Adapting the Rete-Algorithm to Evaluate F-Logic Rules

Florian Schmedding
Nour Sawas
Georg Lausen

Databases and Information Systems Research Group
Albert-Ludwigs-University Freiburg

Overview

1. Motivation
2. F-Logic
3. The Rete-Algorithm
4. Benchmarks
5. Results
6. Conclusions

Motivation

- In deductive databases rules are used to derive new facts from known data.
- These rules may contain redundancies, like
  ```
  p(X) :- a(X,Y), b(X), c(Y).
  q(X) :- a(X,Y), b(X), d(Y,Z).
  ```
- To reduce redundant deriving of facts, the Rete-Algorithm is used in production rule systems
- But: How about its performace when it deals with few rules and a lot of data?

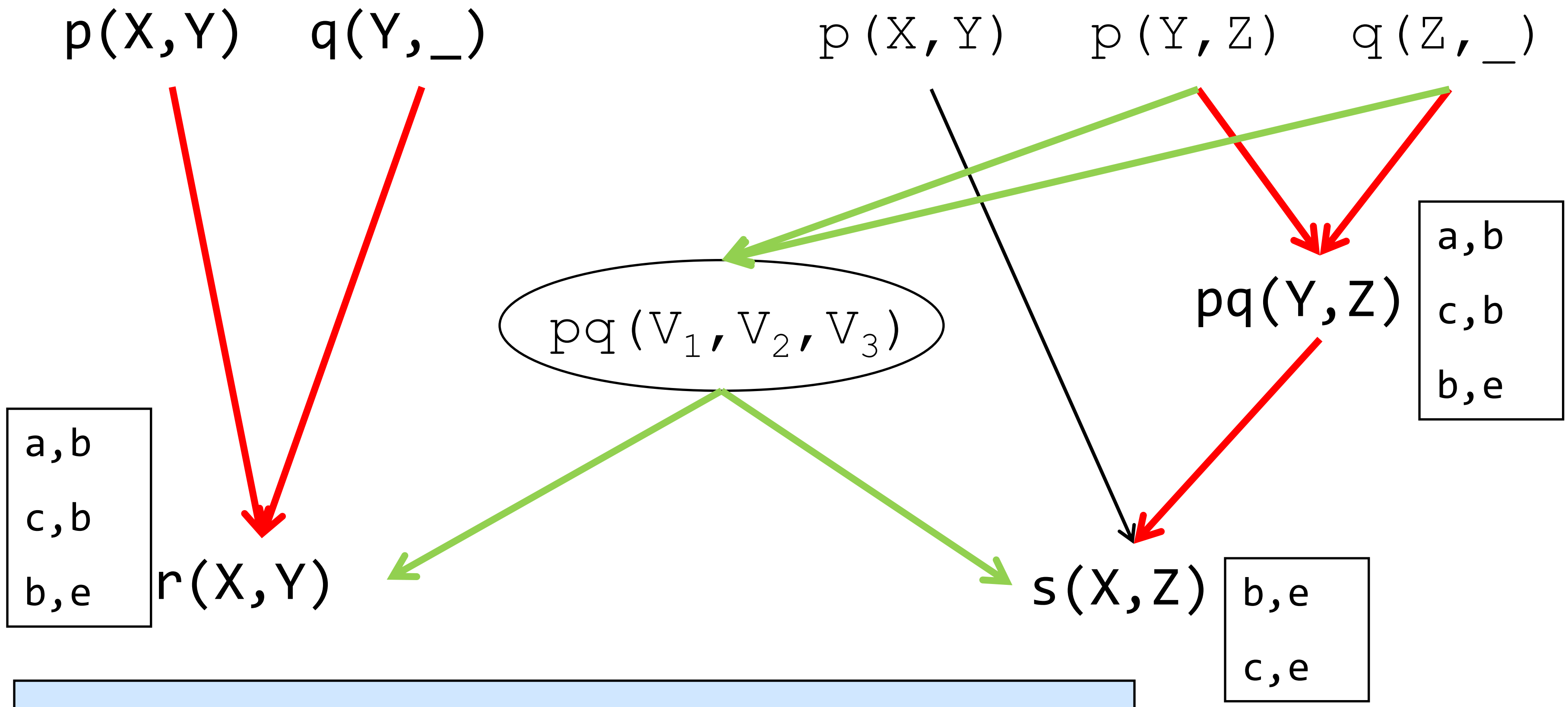
F-logic

- Declarative language
- Combines deductive databases with object-orientation
- First-order semantics, but can express sets and class hierarchy
- Covers Datalog
- Our implementation is ‘Florid‘
  - it has bottom-up fixpoint semantics
  - and seminaive evaluation of predicates

The Rete-Algorithm

- was developed by Ch. Forgy in 1974 for production rule systems
- its target is to identify common subgoals in rule bodies
  - this is done by constructing a network over all subgoals
  - which in turn is used to share matches in different rules
- 'Rete' is the Latin word for net

