Enhancing Rules with Imperfection

Integration with Soft Computing

Davide Sottara [dsotty@gmail.com]

[a]Department of Computer Science, Electronics and Systems
Faculty of Engineering, University of Bologna
Viale Risorgimento 2, 40100 Bologna (BO) Italy

Bologna/San Diego - April 19-23th, 2010

Outline

Outline

Outline

What is Imperfection?

Imperfection

Imperfection, be it Imprecision or Uncertainty, pervades . . . systems that attempt to provide an accurate model of the real world

P.Smets, 1999

What is Imperfection?

Imperfection

Imperfection, be it Imprecision or Uncertainty, pervades . . . systems that attempt to provide an accurate model of the real world

P.Smets, 1999

Uncertainty

Uncertainty is a condition where Boolean truth values are unknown, unknowable, or inapplicable . . .

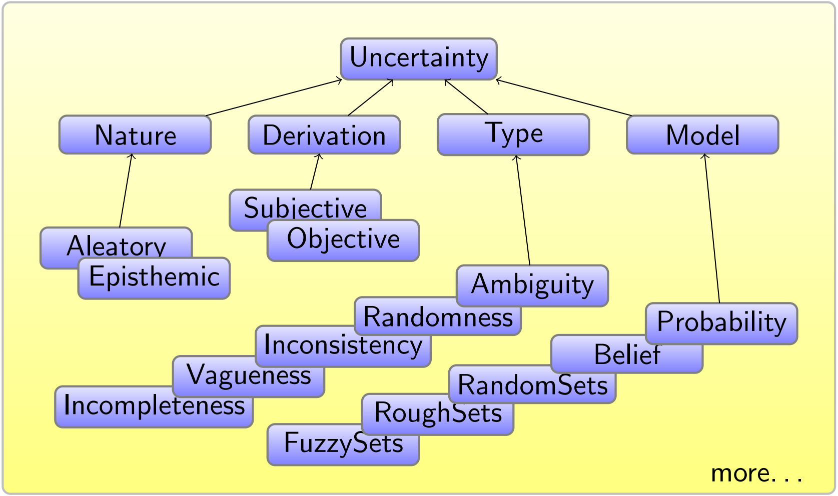
W3C Incubator Group on Uncertainty Reasoning for the Web, 2005

What is Imperfection?

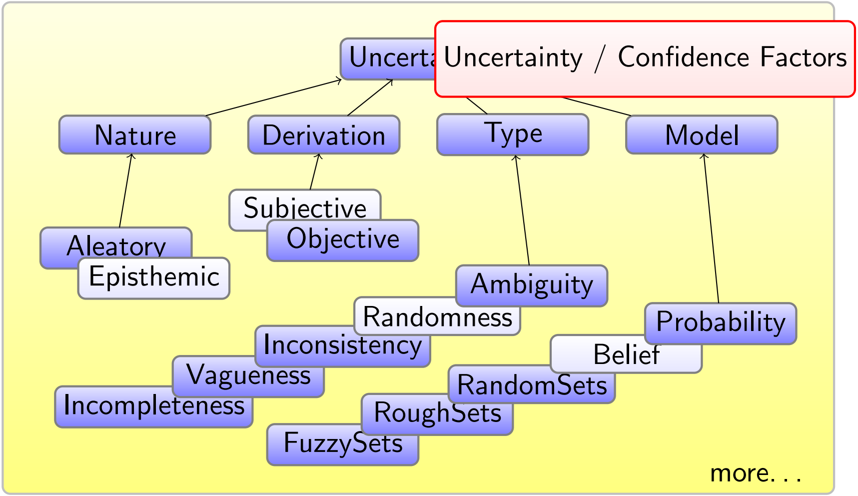
Imperfection - a negative definition

Uncertainty/Imperfection is the opposite of preciseness and certainty, i.e. of what Boolean logic models

