

Expert Systems a.k.a. Business Rules Engines, *another kind of AI*

Laurent Magnin, Ph.D.

MGL 7320: Ingénierie logicielle des systèmes
d'intelligence artificielle

Let's begin with...
some history

Expert Systems, the AI precursor & AI savior of the 80'

“Expert systems were formally introduced around 1965.”

“... such as diagnosing infectious diseases (Mycin) and identifying unknown organic molecules (Dendral).”

“In the early 1980s, AI research was revived by the commercial success of expert systems. By 1985, the market for AI had reached over a billion dollars.”

https://en.wikipedia.org/wiki/Artificial_intelligence

A large variety of applications

- Finance / Banks
 - Morgan Stanley, Fidelity Investments, Casden Banque Populaire, BNP Paribas, Banque de France (AMF)
- Insurance
 - Shenandoah Life Insurance, Farmers, Manulife, Intact
- Public Sector
 - Hydro Quebec, State of Nevada, Teranet
- HealthCare
 - Partners HealthCare System, Optum

Fine, but what is...
an expert system?

Are Expert Systems...

- Systems that reproduce the reasoning of (human) experts?
 - Actually, Experts are mostly relaying on past experiences... (Case-based reasoning)
- Systems that (can) replace (human) experts?
 - Systems that can compute complex decisions
 - Not based on hardcoded sequences of formulas/functions
 - But on "Unknown" combinations of individual/local reasoning

The key element of the ES:

Inference

Inference...

- Starting from a set of initial piece of knowledge
- Then, applying successive rules to this knowledge base to produce additional knowledge
- Two types of inference (based on logical "rules"):
 - Forward Chaining (the most common)
 - To deduce (all possible) new knowledge from existing “facts”
 - Backward Chaining
 - To prove hypothesis or to answer specific questions

“Logical” Rules...

- Logical relations between facts
- Can be expressed by $p(f) \Rightarrow p'(f)$
 - If/when conditions then actions (!)

To be (a vehicle) or not to be?

`if car(x) then vehicle(x)`

`if car(x) then hasWheels(x)`

`if plane(x) then vehicle(x)`

`if vehicle(x) then canMoveByItself(x)`

car(Corolla)

1. `plane(Corolla)?` `canMoveByItself(Corolla)?`

- backward or forward chaining?

2. \Rightarrow `vehicle(Corolla), canMoveByItself(Corolla),`
`hasWheels(Corolla)`

Forward Chaining: **how to implement it?**

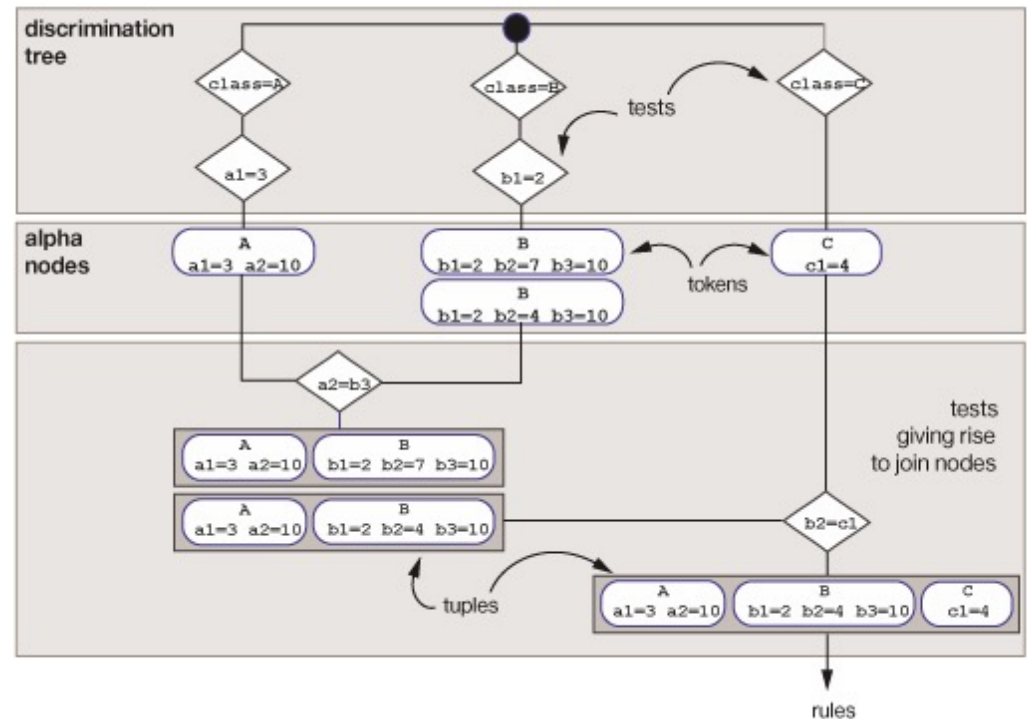
The *RETE* algorithm

- The RETE algorithm was designed by Charles L. Forgy of Carnegie Mellon University between 1974 and 1982
- Was developed to *efficiently* apply many patterns (i.e. the conditions of the rules) to many objects
- Became the root of the **Rules Engines**
- (The RETE algorithm is no more the most efficient one, but is still usually the most flexible)

