High level programming language for quantum computing

Davide Camino

2 febbraio 2026

Sommario

In questo lavoro esploriamo una pipeline di trasformazioni successive che permettono di programmare un quantum computer attraverso un linguaggio ad alto livello. Il lavoro nasce dalla volontà di fare inferenza su strutture ontologiche [**?**] sfruttando hardware quantistico. Il nostro lavoro parte da un progetto, ormai deprecato, che propone una pipeline per trasformare un programma Prolog in un istanza di problema risolvibile attraverso quantum annealing [**?**]. Ci siamo quindi concentrati nel ripristinare la compatibilità del progetto con i framework attualmente in uso, successivamente abbiamo esplorato la possibilità di sfruttare anche architetture basate su quantum gate grazie all'algoritmo QAOA [**?**]. Il lavoro termina con un esempio di una piccola base di conoscenza espressa in Prolog (sulla quale sia possibile fare inferenza) e alcune considerazioni sulla possibilità di trasformare un'ontologia espressa in OWL [**?**] in un'ontologia espressa in Prolog e compatibile quindi con hardware quantistico.

1 Quantum Computing

Il quantum computing sostituisce i classici bit con i *qubit*. I *qubit* trasportano la minima quantità di informazione possibile [**?**] e se misurati possono trovarsi esclusivamente in due stati: $|1\rangle$ o $|0\rangle$[1]. Durante l'elaborazione tuttavia i *qubit* possono essere in una sovrapposizione di stati che decadrà solo all'atto della misurazione.

Sfruttando la sovrapposizione degli stati e l'entanglement tra *qubit* si ritiene sia possibile violare la tesi di Church-Turing estesa[2] secondo cui qualsiasi modello computazionale possa essere simulato attraverso una macchina di turing deterministica con overhead al più polinomiale[**?**]. Si oterebbero così degli speedup più che polinomiali per problemi che ad oggi consideriamo non risolvibili con algoritmi efficienti[3].

[1] Esprimiamo gli stati dei *qubit* con le parentesi di Dirac.
[2] *Strong Church-Turing Thesis.*
[3] Un algoritmo è definito efficiente se ha complessità al più polinomiale.

1.1 Quantum Gate

Il paradigma di computazione quantistica basato su quantum gate è il più noto e studiato, le prime formalizzazione del paradigma e i primi algoritmi che sfruttano computazione quantistica basata su gate anticipano di molti anni la costruzione effettiva di computer quantistici e già Richard Feinman nel 1982 propone la computazione quantistica come alternativa ai modelli classici di computazione[**?**].

Come suggerisce il nome questo paradigma si basa su porte logiche, che agiscono sui *qubit* esattamente come le porte logiche classiche (es. **and**, **or** e **not**) agiscono sui bit classici.

Le porte logiche quantistiche (che da ora chiameremo quantum gate) modificano lo stato del *qubit* operando una trasformazione fisica, non possiamo quindi progettare un gate partendo da un'arbitraria tabella di verità come faremmo nel caso classico, ma dovremo sottostare alle leggi della meccanica quantistica. In particolare le seguenti affermazioni sono tutte equivalenti e definiscono le caratteristiche di un quantum gate:

- I quantum gate sono mappe lineari che mantengono lo stato normalizzato (le probabilità di misurare il o i *qubit* in ogni stato sommano a uno[4]);
- Tutti i gate classici reversibili (dato l'output si può risalire all'input) sono quantum gate validi;
- Le tavole di verità i cui output sono una permutazione degli input rappresentano quantum gate validi. [**?**]

I gate reversibili possono essere espressi come matrici unitarie. Una matrice è unitaria se la sua inversa coincide con la coniugata Hermitiana (trasposta dei complessi coniugati): $\mathbf{U} \times \mathbf{U}^\dagger = \mathbf{U}^\dagger \times \mathbf{U} = \mathbf{I}$.

