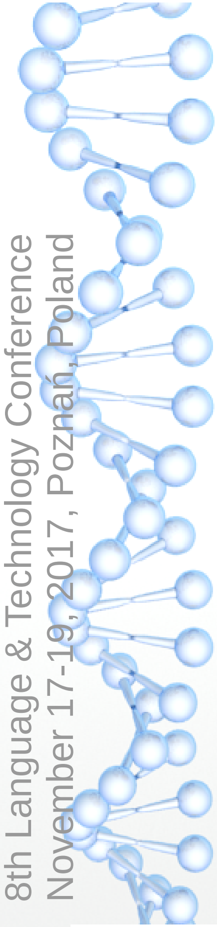
A second life for Prolog

Algorithm = Logic + Control

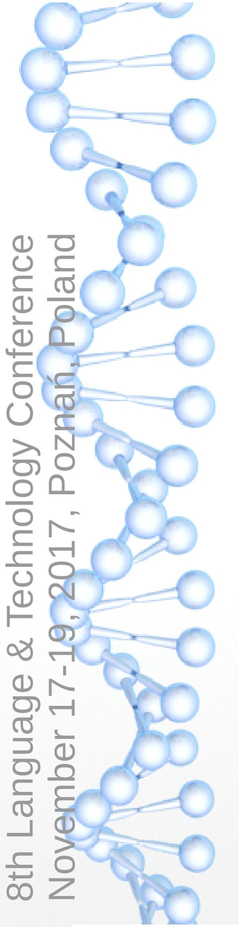
Jan Wielemaker
J.Wielemaker@cwi.nl

8th Language & Technology Conference
November 17-19, 2017, Poznań, Poland



Overview

- *Algorithm = Logic + Control*
- Limitations of SLD
- Beyond SLD



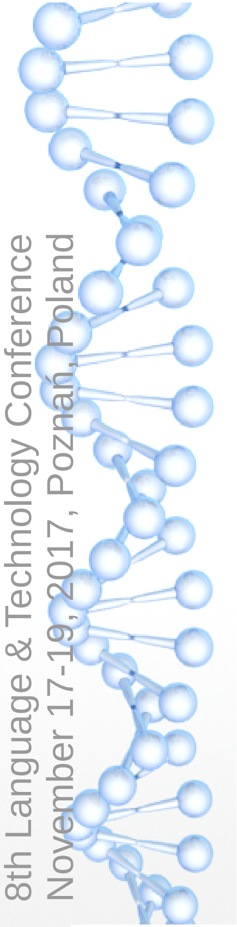
Algorithm = Logic + Control

Bob Kowalski - 1979

- In Logic programming we only specify the **logic**
- For classic Prolog the Control is „SLD resolution“
  - ✔ Defined execution order gives procedural reading
  - ✗ Depth-first search is sensitive to non-termination
  - ✗ Exploration order of the search space has huge impact on performance
  - ✗ Wrong backtracking order leads to frequent recomputation
- See https://swish.swi-prolog.org/p/ltc_underground.swinb

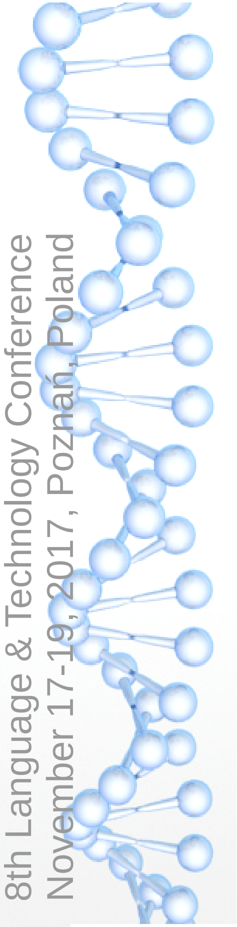
What now?