The pattern matching of the *RETE* algorithm

A(a1=3 a2=10)
 B(b1=2 b2=4 b3=10)
 B(b1=2 b2=7 b3=10)
 C(c1=4)

```
rulefilter {
  when {
    A (a1==3; ?x:a2);
    B (b1==2; ?y:b2; b3==?x);
    C (c1==?y);
  }
  then {
    System.out.println("filter");
  }
}
```



https://www.ibm.com/support/knowledgecenter/SS7J8H/com.ibm.odm.dserver.rules.designer.run/optimizing_topics/tpc_opt_reteplusalgo.html

The key advantages of the *RETE* algorithm

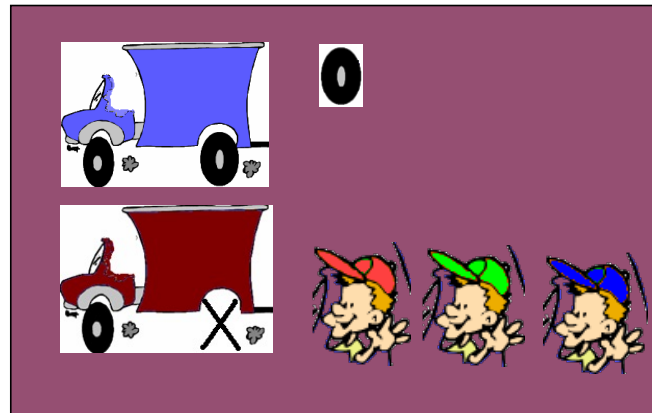
- No need for a *full* reevaluation of the RETE network when:
 - Objects / facts are added
 - objects / facts are removed
 - objects / facts are updated
- Some rules can be partially combined
 - Only when they start with the *same conditions*, in the *same order*
- (The most discriminant conditions should be declared first)

An iterative process... to explore all derivative Knowledge

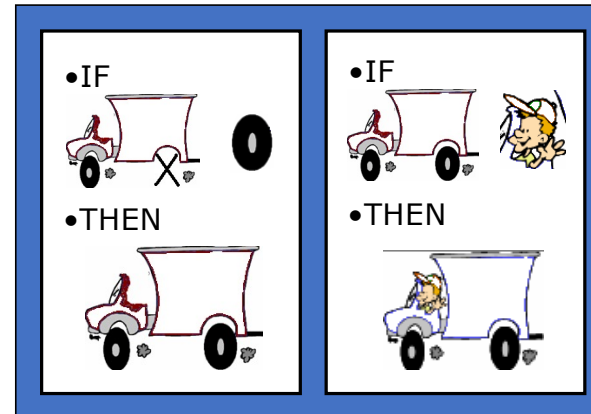
Agenda



Working Memory





Rules







An iterative process...