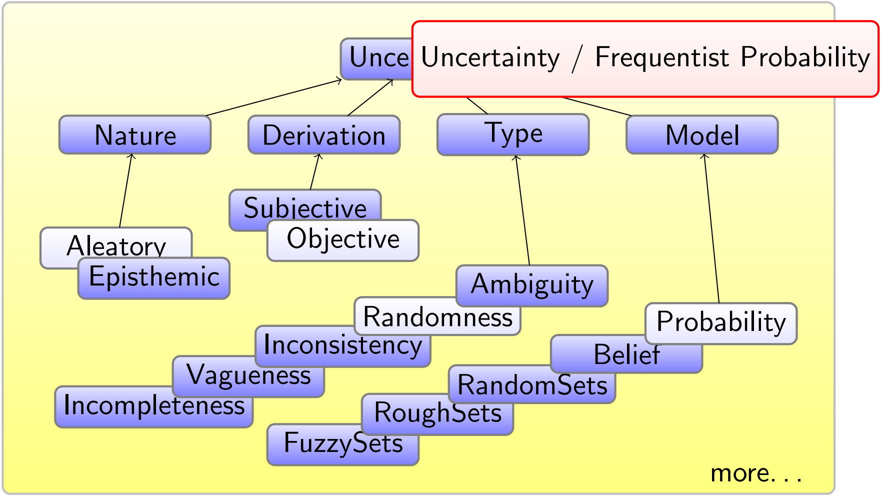
An Ontology for Imperfection



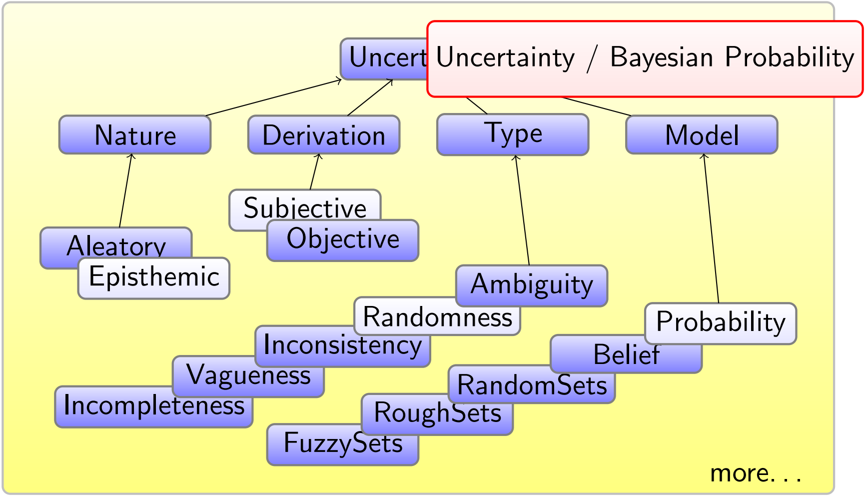
An Ontology for Imperfection



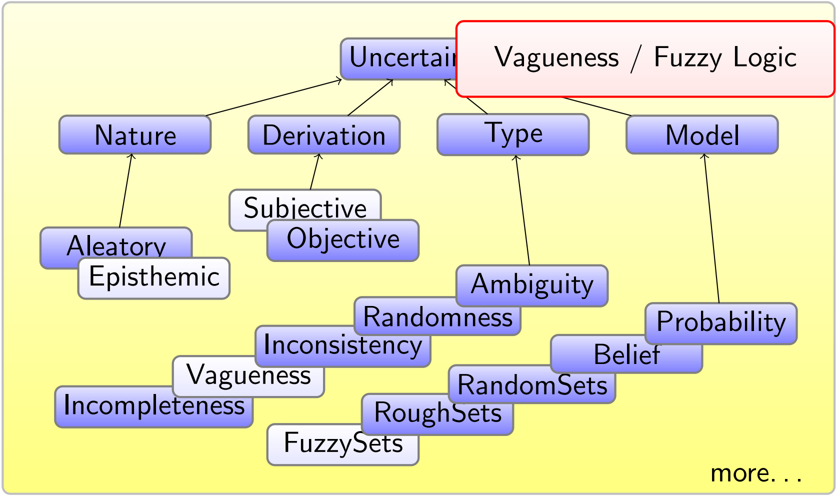
An Ontology for Imperfection



An Ontology for Imperfection



An Ontology for Imperfection



Possible Applications

- Information Processing
  - Clinical Procedures
  - Semantic Web

Possible Applications

- Information Processing
  - Clinical Procedures
  - Semantic Web
- Classification
  - Symptom Matching
  - Fraud Detection

Possible Applications

- Information Processing
  - Clinical Procedures
  - Semantic Web
- Classification
  - Symptom Matching
  - Fraud Detection
- Prediction
  - Prognosis
  - Stock Market

Possible Applications

- Information Processing
  - Clinical Procedures
  - Semantic Web
- Classification
  - Symptom Matching
  - Fraud Detection
- Prediction
  - Prognosis
  - Stock Market
- Diagnosis
  - Health Care
  - Machine Failure