- Two directions
  - Live with it, exploit the good stuff classical Prolog brings
    - This is **tomorrow's** central topic
  - Aspect programming: bring control under the control of the user
    - This is **today's** central topic



SLG Resolution (tabling)

- Memoize the results of old queries and their answers
  - Avoids recomputation
- Explore other paths first if a variant of the current query is encountered
  - Avoids non-termination
- In practice acts as a lazy form of bottom-up evaluation.



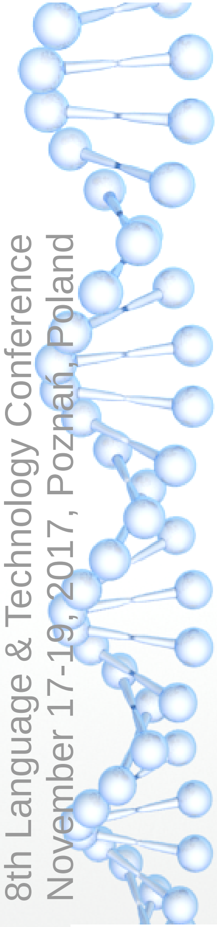
Avoid recomputation using tabling

```
:- table fib/2.

fib(0, 1) :- !.
fib(1, 1) :- !.
fib(N, F) :-
        N > 1,
        N1 is N-1,
        N2 is N-2,
        fib(N1, F1),
        fib(N2, F2),
        F is F1+F2.
```

- https://swish.swi-prolog.org/p/ltc_fibonacci.swinb

Avoid non-termination on left-recursion

```
:- table connected/2

% connections go both ways
connected(A, B) :- connected(B, A).
% and connections are transitive
connected(Start, End) :-
    connected(Start, Somewhere),
    connected(Somewhere, End).
```



SLG Resolution is the answer?

✔Guaranteed termination for finite data structures

✔No recomputation

✗ Potentially large memory footprint

✗ Hard to predict execution order → no procedural reading

→ For relatively small, but combinatorially hard problems

→ Small? CYC, uses F-logic on top of XSB tabling!

Constraints

- Constrain the permissible values of a variable by
  1. Adding data (attributes) to a variable
  2. Call a predicate if the variable is unified with a concrete value or another constraint variable
- Uses domain knowledge to reduce backtracking, i.e. given X in S1, Y in S2, after X=Y X(Y) is in the intersection of S1 and S2.
  - Traditional: member(X, S1), member(Y, S2) → O(N$^{2)}$
  - Constraint: use interval (O(1)) or ordered set (O(N))

Example

```
  S E N D +
  M O R E =
M O N E Y
```

- 8 digits → 10^8 (100,000,000) combinations
  - Naive: too costly
  - Merge tests and computation into generator: 3.8 sec
  - clp(fd): 0.001 sec.

- https://swish.swi-prolog.org/p/ltc_send_more_money.swinb

Constraints are the holy grail?

- ✔Compact description of problems
- ✔Efficient exploration of the search space
- ✗ Development of a solver requires domain knowledge
- ✗ Development of a solver is very complex
- ✗ We lost control: great if it works, but if it doesn't it is hard to find out why and how to fix it

Tor: lightweight custom search methods for Prolog

- Make choice-points (clause or ;/2) explicit and *hook* them
- Control order of exploration using the hooks
  - Iterative deepending
    - ?- queens(Vars), search(**id**(label(Vars))).
  - Limited discrepancy search
    - ?- queens(Vars), search(**lds**(label(Vars))).
  - Search with 50 credits and switch to bounded-backtrack search (1 backtrack allowed) when the credits are exhausted
    - ?- queens(Vars), search(**credit(50,bbs(1)**,label(Vars))).
- http://tomschrijvers.blogspot.nl/2012/03/tor-lightweight-custom-search-methods.html

Probabilistic Logic Programming

- The real world often needs *maybe*!
- Annotate facts with probabilities
- Scenarios
  - Create a logic program and learn the probabilities from data
  - Compute the probability of an answer based on the probabilities of all explanations
  - Find the most probable answer
  - ...
- See http://cplint.lamping.unife.it/

