High level programming language for quantum computing

Davide Camino

UNIVERSITÀ
DI TORINO

davide.camino@edu.unito.it

Obbiettivo del lavoro

Reasoning su quantum computer

1. Base di conoscenza classica + query
2. Embedding su quantum computer
3. Esecuzione e recupero risultati

Perché il quantum computing

Grazie a

- Superposition
- Entanglement

$$\mathcal{H}(t)\,|\psi(t)\rangle = i\hbar\frac{\partial}{\partial t}\,|\psi(t)\rangle$$

$$\langle A_0 \otimes B_0\rangle + \langle A_0 \otimes B_1\rangle + \langle A_1 \otimes B_0\rangle - \langle A_1 \otimes B_1\rangle = 2\sqrt{2}$$

Violiamo

Strong Church-Turing Thesis

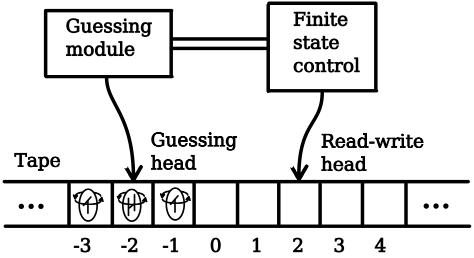


Figure: Probabilistic Turing Machine

Due paradigmi di programmazione

Quantum Gate

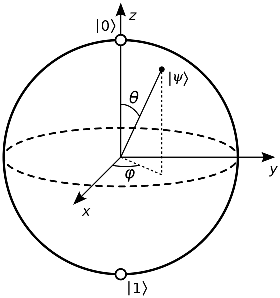


Figure: A single *qubit* gate can be seen as a rotation on the Bloch Sphere

Quantum Annealing

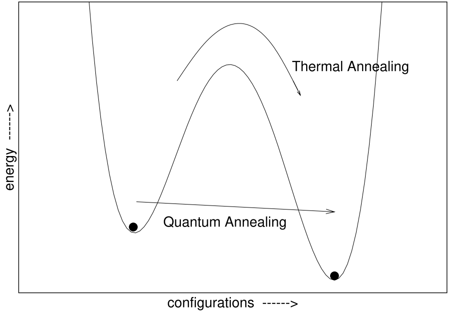


Figure: Tunneling effect (K. Chakrabarti, A. Das)

Quantum Gate

- Formalismo molto studiato
- Gate Reversibili
- Set di porte universali
- Algoritmi di Shore, ecc.

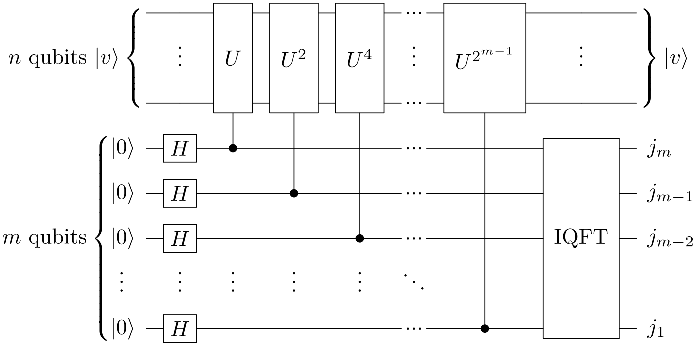


Figure: Step for Shor algoritm (Thomas G. Wong)

Quantum Annealer

immagine

- Problemi in forma QUBO
- Ispirato a Simulated Annealling
- Effetto tunneling
- Adiabatic Quantum Computing (AQC)
- $\mathcal{H}(t) = s(t)\mathcal{H}_i + (i - s(t))\mathcal{H}_f$

QA-Prolog

Il progetto

- Sviluppato da Sott Pakin
- Proof of concept
- Trasformazioni successive
- Prolog + Query $\rightarrow \mathcal{H}_f$
- Interagisce direttamente con il solver
- Raccoglie e organizza i risultati

Figure: Scott Pakin - Los Alamos National Laboratory

Pipeline

immagine

Trasformazioni

1. Prolog $\rightarrow$ Verilog (HDL)
2. Verilog $\rightarrow$ Circuito digitale
3. Circuito digitale $\rightarrow$ $\mathcal{H}_f$ simbolica
4. $\mathcal{H}_f$ simbolica $\rightarrow$ $\mathcal{H}_f$ fisica