Possible Applications

- Information Processing
  - Clinical Procedures
  - Semantic Web
- Classification
  - Symptom Matching
  - Fraud Detection
- Prediction
  - Prognosis
  - Stock Market
- Diagnosis
  - Health Care
  - Machine Failure
- Monitoring
  - Vital Sign Monitoring
  - Video-Surveillance

Possible Applications

- Information Processing
  - Clinical Procedures
  - Semantic Web
- Classification
  - Symptom Matching
  - Fraud Detection
- Prediction
  - Prognosis
  - Stock Market
- Diagnosis
  - Health Care
  - Machine Failure
- Monitoring
  - Vital Sign Monitoring
  - Video-Surveillance
- ...

Outline

Using Imperfection

Rules should handle uncertainty, not ignore it

Benefits

- Conciseness
- Robustness

Drawbacks

- Complexity
- Correctness and Coherence

Using Imperfection

Rules should handle uncertainty, not ignore it

Benefits

- Conciseness
- Robustness

Drawbacks

- Complexity
- Correctness and Coherence

”**If** *you place your bet on an improbable number,*
and *it gets extracted on next round,*
then *expect an increase in your capital*”

Some Issues

”**If** *you place your bet on an improbable number,*
and *it gets extracted on next round,*
then *expect an increase in your capital”*

Some Issues

If bet(Sum,Number,T)
$\wedge$ improbable(Number)
$\wedge$extracted(Number,T+1)

Some Issues

If bet(Sum,Number,T)
$\wedge$ improbable(Number)
$\wedge$extracted(Number,T+1)

Then
eval(bet(Sum,Number,T),B)
eval$_F$(improbable(Number), D)
eval$_P$(extracted(Number), P)
eval$_\wedge$(B,D,P,Number,T,X)
eval$_\rightarrow$(X,Number,T)
print('Gain is',gain(Number,X))

Some Issues

If bet(Sum,Number,T)
$\wedge$ improbable(Number)
$\wedge$extracted(Number,T+1)

Then
eval(bet(Sum,Number,T),B)
eval_F(improbable(Number), D)
eval_P(extracted(Number), P)
$\text{eval}_\wedge$(B,D,P, X)
$\text{eval}_\rightarrow$(X)
print('Gain is',gain(Number,X))

- Truth-functionality
  - Simplifies computation
  - Not always possible (e.g. probability)

Some Issues

If bet(Sum,Number,T)
$\wedge$ improbable(Number)
$\wedge$extracted(Number,T+1)

Then

print('Gain is',gain(Number,X))

- Truth-functionality
  - Simplifies computation
  - Not always possible (e.g. probability)
- Transparency
  - Automatic computation
  - User should not be aware

Outline

Outline

Generalized Inference

$$\frac{\langle P(x), P(X) \rightarrow C(Y) \rangle}{C(y)}$$

- Classic Modus Ponens
  - Premise and Implication entail Consequence

Example

$Rich(X) \wedge Healthy(X) \rightarrow Happy(X)$

Generalized Inference

$$\frac{\langle \Phi(\ldots, A_j(x)/\varepsilon_j, \ldots), P(X) \rightarrow C(Y) \rangle}{C(y)}$$

- Premise
  - Atomic constraints are evaluated
  - General, pluggable `Evaluators`
  - A `Degree` is returned

Example

$Rich(x)_{0.6} \wedge Healthy(x)_{0.8} \rightarrow Happy(X)$

Generalized Inference

$$\frac{\langle \Phi(\ldots, A_j(x)/\varepsilon_j, \ldots)/\varepsilon_P, P(X) \rightarrow C(Y)\rangle}{C(y)}$$

- Premise
  - Atomic constraints are evaluated
  - General, pluggable `Evaluators`
  - A `Degree` is returned

- Premise
  - Atoms are aggregated in formulas
  - using generalized logic Connectives
  - evaluated by `Operators`

Example

$Rich(x) \wedge_{0.6} Healthy(x) \rightarrow Happy(X)$

Generalized Inference

$$\frac{\langle P(x)/\varepsilon_P \;, \;\rightarrow(X,Y)/\varepsilon_{\rightarrow}\rangle}{C(y)}$$

- Implication
  - Implication has a Degree
  - often given *a priori*

Example

$Rich(x) \wedge Healthy(x) \rightarrow_{0.4} Happy(X)$

Generalized Inference

$$\frac{\langle P(x)/\varepsilon_P \; , \; \rightarrow(X,Y)/\varepsilon_{\rightarrow} \rangle}{C(y)/\varepsilon_C}$$