Simplified example

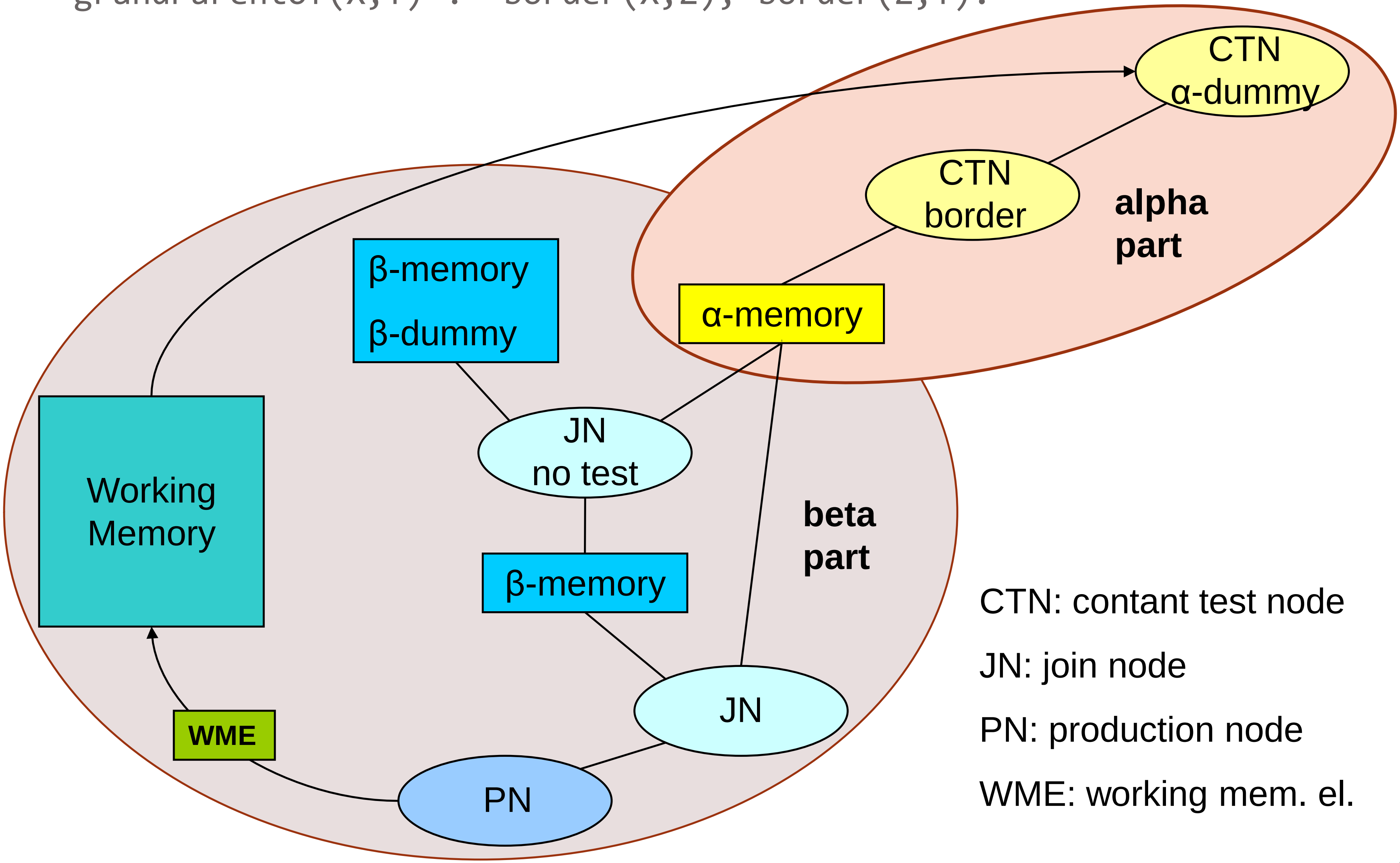


Program:

```
p(a,b).
p(c,b).  q(b,f).   r(X,Y) :- p(X,Y), q(Y,_).
p(d,e).  q(e,g).   s(X,Z) :- p(X,Y), p(Y,Z), q(Z,_).
```

Rete network for:

`grandParentOf(X,Y) :- border(X,Z), border(Z,Y).`



Adaption to F-Logic

- In Florid, we want to compute all results
- Immediate effects of new facts are not considered
- Therefore we compute all new facts before passing them down in the network (similar to stratification)
- Make use of index structures
- Re-use production nodes as beta memories

Benchmarks