Assemblando opportunamente quantum gate è possibile costruire circuiti logici che operano su *qubit*. Questi circuiti possono essere a tutti gli effetti interpretati come algoritmi, uno dei più famosi è l'algoritmo per la fattorizzazione di un numero proposto da Shore [**?**], questo algoritmo presenta uno speedup più che polinomiale rispetto al miglior algoritmo di fattorizzazione classico, e rafforza l'ipotesi che i quantum computer possano violare la tesi di Church-Turing estesa.

1.2 Quantum Annealing

Il quantum annealing è un approccio differente alla computazione quantistica e si concentra principalmente sulla risoluzione di problemi di ottimizzazione. Tipicamente le istanze non banali di questi problemi sono NP-Hard, di conseguenza anche questo paradigma mira a risolvere problemi che prevedono soluzioni classiche non efficienti.

[4] Se consideriamo n *qubit* nello stato $|A\rangle$ avremo 2^n esiti della misura, detta p_i la probabilità di ottenere l'esito i, $|A\rangle$ è normalizzato se $\sum_1^{2^n} p_i = 1$, cioè $\langle A \,|\, A\rangle = 1$

L'annealing (letteralmente ricottura), che sia classico o quantistico prevede l'esplorazione di uno spazio in cui ogni punto è associato ad un'energia, al fine di trovare il punto a energia minima.

L'ennealing classico[5] è un algoritmo probabilistico che permette di trovare il minimo globale di una funzione avete molti minimi locali[**?**]. Viene simulato il processo di ricottura di un solido, durante la fase di raffreddamento (se questa è sufficientemente lenta) le particelle del solido si dispongono nella configurazione di minima energia[**?**]. L'algoritmo sfrutta la temperatura per uscire dai minimi locali.

Il Quantum annealing sostituisce la temperatura con il tunneling quantistico, che permette di attraversare barriere di potenziale passando da un minimo a un altro senza dover superare la barriera, ma passandoci attraverso[**?**]. Questo permette di superare anche barriere di potenziale infinite, ma infinitamente strette (con il simulated annealing sarebbe necessaria temperatura infinita) [**?**].

Siccome si perde il concetto di temperatura che diminusce durante la simulazione il modo più corretto di riferirsi a questo approccio è adiabatic quantum computing (AQC). L'AQC sfrutta la tendenza di un sistema fisico a rimanere nella condizione di minima energia. Si parte con un spazio per il quale sia molto facile trovare il minimo di energia, descriviamo l'energia di quasto sistema con l'Hamiltoniana $\mathcal{H}_i$, avremo poi l'Hamiltoniana del nostro problema detta $\mathcal{H}_f$ il cui minimo di energia rappresenta la soluzione del problema. Infine avremo una funzione $s(t)$ che decresce da 1 a 0 e rappresenta il cammino di evoluzione adiabatica. L'algoritmo AQC è definito dall'Hamiltoniana in funzione del tempo $\mathcal{H}(t)$ che gradualmente passa da $\mathcal{H}_i$ a $\mathcal{H}_f$: $\mathcal{H}(t) = s(t)\mathcal{H}_i + (1 - s(t))\mathcal{H}_f$ [**?**].

Per esprimere $\mathcal{H}_i$ si usa una base ortogonale a quella usata per $\mathcal{H}_f$ in questo modo quando i *qubit* sono nel minimo specificato da $\mathcal{H}_i$ e passiamo alla base di $\mathcal{H}_f$ abbiamo una sovrapposizione equiprobabile di tutti gli stati [**?**]. Man mano che l'Hamiltoniana evolve da $\mathcal{H}_i$ a $\mathcal{H}_f$ aumenta sempre di più la probabilità di effettuare una misura e trovare i *qubit* nel ground state di $\mathcal{H}_f$.

D'ora in poi ci concentreremo solo più sul trovare la forma di $\mathcal{H}_f$ e daremo per scontato che sia facile trovare $\mathcal{H}_i$ e $s(t)$.

2 QA-Prolog

QA-Prolog [**?**] è un proof of concep sviluppato da Scott Pakin, che mostra come sia pssibile compilare un sottoinsieme del Prolog in un'Hamiltoniana compatibile con il paradigma AQC.

Il lavoro risale al 2017 ed è stato supportato fino al 2021. Nell'articolo originale il progetto viene testato sul quantum annealer a 1,095 *qubit* prodotto da D-Wave [**?**].