Agenda



- IF
- THEN

Working Memory



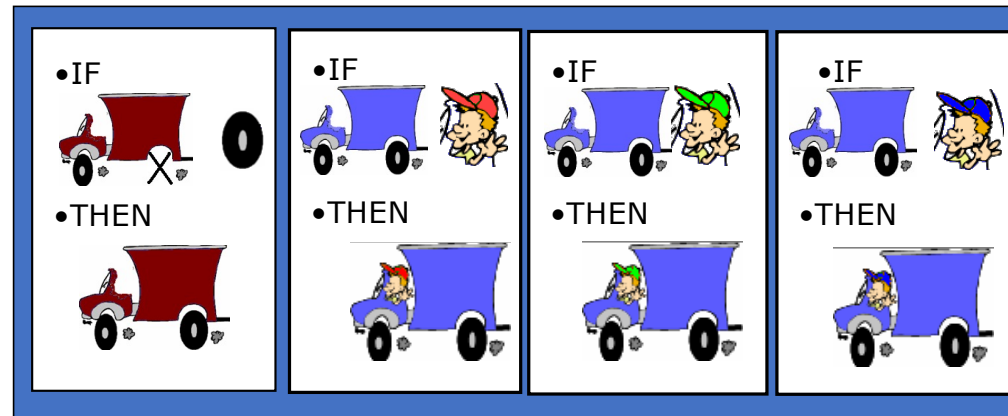
Rules

- IF
- THEN

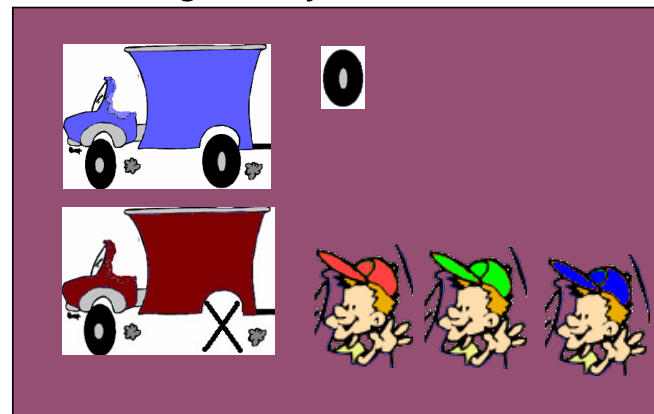
- IF
- THEN

An iterative process...

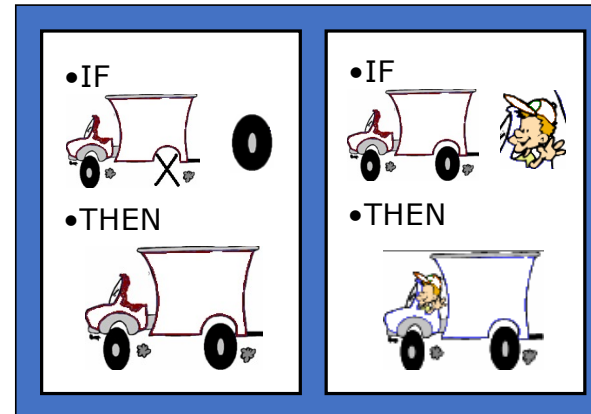
Agenda



Working Memory

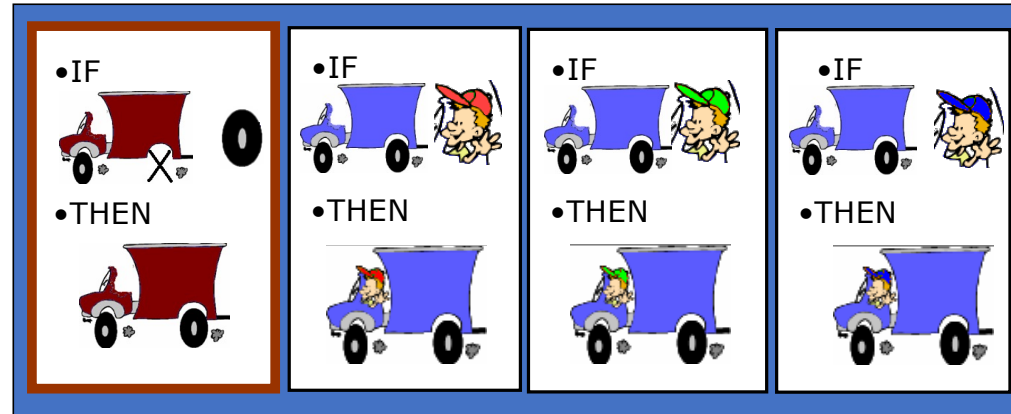


Rules

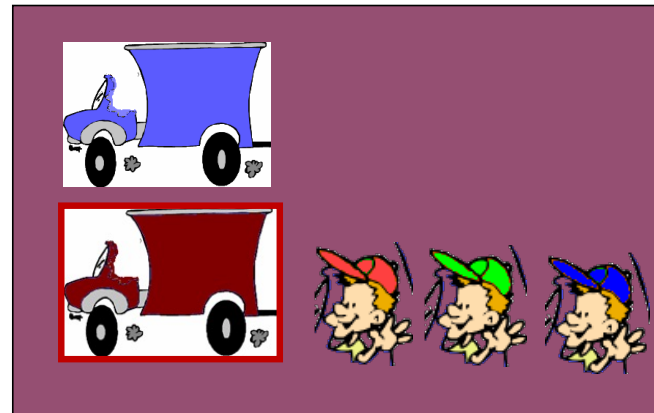


An iterative process...

