Why Database Languages are Simpler than Turing Complete

David Maier

Maseeh Professor of Emerging Technologies
Portland State University
Shaw Visiting Professor
National University of Singapore

Portland State UNIVERSITY

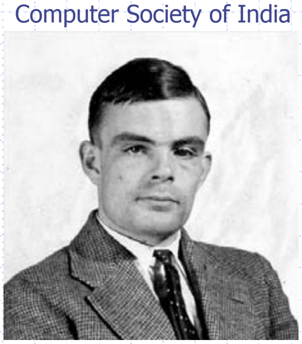
NUS National University of Singapore

Why this Talk?

2012 The Alan Turing Year

A Centenary Celebration of the Life and Works of Alan Turing

Computer Society of India



Alan Mathison Turing

b. 23 June 1912

Mathematician
Logician
Cryptanalyst
Computer Scientist

How I Chose my Grad School

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

The Graduate College,
Princeton University,
New Jersey, U.S.A.

The paper that defined Turing Machines!

Proc. Lond. Math. Soc. (2) 42 pp 230-265 (1936).

Why this Topic?

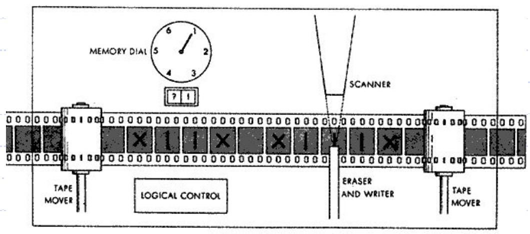
Alan M. Turing - Simplification in Intelligent Computing Theory and Algorithms

- Turing defined a standard for computational expressiveness
- But database languages don't come up to that standard – they're simpler

Why?

Turing Computable

Realizable by a Turing Machine (TM)



Turing Complete computation model:

Can express any function a TM can

Church-Turing Thesis

Effectively calculable = Turing computable

Turing Machines

Lambda Calculus

General μ-Recursive Functions

Semi-Thue Systems

Post Tag Systems

Type 0 Languages

Combinatory Logic

HTML + CSS

Minecraft

Magic – The Gathering

Game of Life

Two rocks and a roll of toilet paper

…

… And Just About Every PL

Pascal, Smalltalk-80, LISP, Forth, APL, Algol-60, Fortran, COBOL, Java, PHP, Python, Erlang, Haskell, ML, PL/I, 360 Assembler, BCPL, C, C++, C#, Objective C, Perl, SNOBOL, Prolog, Ruby, Scheme, Logo, BASIC, Modula-2, Ada, New S, R, Matlab, Simula, Eiffel, Occam, …

But why not database languages?

SQL, Quel, QBE, FQL, …

First, Consider a Very Simple Language: Regular Expressions

REs can express families of strings

- Binary Numbers: `1(0*1)*0* + 0`
- `1011`, `111`, `0`, `1000`, but not `00`, `0001`

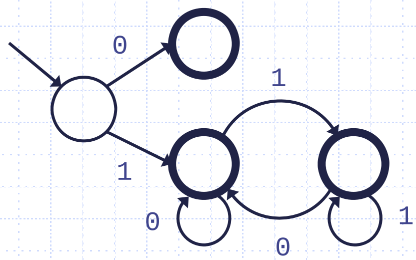
Not Turing Complete by a long shot

- (*How is this even a function?*)
- (*Think of it mapping String to Boolean.*)

Has a Corresponding Computer

Finite State Machines (FSMs)

- Can automatically translate to FSMs from regular expressions
- Always halt (on finite inputs)
- Efficient: Constant work per input symbol



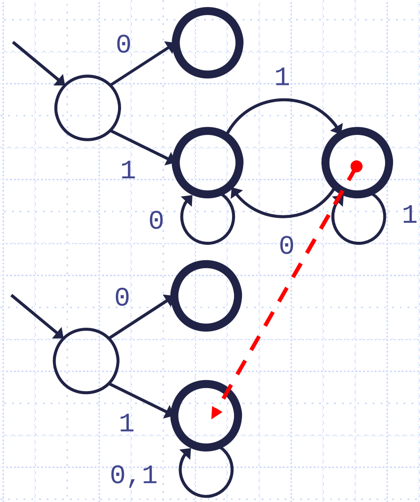
Many Nice Properties

- Can decide if an RE expresses all strings over an alphabet
- Can decide if two REs have a string in common
- Can decide if two REs express the same set of strings

(None of these are possible for Turing Complete models.)

Can Minimize FSMs

- Always a unique equivalent FSM with fewest states



Succinct Over Its Domain

- REs are *declarative*: <u>what</u>, not <u>how</u>
- Most PLs take much more space to express the same function (unless they have REs built in)
- But there are things you can't express
  - Strings of the form `(`b`:`c`)` where $b < c$
  - `(100:1011)`

*

Similar for Data Languages → Like SQL.