- To compare rete with semi-naive evaluation, we wrote a non-recursive test program
  - in a short optimized version, e.g.

```
grandParentOf(X,Y)      :- border(X,Z), border(Z,Y).
grandGrandParentOf(X,Y) :- grandParentOf(X,Z),
                           border(Z,Y).
```

  - and in a long version with redundancies, e.g.

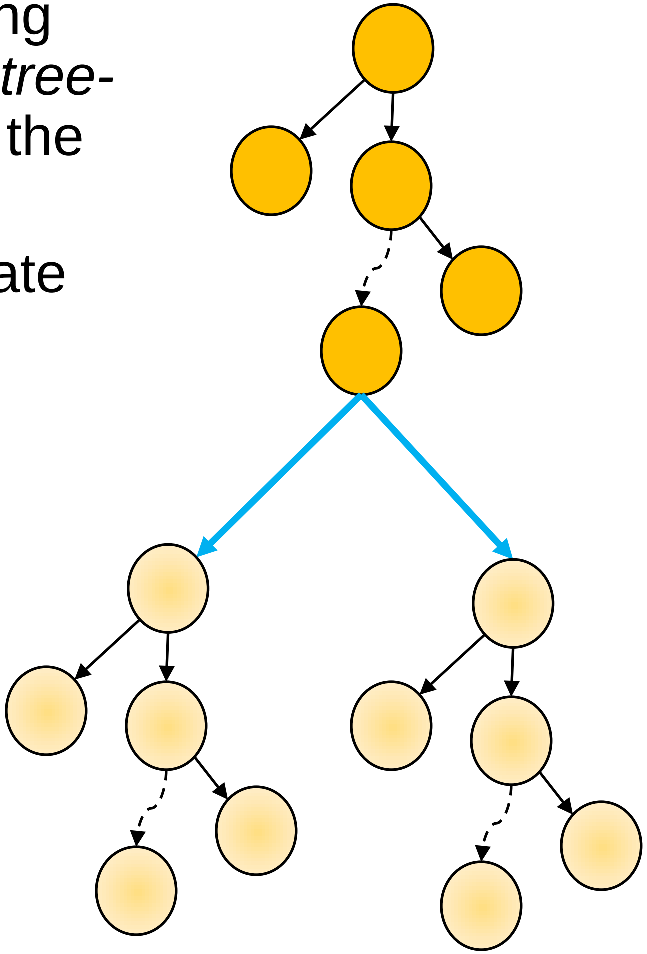
```
grandParentOf(X,Y)      :- border(X,Z), border(Z,Y).
grandGrandParentOf(X,Y) :- border(X,Z), border(Z,U),
                            border(U,Y).
```

- As input we used increasing numbers of the predicate border/2.