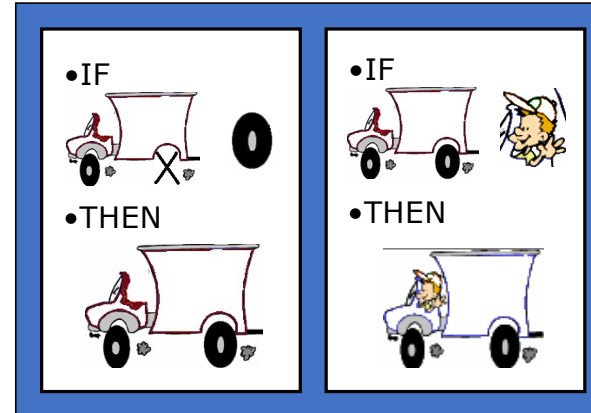
Agenda



Working Memory

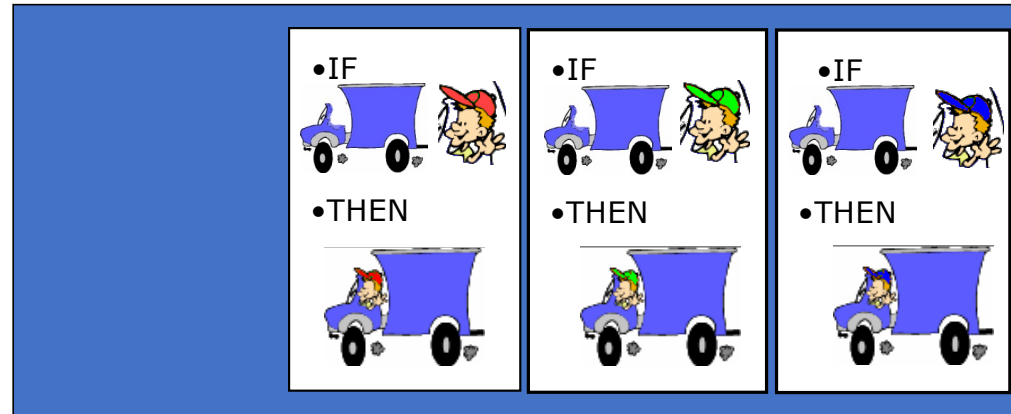


Rules

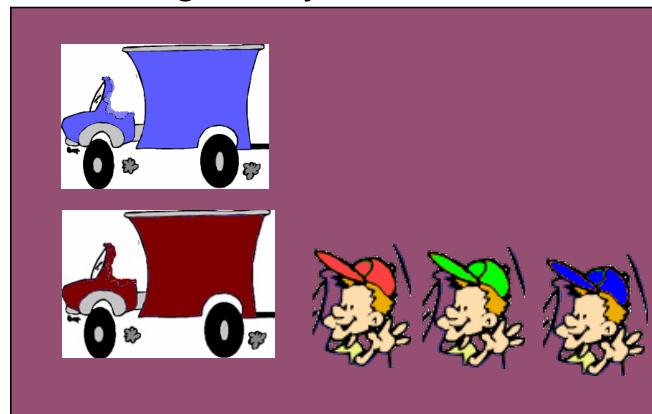


An iterative process...