[5] Chiamato simulated annealing.

2.1 Pipeline

Il progetto consiste in una serie di trasformazioni che permettono di assemblare un programma Prolog e una query in un'Hamiltoniana il cui minimo corrisponde alla risposta della query.

La pipeline di trasformazione prevede 4 stadi:

1. Il programma Prolog viene trasformato in un programma Verilog, un linguaggio di descrizione hardware (HDL) [**?**];
2. Il programma Prolog viene sintetizzato in un circuito digitale fatto di porte logiche e ottimizzato da Yosys [**?**];
3. Il circuito logico viene trasformato in un'Hamiltoniana simbolica, che offre qualche astrazione (come macro, riferimenti simbolici ai *qubit*, ecc.) rispetto all'Hamiltoniana fisica $\mathcal{H}_f$. Questo step è implementato da edif2qmasm[**?**];
4. Hamiltoniana fisica $\mathcal{H}_f$ è assemblata da QMASM un quantum macro assembler [**?**].

L'unico pezzo software non sviluppato appositamente per questa pipeline è Yosys, tutti gli altri componenti sono stati sviluppati da Scott Pakin con l'obbiettivo di astrarre sempre di più dalle "istruzioni macchina" dei quantum annealer di D-Wave e offrire un linguaggio di alto livello per interagire con un quantum computer. Ognuno di questi componenti è stato presentato in un articolo (citato nell'elenco puntato qui sopra) che spiega come il componente è stato realizzato, quali sono le funzionalità e in che modo astrae rispetto alla definizione di un'Hamiltoniana fisica $\mathcal{H}_f$.

2.2 Risultato finale

QA-Prolog mette insieme tutta la pipeline, e seppure sia possibile esplorare le trasformazioni passo passo e usare le singole componenti della pipeline per percorrerne anche solo una parte, uno dei punti di forza di QA-Prolog è la possibilità di eseguire tutta la pipeline compilando un programma Prolog con annessa query, inviare la richiesta a un quantum annealer di D-Wave (o a un simulatore) e ottenere indietro i risultati presentati in modo chiaro e già organizzato.

Presentiamo a titolo di esempio un piccolo programma Prolog che può essere riassunto con: "il nemico del mio nemico è mio amico" (Figura 1).

Usiamo QA-prolog per inferire se esiste una coppia di amici. Per questo esempio non useremo un vero quantum computer ma un simulatore (è possibile selezionare il solver direttamente come parametro di QA-Prolog). Il comando che invocheremo sarà:

```
QA-Prolog --qmasm-args="--solver=tabu --postproc=opt"
↪  --query='friends(P1, P2).'  friends.pl
```

Stiamo specificando il solver (tabu search), la query, cioè cerchiamo di legare due variabili `P1` e `P2` che rispettino il predicato di essere amiche, infine

```
hates(alice, bob).
hates(bob, charlie).

enemies(P, Q) :- hates(P, Q).
enemies(P, Q) :- hates(Q, P).

friends(A, B) :-
        enemies(A, X),
        enemies(X, B),
        A \= B.
```

Figura 1: The enemy of my enemy is my friend.

```
P1 = alice
P2 = charlie

P1 = charlie
P2 = alice
```

Figura 2: Output

specifichiamo il nome del file: `friend.pl`. Il risultato che otterremo avrà un aspetto simile a quello mostrato in Figura 2. Possiamo notare che la coppia di amici viene individuata correttamente; inoltre, data la simmetria delle relazioni definite, otteniamo 2 risposte con i ruoli invertiti. Questo corrisponde a due minimi assoluti dell'Hamiltoniana che è stata genrata da QA-Prolog.

2.3 Update the original project

Il progetto è stato mantenuto fino al 2021

3 From QA-Prolog to Q-Prolog

TO DO

3.1 QAOA

TO DO

3.2 From H to Paoli operator

TO DO

3.3 Results

TO DO

4 A Quantum Ontology

TO DO

4.1 A small example

TO DO

4.2 Triple-RDF to Prolog

TO DO