- Implication
  - Implication has a Degree
  - often given *a priori*

- Modus Ponens
  - MP computes the Degree of the Consequence

Example

$Rich(x) \wedge_{0.6} Healthy(x) \rightarrow_{0.4} Happy(x)_{0.4}$

Generalized Inference

$$\frac{\frac{\langle P_1, \rightarrow_1 \rangle}{C_1 / \varepsilon_{C_1}}, \ldots, \frac{\langle P_n, \rightarrow_n \rangle}{C_n / \varepsilon_{C_n}}}{C(y) / \varepsilon_C}$$

- Merging multiple sources
  - Multiple premises for the same conclusion
  - Solve conflicts
  - Handle missing values

Example

$Rich(x) \wedge Healthy(x) \rightarrow Happy(x)_{0.4 \cap 0.2 \cap 0.7}$

Outline

Language extensions : Example

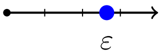
```
rule "Rule"
  // custom: implications and MP
  implication @[degree ="0.75"]
  deduction @[kind="min"]
when
    $o1 : Type( $f1 : field1
      /* custom: external evaluator */
      == @[id="i1", kind="external", params="..."]
    "val")
  or @[kind="max"]  // custom: operators
    $o2 : AnotherType(
      field3 == 0
      ^^ // custom: operators
      field3 == @[crisp] $f1 ) //custom: behaviour
then
  /* consequence degree */
  ... = drools.getConsequenceDegree();
```

Generalized Degrees

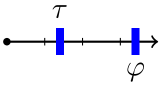
Degrees generalize the boolean `true/false`

- **truth**: compatibility with a prototype
- **probability**: ratio of relevant events over total
- **belief**: opinion in assuming a property to be true.
- **possibility**: disposition towards accepting a situation to be true.
- **confidence**: strength of an agent's belief in a statement.

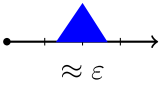
Different models, including:



Simple



Interval



Type-II degrees

Custom Evaluators

$$Object \times Object \mapsto Degree$$

```
when
   Patient( fever ~seems ''high'' )
then
   ...
```

- Wrap an external function
- Define (and evaluate) a property $p(L, R)$
- Return a *Degree*

Custom Operators

$\{(Tuple), Degree\}^n \mapsto Degree$

```
rule "Ops"
  implication
  deduction
when
  $p : Patient( temperature ~>= 38 ^^ ~<= 41 )
  and
  exists Medicine( this not ~allergenic $p )
then
  ...
```

- Aggregate evaluations
- Better if truth-functional
- Return a *Degree*
- Noteworthy : implication and modus-ponens

Configuration Attributes

Control the behaviour of the engine

- **id** : assign id to constraint/operator
- **kind** : choose evaluator/operator implementation
- **degree** : set “prior” degree
- **params** : additional initialization info
- **crisp** : cast to boolean
- **filter** : configure propagation strategy
- more...

Injection[1]

```
rule "Inject"
when
  $p : Patient( temperature ~>= 38 )
then
  inject(''idFever'',$p)
end

rule "Injected"
when
  $p Patient( fever ~seems @[id=''idFever''] ''high'' )
...
```

Chaining by evaluation: source conseguence degree sets the target's

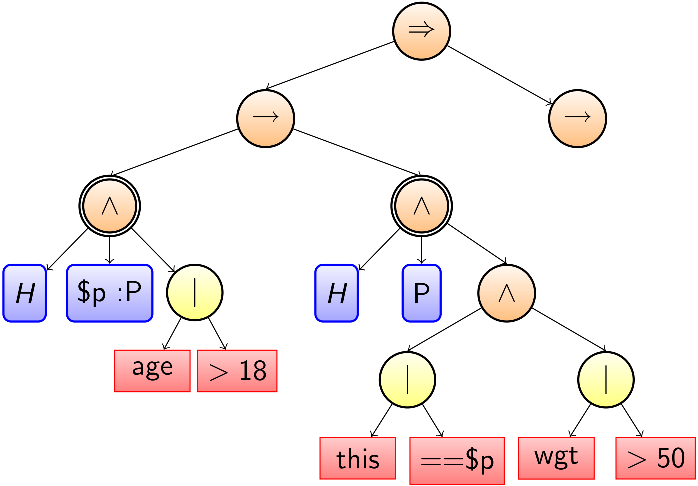
[1] Soon to be deprecated in form, but not in concept