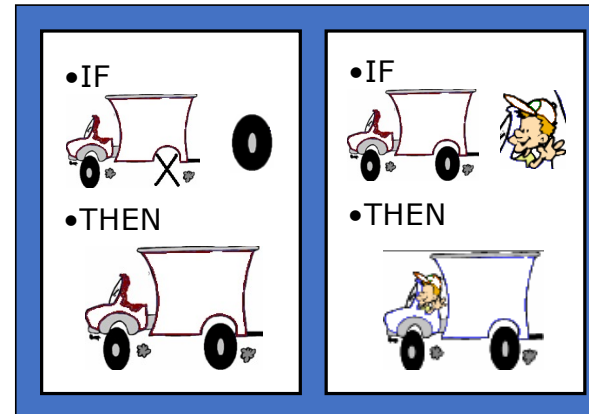
Agenda



Working Memory

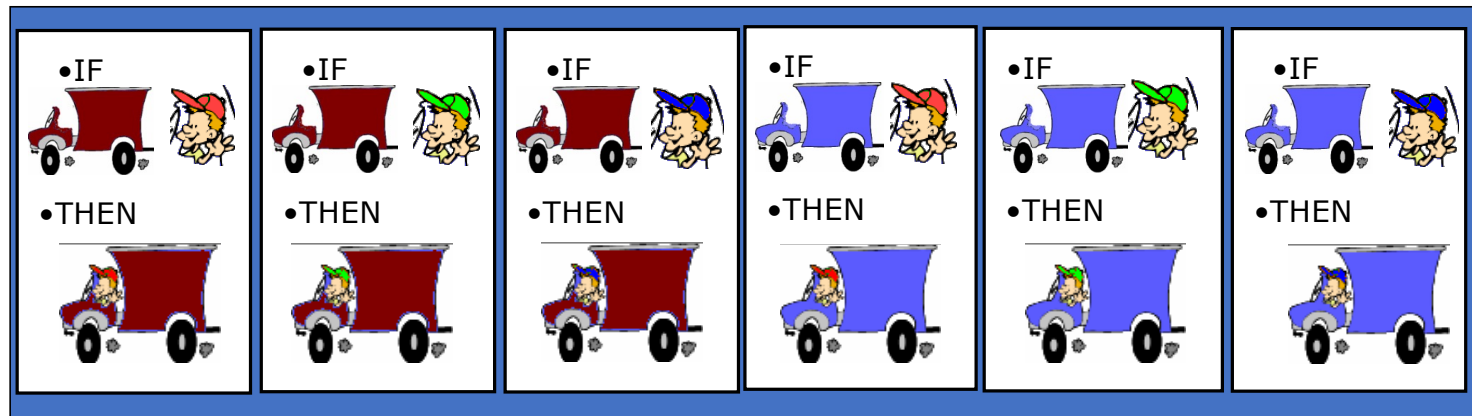


Rules



An iterative process...

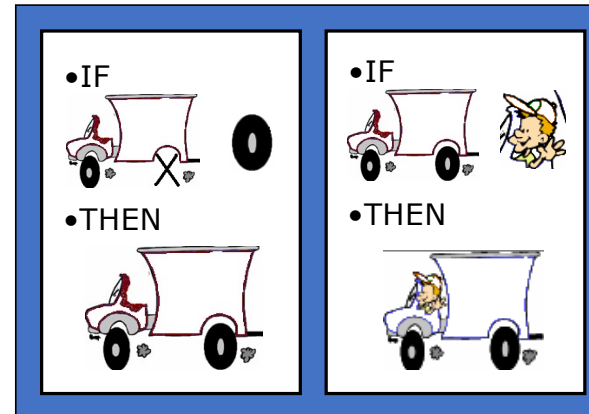
Agenda



Working Memory

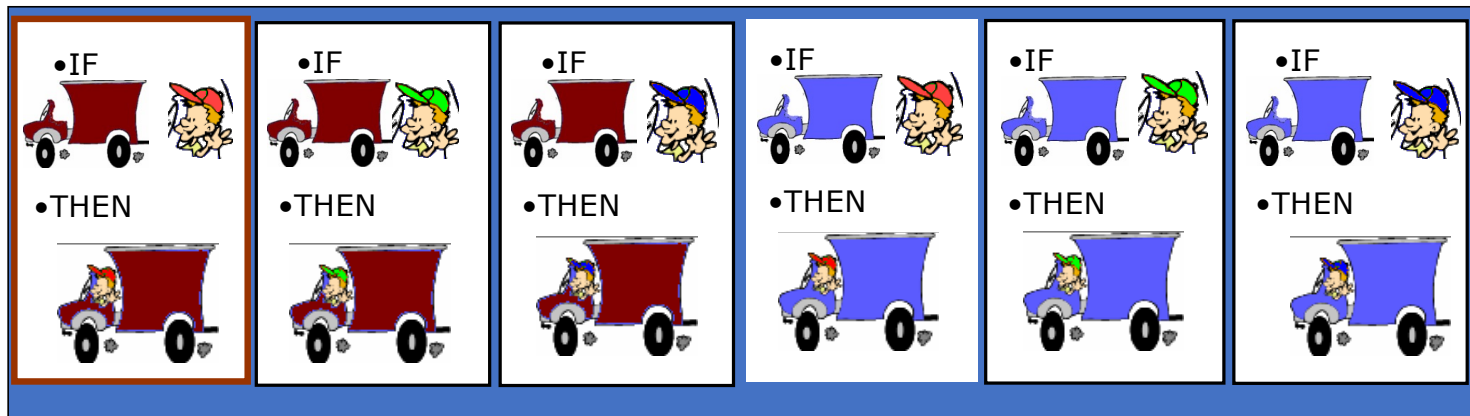


Rules



An iterative process...