Coroutines

- *Traditionally* these were the hooks called from unifying annotated (attributed) variables for constraints.
- Recent
  - *Continuations* (SWI) are inherited from functional programming:
    - Capture the ‚remainder‘ of the computation (stack)
    - Do something else, to resume the captured continuation later
  - *Interactors (SWI, Lean Prolog)* are Prolog inference engines you can control from Prolog

Data in Prolog

- Modern Prolog systems allow for predicates with many clauses.   E.g.
  - ?- logrecord(A,B,C,D,E,F,G,H,I,J)
    - A = 6,
    - B = 'P101_u_ex1510.log.gz',
    - C = 1443657696.0,
    - D = get,
    - E = "/nl/pres/view/cite",
    - F = "identifier=ddd%3A010132734%3Ampeg21%3Aa0031&coll=ddd&query=plooij",
    - G = a48cde2180406905aefac97f2899f588,
    - H = "Mozilla/5.0+(Windows+NT+6.1;+WOW64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/45.0.2454.101+Safari/537.36",
    - I = http://www.delpher.nl/nl/kranten/view?query=plooij&facets%5Bspatial%5D%5B%5D=Nederlands-Indi%C3%AB+%7C+Indonesi%C3%AB&page=2&coll=ddd&identifier=ddd%3A010132734%3Ampeg21%3Aa0031&resultsidentifier=ddd%3A010132734%3Ampeg21%3Aa0031
    - J = 200
  - Stats: 6,573,723 clauses, 3,822,297,600 bytes

Example 2: Princeton Wordnet 3.0

- Load time: 32 sec, size 200Mb
- After precompilation (qcompile/1): load time: 1.0 sec

Clause indexing 1.0

- Instead of trying clauses one-by-one, Prolog examines the first argument.
- If this is bound (nonvar) it uses an index (list or hash table) that gives direct access to the candidate clauses.
  - ✔Speeds up finding the right clause
  - ✔Determine there are no more candidates, so we do not need to create a choicepoint.

Clause indexing 2.0

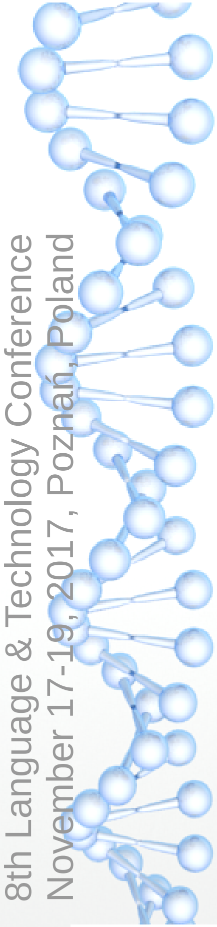
- Pioneered by YAP, now also in SWI and Jekejek
- JITI: Just In Time Indexing
  - If a good index for call is available, use it
  - Otherwise, see whether a good index can be created
    - If so, create it
    - Otherwise mark we tried
  - ✔ Provides indexes on any argument, not just the first
  - ✔ Provides combined argument indexes
  - ✔ Index into term arguments (planned for SWI-Prolog)

Lazy evaluation

- Create a partially instantiated term with attributed variables were it needs to be lazily extended.
- Combine attributed-variable unification hook and non-backtrackable assignment in terms to extend the term as it is accessed.
- library(lazy_lists) turns any input for which we can do a get/read operation into a lazy list.
  - Process infinite input with bounded resources
  - https://swish.swi-prolog.org/p/ltc_lazy_list.swinb

Take home

- SLD resolution allows programming in Prolog, but has limited inference power
- SLG, Constraints and Tor bring alternative inference strategies to Prolog
- Attributed variables, global variables, non-backtrackable assignments, continuations and interactors allow implementing alternative control regimes
- Probabilistic logic programming connects to machine learning
- Modern Prolog systems can efficiently handle large amounts of data



8th Language & Technology Conference
November 17-19, 2017, Poznań, Poland