Outline

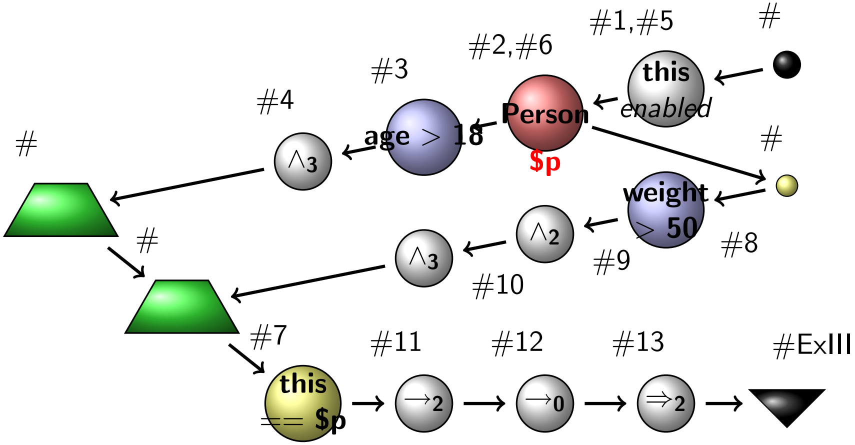
Refactored Rule Structure

```
when
   $p : Patient( age ~> 18 )
   implies
   Person( this == $p, weight ~> 50)
then
   ...
```

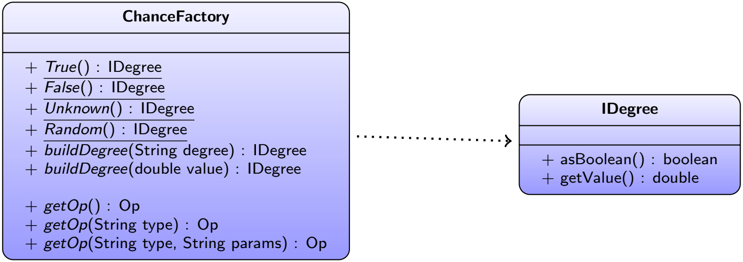
Refactored Rule Structure



Extended RETE



Factory



Factory controls the coherence

- Builds Degrees
- Builds Operators
- Attributes become params for the factory

Outline

Applications

Pure Logic-Based Approaches

- Symbolic reasoning
- Rules are annotated with degrees
- Computation of facts and degrees according to inference rules

Hybrid Approaches

- Mixed Symbolic/Sub-Symbolic reasoning
- Rule delegate, embed or emulate SC techniques

Outline

Certainty Factors

```
rule "Mycin"
  implication @[ degree=''0.7'' ]
    when
      $s : Site( this ~sterile )
       Infection( cause ~== ''bacteremia'',
                  site == @[crisp] $s )
      ...
    then
       // Infection is bacteroid
  end
```

- Simple rule structure
- Evaluators return CF
- Rules have CF themselves

Bayesian Logic Programs

```
rule "BLP"
      //CPT here : p(m|S1,S2)
  implication @[ degree=‘‘...’’ ]
    when
      $s1 : Symptom1( ... )
       $s2 : Symptom2( ... )
    then
      // Illness is ...
     end
```

- Conditional probabilities over state of premises

Many-Valued (Fuzzy) Logic Programs

```
rule "MVL"
  implication @[ kind=''Lukas'' ]
    when
      Patient( pressure ~seems ''high''
                     || @[ kind=''max'' ]
               temperature ~seems ''high'' )
    then
      // ...
    end
```

- Variety of operators (families)
- Full fuzzy set chaining not complete (yet)

Possibilistic Logic Programs

- Degrees given by Necessity/Possiblity intervals
- Similar in form to a specific MVL
    - Specific operators
    - Specific semantics
    - Not gradual truth, nor probability!!
- ... but generalizes to fuzzy possibility easily

Hybrid Logic Programs

```
rule "Hybrid Imperfect"
  // probability
  implication @[ degree=''0.99'' ]
when
  // truth
  true @[ degree=''0.5,0.7''] (
    Patient( temperature ~seems ''high'' )
  )
then
// ...
end
```

- Uncertain/Vague Mix
- Consequence is given a specific probability...
- ... if and only if premise is true to a certain degree

Outline

Soft Computing

Alternative (?) to Rule-Based Systems

A vast family

Basically, everything that is not (purely) symbolic

- Fuzzy Logic
- Neural Networks
- Genetic Algorithms
- Bayesian Network
- Clustering