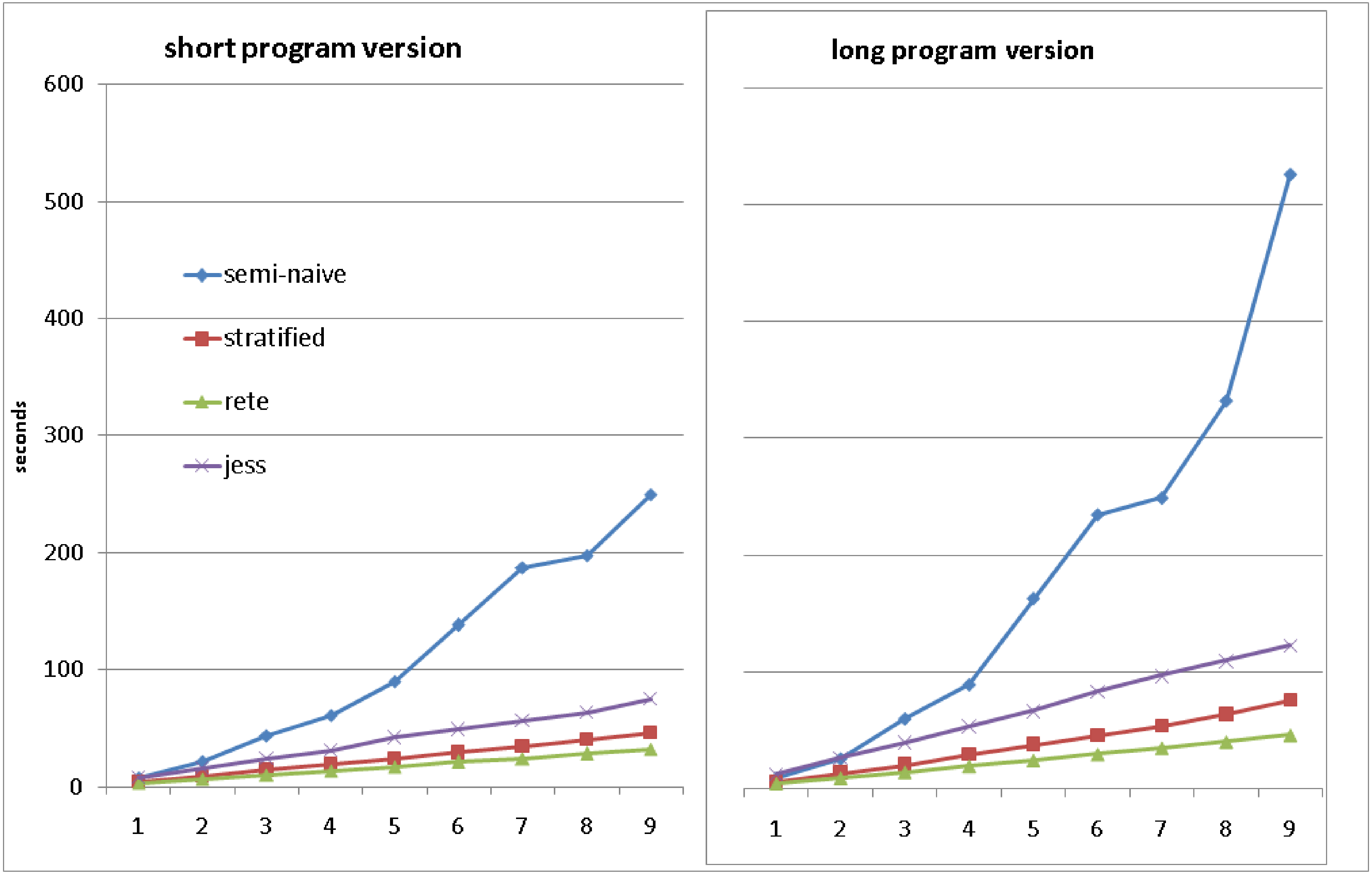
Input EDB

- We extended the data by duplicating the base graph and built a *“binary-tree-style” structure* by connecting with the **blue arrows**.
- We repeated this procedure to create data sets of increasing depth:

Graph	# facts	# results
1	10088	338319
2	20000	671323
3	30089	1010056
4	40001	1343060
5	50090	1682228
6	60002	2015232
7	70091	2353965
8	80003	2686969
9	90092	3026137

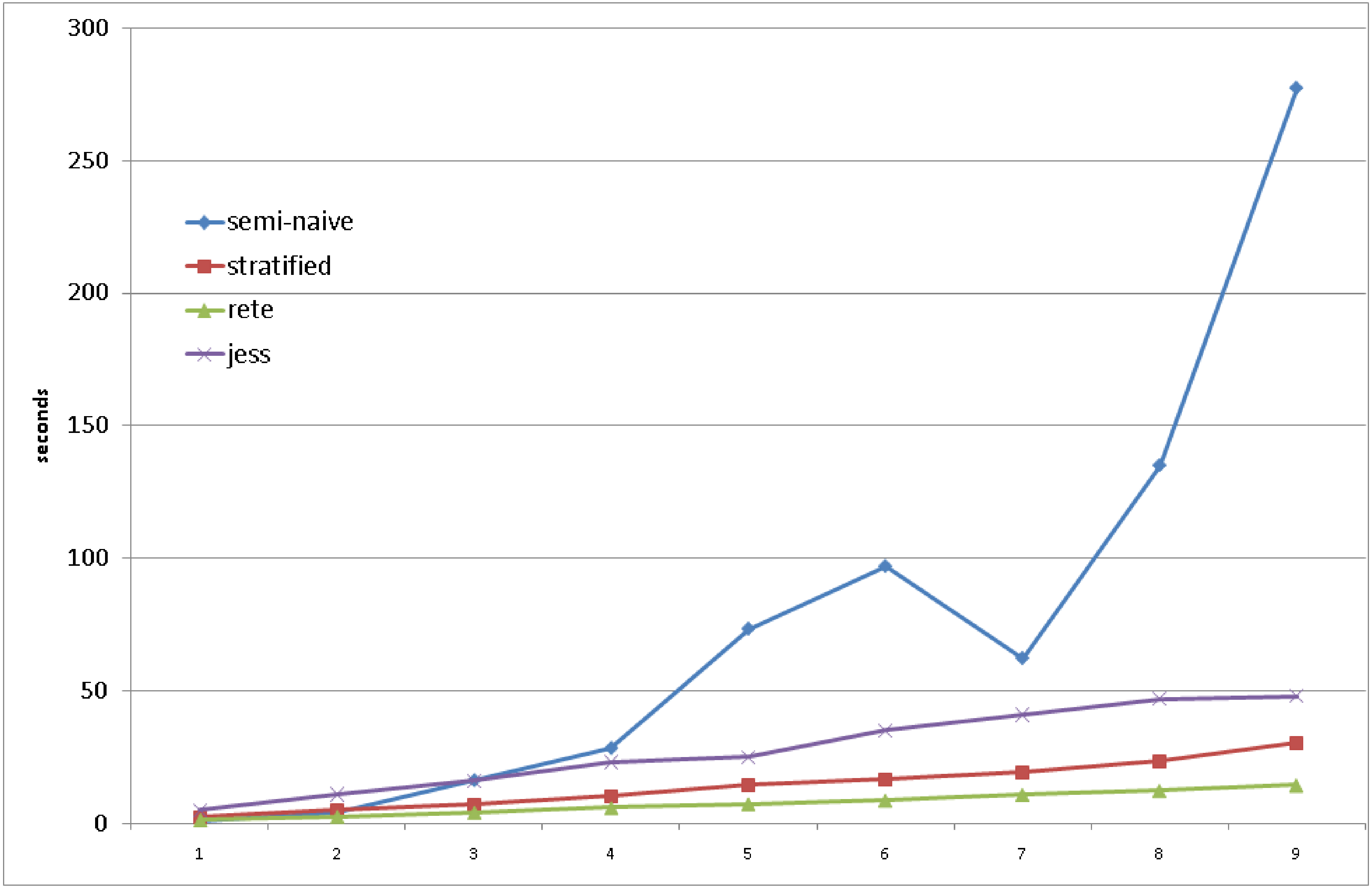


Results



short program version
long program version
seconds
600
500
400
300
200
100
0
1
2
3
4
5
6
7
8
9
semi-naive
stratified
rete
jess