- Declarative, but translatable to procedural model (Relational Algebra)
- Finite answers on finite input
- Can decide equivalence (for subsets)
- Can optimize (find faster, equivalent programs)

Ease of expression, efficiency over limited domain

Relational Model (E. F. Codd)

Tables

```
stalls(Name    Loc     Number  Certif)
       Roxy    Lagoon    48     'B'
       Hougang Lagoon    33     'A'
       Mamu    Bedok     24     'B'

serves(Name    Dish   Price)
       Roxy    Laksa  $4.00
       Roxy    Otah   $0.75
       Hougang Otah   $0.50
```

Relational Query Languages

Basically First-Order Logic

Which stalls serve both laksa and otah?

Tuple Calculus (***Note:*** *shorthand form*)

```
{t.Name | t∊stalls ∧ s∊serves ∧ u∊serves ∧
  t.Name = s.Name ∧ t.Name = u.Name ∧
  s.Dish = 'laksa' ∧ u.Dish = 'otah'}
```

SQL

```
select st.Name
from stalls st, serves sv1, serves sv2
where st.Name = sv1.Name and
 st.Name = sv2.Name and sv1.Dish = 'laksa'
 and sv2.Dish = 'otah'
```


Computer: Relational Algebra

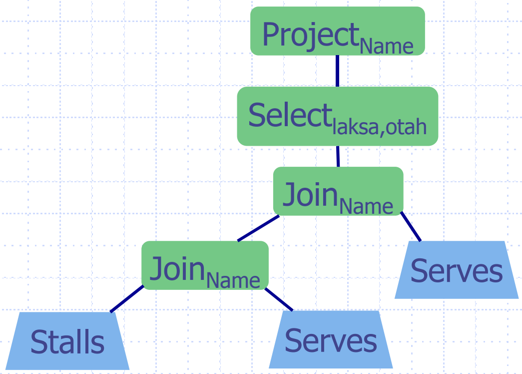
Small set of operations on relations

- SELECT: subset of rows
- PROJECT: subset of columns
- JOIN: combine on equal values
- UNION, INTERSECT, DIFFERENCE
- DIVISION: "for all"

Each one is finite in, finite out. Thus any composition has this property.

Can Translate Relational Queries

```
{t.Name | t∈stalls ∧ s∈serves ∧ u∈serves ∧
  t.Name = s.Name ∧ t.Name = u.Name ∧
  s.Dish = 'laksa' ∧ u.Dish = 'otah'}
```

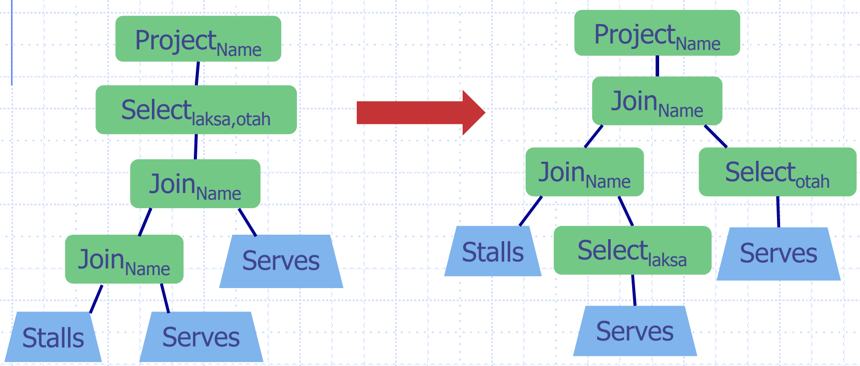


Containment, Identity

- Can decide if one query always returns a subset of another
  - For SELECT, PROJECT, JOIN, UNION
  - Containment both ways: Equivalence
- Have algebraic identities
  - Can analyze all combinations of operators
  - UNION(JOIN(r, s), JOIN(r, u)) =
    JOIN(r, UNION(s, u))

Query Optimization

- Explore equivalent expressions to minimize expected evaluation time
  - As opposed to minimizing expression size



stingy

Parsimony of Expression

- Wouldn't use SQL to do weather simulation
- But it can express succinctly much of what you want to do with bulk processing of data
  - And has properties not shared by general-purpose programming languages

Extensions Beyond FOL

- Aggregation: SUM, COUNT, AVERAGE
- Duplicates: multiset operators
- Nested queries

Most of the framework stays intact finite in, finite out
  - Add or modify a few algebraic operators
  - Algebraic identities might change

But, Too Simple for *Recursion*

It was noted early on that standard relational languages couldn't express certain queries.

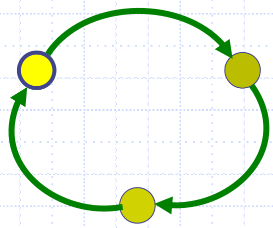
For example, transitive closure of a graph

No general iteration structure – looping is encapsulated in operators

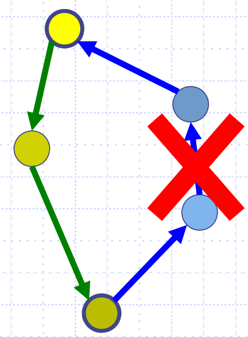
Example Recursive Queries

`better(stall1, stall2, dish)`

Cycle



Dominates



Danger!