Outline

No Integration - External Call

```
rule "No integration"
when
   $s : SCModule(...)
then
   $s.invoke(...);
end
```

- Rules, at best, select the SC module
- SC module is invoked in RHS
- Compatible with boolean logic

Loose Integration - Wrapper

```
rule "Cytofluorimetry"
when
  // Using a neural classifier
  $c : Cell( $f : features ~isA ''red globule'' )
then
  ...
end
```

- SC module is embedded in a custom evaluator
- SC module must evaluate a predicate
- i.e. the return value must be a *Degree*
- Boolean return value would be a limitation

Strong Integration - Emulation

- SC module is implemented using (imperfect) rules
- Based on *Degree* manipulation - using operators

Outline

Induction

```
rule "Induction"
when
   forany ( $p : Patient( heart ~risk ''high'' )
               subject_to
                 Patient(this == $p,
                    weight ~seems ''heavy'' ))
then
   ...
```

- Generalized quantifier
- Accumulates quantitative degrees

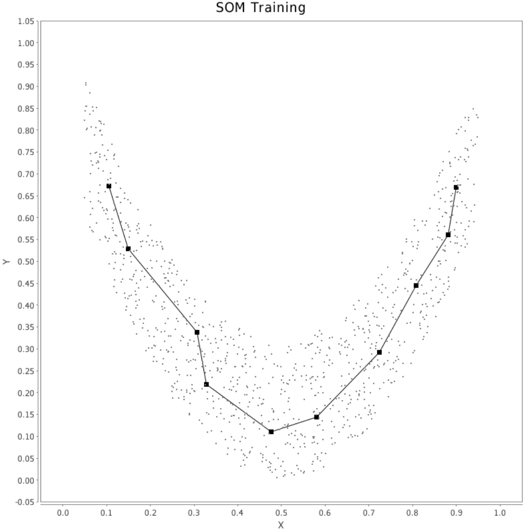
Self-Organizing Map

```
rule "Map Query"
when
  $x : Sample()
  exists Neuron( position ~close $x )
then
  // (Gradual) Recall...
```

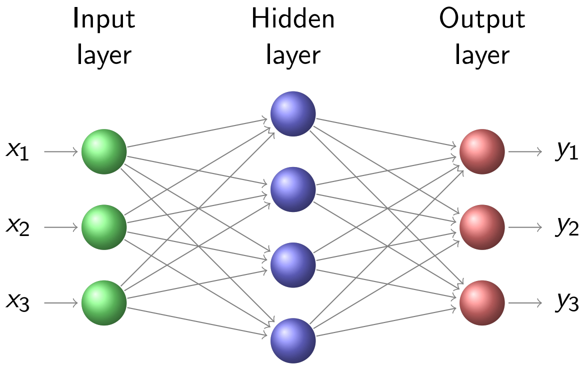
- Rule-based Training algorithm
- Rule-based querying

Self-Organizing Map

Example: 10 neurons in a 2D space:

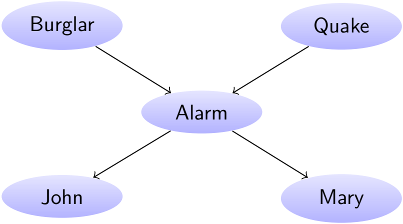


Feed-Forward Neural Network



- Function-Approximating Networks $\rightarrow$ Invoke
- Classification Networks $\rightarrow$ Wrap
- Emulation feasible, under development

Bayesian Network



- Wrappable for use in probabilistic logic
- Emulation is possible (still too verbose)

Outline

Conclusions

- Uncertainty exists in many forms

Conclusions

- Uncertainty exists in many forms

  $\longrightarrow$ Uncertainty should be embedded in rules

Conclusions

- Uncertainty exists in many forms

  $\longrightarrow$ Uncertainty should be embedded in rules

- Several Imperfect Logics do exist

Conclusions

- Uncertainty exists in many forms

  $\longrightarrow$ Uncertainty should be embedded in rules

- Several Imperfect Logics do exist
- Uncertainty can be handled using other approaches:
  - Bayesian Networks, Neural Networks, Fuzzy Systems, . . .

Conclusions

- Uncertainty exists in many forms

  $\longrightarrow$ Uncertainty should be embedded in rules

- Several Imperfect Logics do exist
- Uncertainty can be handled using other approaches:
  - Bayesian Networks, Neural Networks, Fuzzy Systems, ...
- Current Goal : Provide a unified and integrated framework