Time differences between the two versions:



Conclusions

- Rete seems to be a considerable improvement in our test cases
- Our implementation had the smallest difference between the optimized and the redundant version, so it made effective re-use of computed results
- There was no drawback with increased EDBs
- So Rete is applicable to deductive databases
- We have to investigate it with recursive programs
- Extend Rete to full F-logic

Thanks for your attention!

Literature

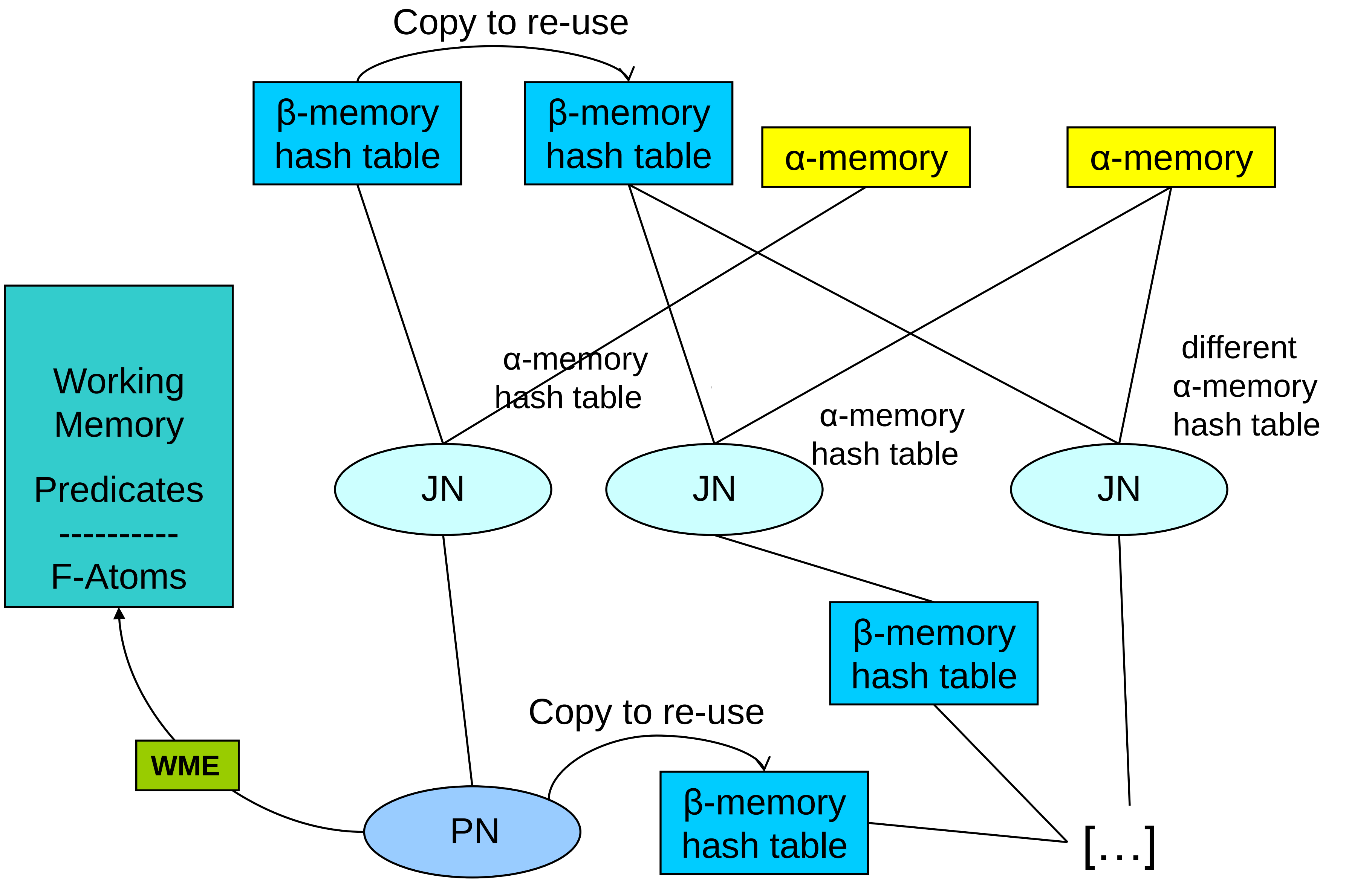
- Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Charles L. Forgy. Artificial Intelligence 19, 1982
- Production Matching for Large Learning Systems. Robert B. Doorenbos. PhD thesis, 1995
- How to Write F-Logic Programs in FLORID. Wolfgang May, Pedro José Marrón, 1999
- Florid User Manual. Wolfgang May, 2000