QMASM

$\mathcal{H}_f$ simbolica $\rightarrow$ $\mathcal{H}_f$ fisica

Cos'è

- Quantum macro assembler
- Sviluppato in Python
- Basso livello di astrazione
- Si interfaccia con Ocean

Cosa permette di fare

- Riferimento simbolico a *qubit*
- *Qubit* "pinnati" a `TRUE` o `FALSE`
- Incapsulare pattern in macro
- Creazione di librerie di macro
- Pulizia dell'output:
  - solo *qubit* "interessanti"
  - no slack variables

QMASM

Esempio: Macro

```
# Y = A OR B
!begin_macro OR
  $A  0.5
  $B  0.5
  $Y -1

  $A $B  0.5
  $A $Y -1
  $B $Y -1
!end_macro OR
```

Figure: **or** gate

```
# Y = NOT A
!begin_macro NOT
  $A $Y 1.0
!end_macro NOT
```

Figure: **not** gate

```
# Y = A AND B
!begin_macro AND
  $A -0.5
  $B -0.5
  $Y  1

  $A $B  0.5
  $A $Y -1
  $B $Y -1
!end_macro AND
```

Figure: **and** gate

possiamo racchiudere queste macro nel file `gates.qmasm`

QMASM

Esempio: $y = x_1 \wedge \neg(x_2 \vee x_3)$

```
!include <gates>

!use_macro OR x2_or_x3
x2_or_x3.$A = x2
x2_or_x3.$B = x3
x2_or_x3.$Y = $x4

!use_macro NOT not_x4
not_x4.$A = $x4
not_x4.$Y = $x5

!use_macro AND x1_and_x5
x1_and_x5.$A = x1
x1_and_x5.$B = $x5
x1_and_x5.$Y = y
```

Figure: CircSat problem

"Pinniamo" il valore di y per ottenere l'assegnamento delle x_i che verificano la formula logica:

```
qmasm --run --pin="y := true" circsat.qmasm
```

```
Solution #1 (energy = -20.0000, tally = 647):

    Variable  Value
    --------  -----
    x1        True
    x2        False
    x3        False
    y         True
```

Figure: CircSat solution

Yosys - edif2qmasm

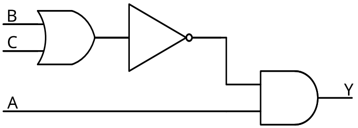
Verilog → Circuito digitale → $\mathcal{H}$ simbolica

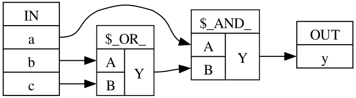
Yosys

- Framework per la sintesi del Verilog
- Free and open software sotto licenza ISC
- Output: RTL Netlist in formato EDIF

edif2qmasm

- Converte dal formato EDIF a QMASM
- Attinge a una libreria di gate






```
!begin_macro sat
  !use_macro AND $id00004
  !use_macro OR $id00003
  $id00003.A = b
  $id00003.B = c
  $id00004.A = a
  $id00004.B = $id00003.Y
  $id00004.Y = y
!end_macro sat
```

Yosys - edif2qmasm

Esempio: moltiplicazione tra interi

```
module mult (multiplicand, multiplier, product);
   input  [1:0] multiplicand;
   input  [1:0] multiplier;
   output [2:0] product;

   assign product = multiplicand * multiplier;
endmodule
```

Figure: Factorization problem

Traduzione in EDIF

```
yosys myfile.v synth.ys -b edif -o myfile.edif
```

Traduzione in QMASM

```
edif2qmasm -o="myfile.qmasm" myfile.edif
```

Esecuzione

```
qmasm --run --pin="mult.product[2:0] := 110"
      --solver="sim_anneal" mult.qmasm
```

```
Solution #1 (energy = -57.5000, tally = 68):

    Variable              Value
    --------------------  -----
    mult.multiplicand[0]  False
    mult.multiplicand[1]  True
    mult.multiplier[0]    True
    mult.multiplier[1]    True
    mult.product[0]       False
    mult.product[1]       True
    mult.product[2]       True
```

Figure: Factorization Solution

QA-Prolog

- Traduzione da Prolog a Verilog
- Wrapper per tutta la Pipeline
- Risultati in formato Human Readable
- Decide dimensione delle variabili

```
sat(A, B, C, Y) :-
    or(B, C, X),
    not(X, Z),
    and(A, Z, Y).
```