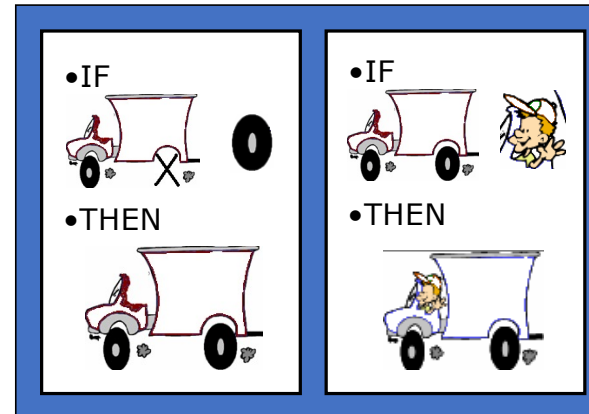
Agenda



Working Memory

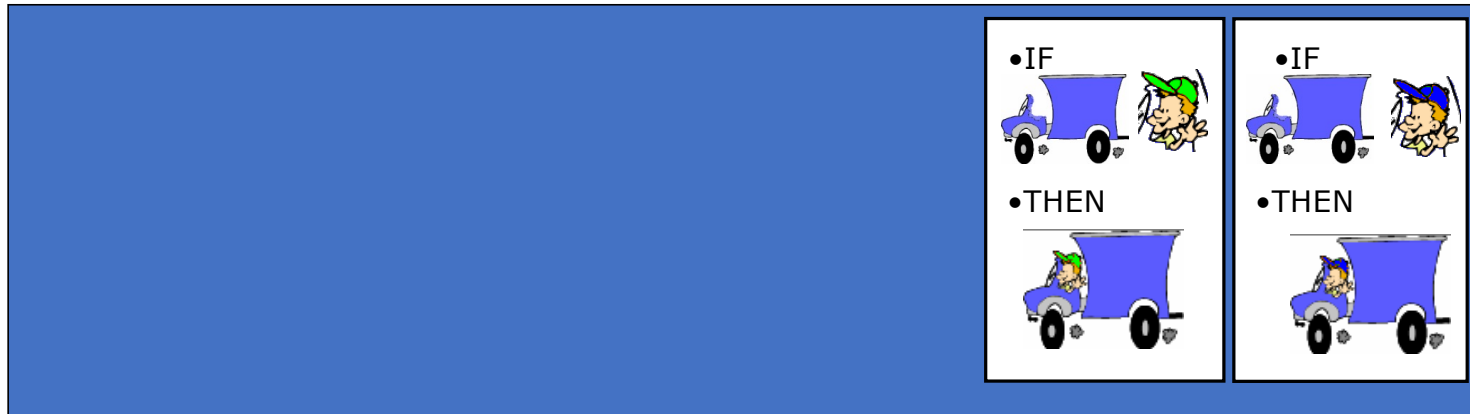


Rules

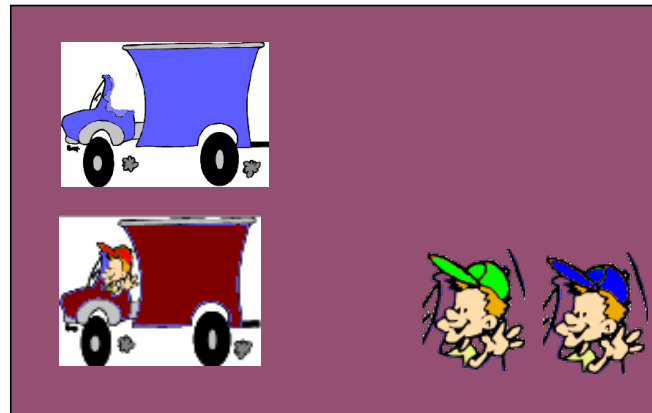


An iterative process...

