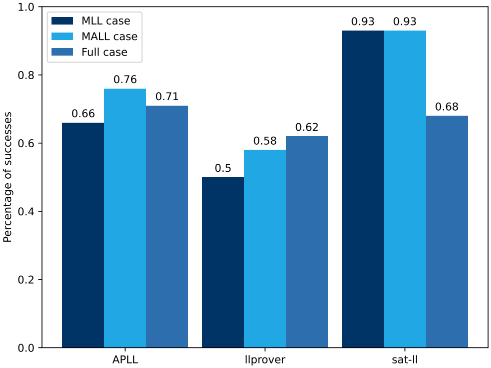
3 Sperimentazione

Il prover è stato confrontato con altri due prover:

- llprover (Prolog, 1997)
- APLL (OCaml, 2019)

Le fonti dei test sono state principalmente due:

- test generati (MLL e MALL);
- un dataset di traduzioni dalla logica intuizionista per i test esponenziali.



Conclusione e lavori futuri

Del calcolo dei vincoli

- abbiamo fornito un nuovo calcolo che utilizzi il focusing, di cui abbiamo dimostrato la correttezza;
- abbiamo fornito un'implementazione in Prolog;
- abbiamo confrontato questa implementazione con altri prover simili e mostrato che nel caso moltiplicativo si ottengono buoni risultati.

Possibili sviluppi futuri possono essere:

- affinamento dell'algoritmo nel caso esponenziale;
- scrittura di altri calcoli e prover simili per altre logiche sotto-strutturali (ILL, BI, ...).

Fine.