1 Ente presso cui è stato svolto il lavoro di stage

Il lavoro di tirocinio è stato di tipologia interna presso il dipartimento di Informatica dell'Università degli Studi di Milano, sotto la supervisione del docente Alberto Momigliano e il docente Camillo Fiorentini.

2 Contesto iniziale

Nell'ambito dei dimostratori automatici bottom-up basati sul calcolo dei sequenti per la logica lineare, una delle principali fonti di complessità è un'operazione chiamata splitting. Questa richiede che ripetutamente durante la computazione della dimostrazione sia necessario partizionare le conclusioni o le assunzioni, portando ad una esplosione combinatoria che si aggiunge all'inerente complessità del proof searching.

Il lavoro di tirocinio inizia da un articolo del 2001 di J. Harland e D. Pym [2] dove si propone un metodo alternativo per affrontare lo splitting affidandosi a vincoli booleani. I vincoli vengono generati in modo tale che se questi sono soddisfacibili, allora la dimostrazione è corretta; in questo modo i multiset di formule non vengono mai effettivamente partizionati, e la complessità viene spostata dalla generazione dei sottoinsiemi corretti, alla ricerca di un assegnamento per i vincoli booleani.

3 Obiettivi del lavoro

L'obiettivo principale del lavoro è stato implementare un prover basato sul calcolo di cui sopra e, una volta fatto questo, valutarne la efficacia anche rispetto ad altri prover esistenti.

4 Descrizione del lavoro svolto

Inizialmente è stato implementato un prover basato direttamente sul calcolo di [2], ma questo si è rivelato inefficiente. Successivamente sono state applicate alcune modifiche ormai assodate nell'ambito del theorem proving quali la normalizzazione e il focusing [1]. Di questo nuovo calcolo è stata prima dimostrata la correttezza, esibendo una traduzione verso il calcolo triadico di [1], poi ne è stata fatta una implemetazione in Prolog. Infine il programma è stato confrontato con due altri prover simili: APLL [4] e llprover [3].

5 Tecnologie coinvolte

Sono state coinvolte le seguenti tecnologie:

- il linguaggio SWI-Prolog per la scrittura del prover, con particolare enfasi sulle sue librerie per il constraint logic programming (CLP);
- il linguaggio OCaML per la scrittura di un generatore di formule randomiche;
- il linguaggio Python e Jupyter Notebook per l'infrastruttura necessaria per comparare diversi prover;

- il linguaggio Nix (in particolar modo l'estensione dei flake) e bash per l'automazione dei processi di compilazione e la generazione di ambienti di sviluppo riproducibili.

6 Competenze e risultati raggiunti

I benchmark hanno evidenziato come l'utilizzo dei vincoli booleani, assieme al focusing e alla normalizzazione, permettano di ottenere risultati competitivi nell'ambito della logica lineare moltiplicativa.

Il lavoro di tirocinio mi ha dato l'opportunità di avvicinarmi all'ambito della dimostrazione automatica, e a familiarizzarmi con la notazione del calcolo dei sequenti. Inoltre l'utilizzo di Prolog per l'implementazione mi ha permesso di esplorare diversi ambiti della programmazione logica, ad esempio le CFG e CLP.

Una delle problematiche principali si è rivelata essere la carenza senza esponenziali – un particolare tipo di connettivo della logica lineare; infatti la quasi totalità dei dataset di formule di logica lineare è costituito da traduzioni di teoremi intuizionisti o di reti di Petri, entrambi casi caratterizzati da un alto numero di esponenziali. Visto che il nostro obiettivo principale era di valutare il prover nel caso moltiplicativo, è stato necessario trovare fonti alternative di formule. A questo scopo si è modificato un generatore randomico di formule affinché producesse casi senza esponenziali.

Riferimenti bibliografici

[1] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.

[2] James Harland and David J. Pym. Resource-distribution via boolean constraints. *ACM Trans. Comput. Log.*, 4(1):56–90, 2003.

[3] Naoyuki Tamura. A linear logic prover (llprover). `https://cspsat.gitlab.io/llprover`. Accessed: 2024-03-20.

[4] Jui-Hsuan Wu. A linear logic prover implemented in ocaml. `https://github.com/wujuihsuan2016/LL_prover`. Accessed: 2024-03-20.