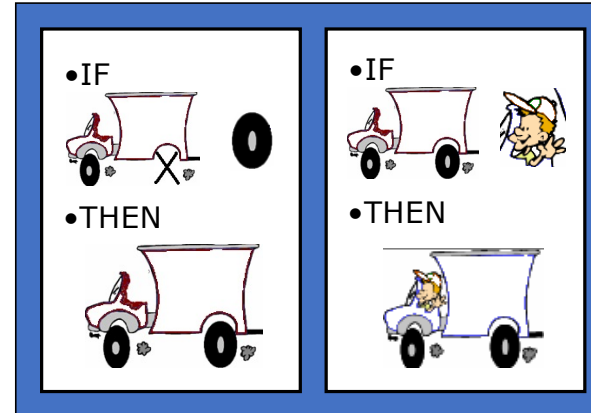
Agenda



Working Memory

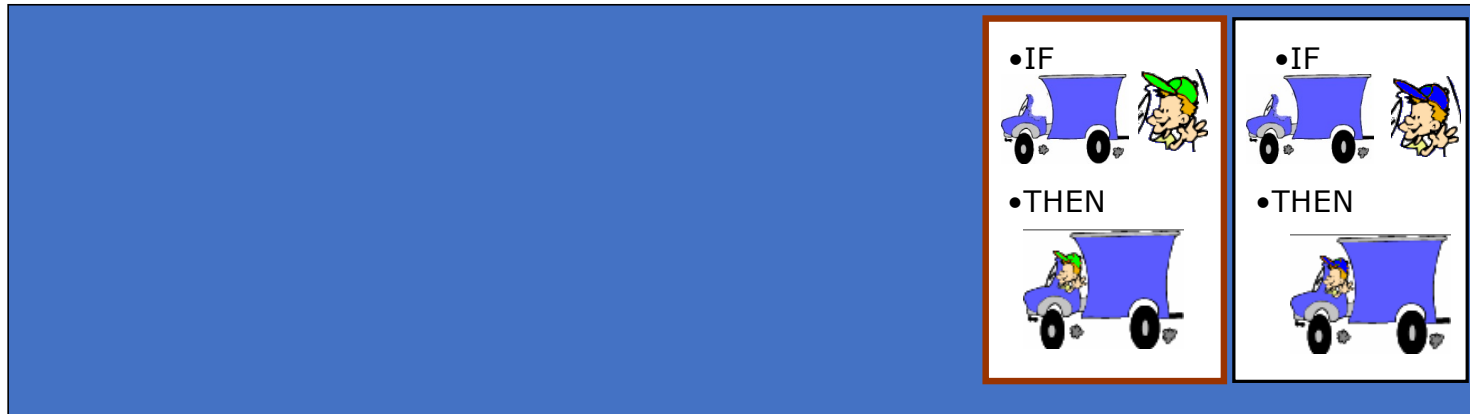


Rules

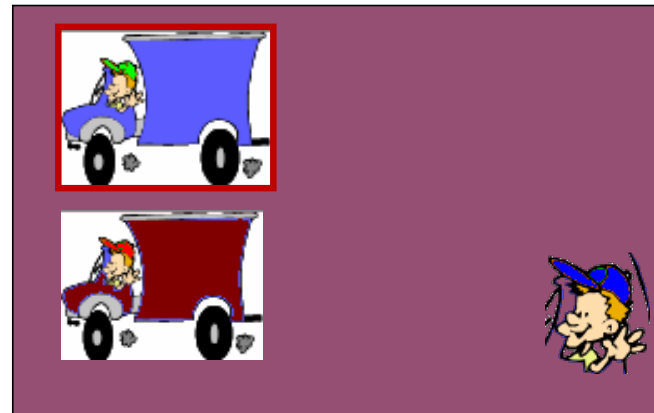


An iterative process...

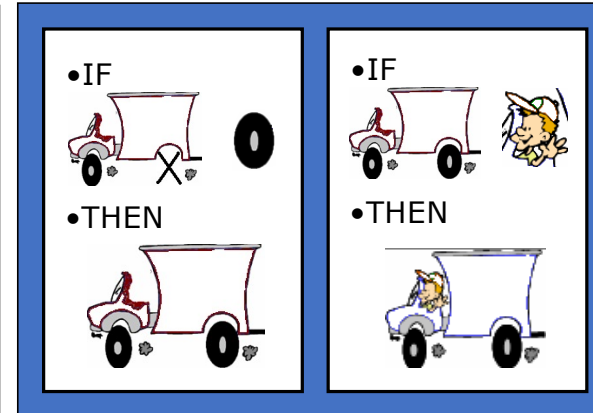
Agenda



Working Memory



Rules



An iterative process... until no more eligible rules

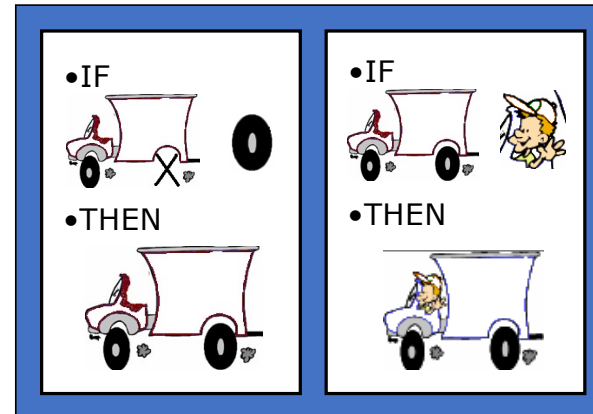
Agenda



Working Memory



Rules



Rule Engines **based on the RETE algorithm**

Rule Engines based on the *RETE* algorithm

- CLIPS (created at NASA in 1985)
 - Written in C but with a Lisp like syntax
- JESS
 - An alternative written in Java, again using a Lisp like syntax

```
Jess> (bind ?x 1)
1
Jess> (if (> ?x 100) then
      (printout t "X is big" crlf)
      else
      (printout t "X is small" crlf))
X is small
```

- ...