Figure: Prolog Code

```
module sat (a, b, c, y);
   input a, b, c;
   output y;

   assign y = a & ~(b | c);
endmodule
```

Figure: Verilog Code

QA-Prolog

Esempio

Base di conoscenza

Il nemico del mio nemico è mio amico

```
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.
```

Figure: Enemy of my Enemy

Esecuzione

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

```
P1 = alice
P2 = charlie

P1 = charlie
P2 = alice
```

Figure: Query Solution

Update al progetto

Integrazione con Ocean

- Aggiornamento librerie obsolete
- Rimozione metodi deprecati
- Sostituzione funzioni "spostate"
- Correzione parametri funzioni

Altro

- Sostituzione funzioni rinominate
- Parametri command line

```
#from dwave.cloud import Client, hybrid, qpu, sw
from dwave.cloud import Client
import hybrid

#from greedy import SteepestDescentSolver
from dwave.samplers import SteepestDescentSolver

#from tabu import TabuSampler
from dwave.samplers import TabuSampler

#from scipy.stats import median_absolute_deviation
from scipy.stats import median_abs_deviation
```

Figure: Update librerie

```
#elif solver == "neal":
elif solver == "sim_anneal":
```

Figure: Correzione parametri command line

Update al progetto

Incompatibilità tra output edif2qmasm e input QMASM

```
// Define hates(atom, atom).
module \hates/2 (A, B, Valid);
  ...
endmodule
```

Figure: Verilog

```
# hates/2
!begin_macro id00011
  ...
!end_macro id00011

# enemies/2
!begin_macro id00010
  !use_macro OR $id00031
  !use_macro hates/2
  ...
```

Figure: qmasm

```
with open(file, 'r') as input:
  first = input.readline()
  second = input.readline()
  while(second_row != ""):
    if first.startswith("#") and second.startswith("!begin_macro"):
      self.name[first[2:-1]] = second[len("!begin_macro "):-1]
    first_row = second_row
    second_row = input.readline()

with open(file, 'r') as input:
  doc = input.read()
  lines = doc.splitlines()
  for line in lines:
    for word in line.split():
      if word in self.name.keys():
        line = line.replace(word, self.name[word])
    self.new_lines.append(line)
```

Figure: Preprocessing

QAOA

Quantum approximate optimization algorithm

1. Definire matrici: $\mathcal{H}_c$ e $\mathcal{H}_m$
2. Definire oracoli parametrici: $\mathcal{U}_c(\gamma) = e^{-i\gamma\mathcal{H}_c}$ e $\mathcal{U}_m(\beta) = e^{-i\beta\mathcal{H}_m}$
3. Applicazione ripetuta di $\mathcal{U}_c(\gamma)$ e $\mathcal{U}_m(\gamma)$
4. Ottimizzazione (classica) dei parametri γ_i e β_i

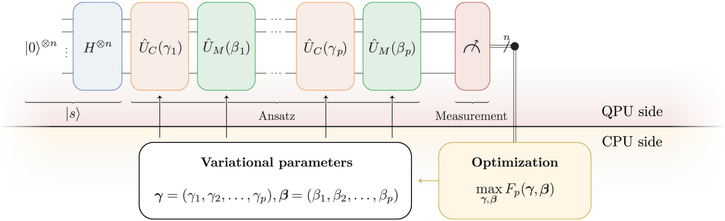


Figure: BLEKOS, Kostas, et al. A review on quantum approximate optimization algorithm and its variants

QUBO → ISING → Operatori di Pauli

```
qubo = qubovert.utils.matrix_to_qubo(qubo_mat)
ising = qubovert.utils.qubo_to_quso(qubo)
qubo_dict = dict(qubo)
ising_dict = dict(ising)
ising_dict.popitem()
ising_mat = qubovert.utils.qubo_to_matrix(ising_dict)
```

Figure: QUBO → ISING (libreria qubovert)

```
def build_paulis(matrix):
  pauli_list = []
  for i in range(len(matrix)):
    pauli_list.append(("Z", [i], matrix[i][i]))
    for j in range(i+1, len(matrix)):
        pauli_list.append(("ZZ", [i, j], matrix[i][j]))
  return pauli_list

sat_paulis = build_paulis(ising_mat)
cost_hamiltonian = SparsePauliOp.from_sparse_list(sat_paulis, n_qubits)
```