Florid example

Separating countries in island and midlands

```
borders(de,fr).           encompasses(fr,europe).
encompasses(de,europe). encompasses(is,europe).

X:midland :- borders(X,_).
X:midland :- borders(_,X).
?- sys.strat.doIt.

X:island(Y) :- encompasses(X,Y), not X:midland.
?- sys.eval.

?- X:island(Y).

% Answer to query : ?- X:island(Y).

is:island(europe).
```

Implementation in Florid



Short program version

```
parentOf(X,Y) :- border(X,Y).
childOf(X,Y) :- border(Y,X).
grandParentOf(X,Y) :- border(X,Z), border(Z,Y).
grandChildOf(X,Y) :- border(Y,Z), border(Z,X).
grandGrandParentOf(X,Y) :- grandParentOf(X,Z), border(Z,Y).
grandGrandChildOf(X,Y) :- grandChildOf(X,Z), border(Y,Z).
grandGrandGrandParentOf(X,Y) :- grandParentOf(X,Z),
  grandParentOf(Z,Y).
grandGrandGrandChildOf(X,Y) :- grandChildOf(X,Z), grandChildOf(Z,Y).

sibling(X,Y) :- border(Z,X), border(Z,Y).
spouse(X,Y) :- border(X,Z), border(Y,Z).
uncleOf(X,Y) :- border(Z,X), grandParentOf(Z,Y).
greatUncleOf(X,Y) :- grandGrandParentOf(W,Y), grandParentOf(W,X).

niceOf(X,Y) :- border(Z,Y), grandParentOf(Z,X).
cousin(X,Y) :- grandParentOf(U,X), grandParentOf(U,Y).
```

Long program version

```
parentOf(X,Y) :- border(X,Y).
childOf(X,Y) :- border(Y,X).
grandParentOf(X,Y) :- border(X,Z), border(Z,Y).
grandChildOf(X,Y) :- border(Y,Z), border(Z,X).
grandGrandParentOf(X,Y) :- border(X,Z), border(Z,U), border(U,Y).
grandGrandChildOf(X,Y) :- border(Y,Z), border(Z,U), border(U,X).
grandGrandGrandParentOf(X,Y) :- border(X,Z), border(Z,U),
  border(U,V), border(V,Y).
grandGrandGrandChildOf(X,Y) :- border(Y,Z), border(Z,U),
  border(U,V), border(V,X).
sibling(X,Y) :- border(Z,X), border(Z,Y).
spouse(X,Y) :- border(X,Z), border(Y,Z).
uncleOf(X,Y) :- border(Z,X), border(Z,U), border(U,Y).
greatUncleOf(X,Y) :- border(W,U), border(W,V), border(U,Z),
  border(Z,Y), border(V,X).
niceOf(X,Y) :- border(Z,Y), border(Z,W), border(W,X).
cousin(X,Y) :- border(V,X), border(U,V), border(U,W), border(W,Y).
```