Experta, the inspired by CLIPS Python library

```
from random import choice
from experta import *

class Light(Fact):
    """Info about the traffic light."""
    pass

class RobotCrossStreet(KnowledgeEngine):
    @Rule(Light(color='green'))
    def green_light(self):
        print("Walk")
    @Rule(Light(color='red'))
    def red_light(self):
        print("Don't walk")
    @Rule(AS.light << Light(color=L('yellow') | L('blinking-yellow'))
    def cautious(self, light):
        print("Be cautious because light is", light["color"])
```

Experta, the inspired by CLIPS Python library

```
>>> engine = RobotCrossStreet()

>>> engine.reset()

>>> engine.declare(
    Light(color=choice(['green', 'yellow', 'blinking-yellow', 'red']))

>>> engine.run()
Be cautious because light is blinking-yellow
```

- Great fit when you have to call simple rules from your existing Python code

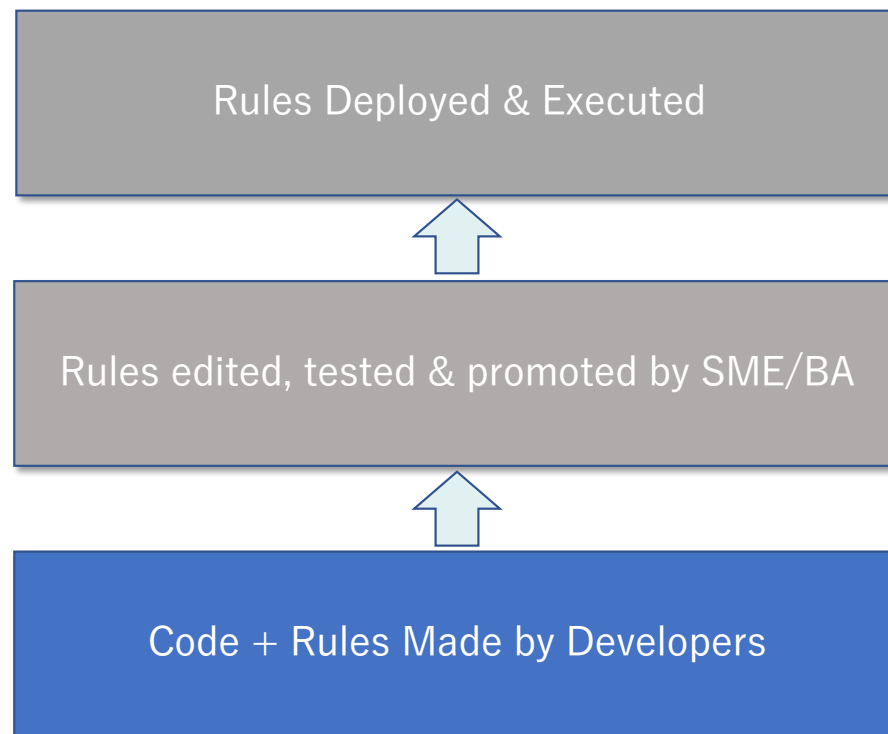
Rule Engine: is it enough?

(simple) Rule Engines do lack...

- A language to easily express rules (by SME)
- Ways to logically regroup rules
- Rules represented by Decision Tables
- A collaborative platform (through a common Web interface)
- Integrated rules governance, packaging & deployment
- Robust, very fast & distributed engines
- ...

Business Rule Management Systems, beyond basic Rule Engines

Business Rule Management Systems



ODM and Drools, two examples of BRMS

The image displays two overlapping software interfaces. The background interface is ODM (Open Decision Model), showing a decision model for 'Loan Pre-Qualification'. It features a central node 'Loan Pre-Qualification' with arrows pointing to it from several input nodes: 'Lender Acceptable DTI', 'Back End Ratio', 'Front End Ratio', 'Lender Acceptable PITI', and 'Credit Score Rating'. A 'Requested Product' node is also shown below. The foreground interface is Drools, showing a 'Code' editor with a decision rule. The rule starts with 'definitions' and 'set' statements, followed by an 'if' condition that checks the billing address and age of 'the customer'. The 'then' part of the rule states that 'the customer' is eligible for the 'DF5' product. A list of locations is visible on the right side of the Drools interface.

Model Built for an example defined by integrated BPMN, and Learning p

```
definitions
    set 'the customer' to a customer;
if
    the billing address of 'the customer' is in "Albertville"
    and the age of 'the customer' is at least 25
then
    'the customer' is eligible for the "DF5" pr
```

- ▲ Abbeville
- ▲ Alabaster
- ▲ Albertville
- ▲ Alexander
- ▲ Anniston
- ▲ Ardmore
- ▲ Athens
- ▲ Atmore
- ▲ Attalla

Expert Systems a.k.a. Business Rules Engines,

Conclusion

(Business) Rule Engines...

- Can be a *very powerful* way to implement *certain* types of AI, when:
 - Knowledge is not made of data, but of individual "rules"
 - Explanations (& some reproducibility) are required
- Developing “Expert Systems” requires:
 - Dedicated tools (Rule Engine or BRMS)
 - Specific mindset (not procedural programming, not ML...)