Figure: ISING → Operatori di Pauli

Esempio

Esempio

Risultato 1

Esempio

Risultato 2

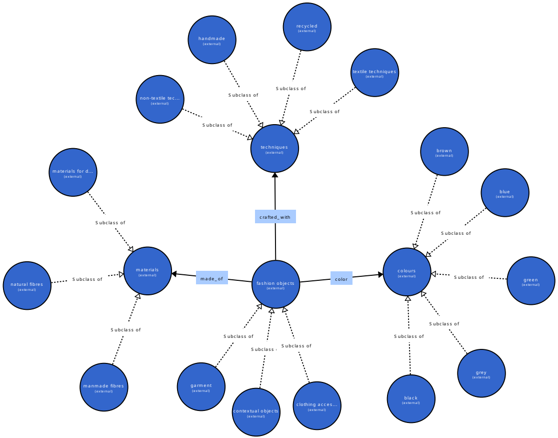
Ontologie

Cos'è un'ontologia

- Insieme di fatti
- Riguardanti un dominio di interesse
- Prevengono interpretazioni sbagliate
- Assicurano la cooperazione tra software

Come si definisce un'ontologia

- Ontology Web Language (OWL)
- Diversi "flavours"
  - OWL Full
  - OWL DL
- Classi Individui e Relazioni



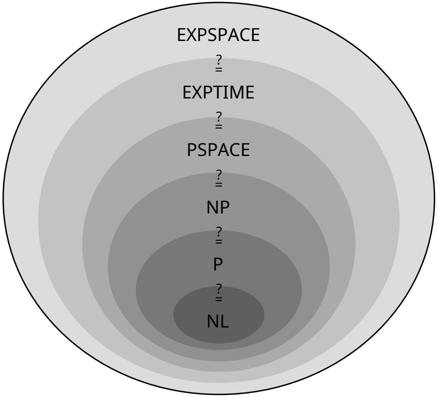
Inferenze in OWL

Complessità

- Variazione sintattica di $\mathcal{SROIQ}$
- $\mathcal{ALC}$ è una restrizione di $\mathcal{SROIQ}$
- $\mathcal{ALC}$ è PSpace-hard

Reasoner

- Pellet
- Fact++
- HermiT
- ...



Esempio completo

Quantum Simpson

```
<owl:NamedIndividual rdf:about="http://www.people#marge">
    <rdf:type rdf:resource="http://www.people#People"/>
    <www:marry rdf:resource="http://www.people#homer"/>
    <www:parent_of rdf:resource="http://www.people#bart"/>
</owl:NamedIndividual>
```

Figure: Marge Simpson entry

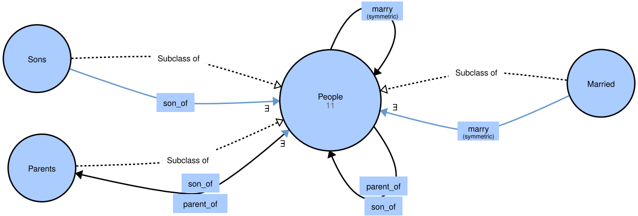


Figure: Ontology Structure

Da OWL-rdf a Prolog

```
person(marge).
person(bart).
person(jackie).
person(selma).
person(ling).

parent_of(marge, bart).
parent_of(jackie, marge).
parent_of(jackie, selma).
parent_of(selma, ling).

son_of(P, Q) :- parent_of(Q, P).

gran_parent_of(P, Q) :-
  son_of(Q, X),
  son_of(X, P).
```

Figure: Prolog Ontology

Interrogazione

```
QA-Prolog --verbose --qmasm-args="--solver=sim_anneal
--postproc=opt" --query='gran_parent_of(P1, P2).'  family.pl
```

```
P1 = jackie
P2 = bart

P1 = jackie
P2 = selma
```

Figure: Query Result 1

```
P1 = jackie
P2 = bart
```

Figure: Query Result 2

Limitazioni del simulatore

```
cousins(P, Q):-
  gran_parent_of(X, P),
  gran_parent_of(X, Q),
  parent_of(Y, P),
  parent_of(Z, Q),
  Z \= Y.
```

Figure: Cousins rule

Interrogazione

```
QA-Prolog --verbose --qmasm-args="--solver=sim_anneal
--postproc=opt" --query='cousins(P1, P2).' family.pl
```

```
QA-Prolog: No solutions were found
```

Figure: Query Result