Can easily lose finiteness

```
combo(Roxy, [laksa,otah], $4.75)
serves(Roxy, otah, $0.75)
```

$\Rightarrow$

```
combo(Roxy, [laksa,otah,otah],
                          $5.50)
```

How to Proceed?

Limit recursion by data:
finite data gives finite computation

- Traversal recursion: data gives call structure

```
TMax(leaf(V)) = V
TMax(node(L,R)) = max(TMax(L),TMax(R))
```

Recursion follows structure of data
↳ will stop.

- Safety: Values in results come from the database (or the query)

  Only a finite number of ways to combine

Datalog

Simplified Prolog: no function symbols

Alternative execution strategies

```
{t.Name | t∊stalls ∧ s∊serves ∧ u∊serves ∧
 t.Name = s.Name ∧ t.Name = u.Name ∧
 s.Dish = 'laksa' ∧ u.Dish = 'otah'}

result(N) :-
   stalls(N, L1, S1, C1),
   serves(N, laksa, P2),
   serves(N, otah, P3).
```

Generate Answers from Instances

```
result(N) :-
   stalls(N, L1, S1, C1),
   serves(N, laksa, P2),
   serves(N, otah, P3).

result(roxy) :-
   stalls(roxy, lagoon, 48, 'B'),
   serves(roxy, laksa, 4.00),
   serves(roxy, otah, 0.75).
```

~~result(hougang)~~ :-
   stalls(hougang, lagoon, 33, 'A'),
   ~~serves(hougang, laksa, *)~~,
   serves(hougang, otah, 0.50).

Recursion via Self-Reference

```
chain(S1,S2,D) :- better(S1,S2,D).
chain(S1,S3,D) :- chain(S1,S2,D),
                  better(S2,S3,D).

cycle(S) :- chain(S,S,D).

dominates(S1,S2) :- chain(S1,S2,D1),
                    ¬chain(S2,S1,D2).
```

Evaluation: Generate Till No Change

```
better(Stall1, Stall2, Dish)
       roxy    hougang otah
       hougang mamu    otah
       mamu    roxy    laksa

chain(roxy, hougang, otah) :-
           better(roxy, hougang, otah).
chain(mamu, roxy, laksa) :-
           better(mamu, roxy, laksa).
chain(roxy, mamu, otah) :-
           chain(roxy, hougang , otah),
           better(hougang, mamu, otah).
```


Retain Many Properties

- Optimization opportunities

```
roxyBetter(S) :- chain(roxy,S,D).
chain(roxy,S3,D) :-
          chain(roxy,S2,D),
          better(S1,S2,D).
```

- Alternative evaluation strategies
  - Semi-Naïve
  - Extension tables
  - Query-Subquery

Did We Lose Anything?

Not for *Monotone* programs:
Bigger database ⇒ Bigger answer

Apply rules in any order until no change

- Always get the same result
- Minimum model = least fixpoint

Negation Isn't Montone

Evaluation is order dependent

```
chain(roxy, hougang, otah) :-
           better(roxy, hougang, otah).
chain(mamu, roxy, laksa) :-
           better(mamu, roxy, laksa).
dominates(mamu, roxy) :-
           chain(mamu, roxy, laksa),
           ¬chain(roxy, mamu, *).
chain(roxy, mamu, otah) :-
           chain(roxy, hougang , otah),
           better(hougang, mamu, otah).
```

How to Handle Negation

Restrict to *stratified* programs
- No recursion through negation
- Fully evaluate recursive relation before negating it
- Stable model: analog of minimum model

Need similar care with aggregation

So, we have lost something because of added expressiveness

Datalog Sounds Wonderful

So why did it disappear in the 90's …

Ullman: Not many *large-scale* recursive apps

Vardi: Entrenched orthodoxy of RDBMS

Aref: Hostile system types, limited implementations

Hellerstein: Dry mode of discourse

Ramakrishan: No *unserved* killer instances

Abiteboul: What, Datalog went away?

Limited Recursion in SQL

- Oracle CONNECTS TO clause for hierarchical data
- WITH clause for linear recursion

*

```
with chain(stall1,stall2,dish) as
(select * from better
 union all
 select c.stall1, b.stall2, b.dish
 from chain c, better b
 where c.stall2 = b.stall1 and
       c.dish = b.dish)
```


Recent Resurgence

In research:

- BOOM project at Berkeley – avoiding coordination in distributed protocols
- Webdam at INRIA – formal foundations of interacting web applications
- Data exchange – IBM, U Pennsylvania
- Program analysis: DOOP, Codequest, PQL

And in Companies

- Lixto: Web extraction of pricing data
- Semmle: Software analytics
- LogicBlox: Big Data apps for enterprises
- Datomic: Temporal data service
- DLVSYSTEM: Knowledge-intensive apps

Many use it as an "internal" language

To Conclude

Turing Completeness is still the benchmark for expressiveness …

… but sometimes being simpler has its advantages.

To investigate further:

- Bently on "Little Languages"
- Domain-specific languages (DSLs)

Questions? Enigmas?