

*What Is Approximate Reasoning?*  
by Sebastian Rudolph, Tuvshintur Tserendorj,  
and Pascal Hitzler

Presented by Jesse Weaver  
CSCI 6965–Advanced Semantic Technologies  
Tetherless World Constellation  
Rensselaer Polytechnic Institute

January 27, 2009

# Outline

What is approximate reasoning?

Mathematical Framework

Composition of Reasoning Algorithms

Example

Conclusions

# What is approximate reasoning?

- ▶ “Approximate reasoning for the Semantic Web is based on the idea of sacrificing soundness or completeness for a significant speedup of reasoning.”

# What is approximate reasoning?

- ▶ “Approximate reasoning for the Semantic Web is based on the idea of sacrificing soundness or completeness for a significant speedup of reasoning.”
- ▶ “... allowing for unsound and/or incomplete reasoning procedures ... must not lead to arbitrary ‘guessing’ or to deduction algorithms which are not well-understood.”

# What is approximate reasoning?

- ▶ “Approximate reasoning for the Semantic Web is based on the idea of sacrificing soundness or completeness for a significant speedup of reasoning.”
- ▶ “... allowing for unsound and/or incomplete reasoning procedures ... must not lead to arbitrary ‘guessing’ or to deduction algorithms which are not well-understood.”
- ▶ “Often, the notion [of *approximate reasoning*] is associated with *uncertainty reasoning* .... The notion of approximate reasoning we use in this paper refers to approximate reasoning algorithms on data which is *not* uncertain in this sense.”

# INPUT SPACE

$$\text{InputSpace} = (\Omega, P)$$

$\Omega$  is the set of all queries that can be posed to the reasoning system.

$P$  is the set of all probabilities associated with queries in  $\Omega$  .

$P(\omega)$  denotes the probability of query  $\omega$  being asked.

# OUTPUT SPACE

$$\text{OutputSpace} = X$$

$X$  is the set of all possible outputs.

$\perp \in X$  denotes *no output* (distinct from the empty set).

# ERROR FUNCTION

$$e : X \times X \rightarrow \mathbb{R}^+$$

$e(x, y)$  is the error in the answer  $x$  assuming  $y$  is the correct answer.

$$\forall x \in X, e(x, x) = 0$$

When checked against itself, an output  $x$  has an error of zero.

# IO-FUNCTION (1)

$$f_a : \Omega \times \mathbb{R}^+ \rightarrow X \times \tau$$

$a \in \mathcal{A}$  is an algorithm.

$\tau = \{0, 1\}$ , indicating whether the algorithm  $a$   
reached its final output.

$f_a(\omega, t) = (x, b)$  means that algorithm  $a$  applied to input  $\omega$   
yields the result  $x$  after running time  $t$  together with the  
information  $b$  indicating whether the algorithm reached  
its final output.

## IO-FUNCTION (2)

$f_a(\omega, t_1) = (x, 1)$  implies  $f_a(\omega, t_2) = (x, 1)$  for  $t_2 \geq t_1$

If algorithm  $a$  reaches its final output  $x$  by time  $t_1$ , then for any time  $t_2$  after  $t_1$  the output does not change.

$f_a^{res}(\omega, t) = x$  and  $f_a^{term}(\omega, t) = b$  if  $f_a(\omega, t) = (x, b)$

$f_a^{res}$  gives only the result/output of the IO-function while  $f_a^{term}$  gives only the indicator of whether the result/output is the final output.

## CORRECT OUTPUT FUNCTION

$$f_0 : \Omega \rightarrow X$$

$f_0(\omega)$  provides the correct output  $x$  for the given input  $\omega$ .

$$e(x, f_0(\omega)) = 0 \text{ iff } f_0(\omega) = x$$

If the error of an output  $x$  compared to the correct output  $f_0(\omega)$  is zero, then the correct output  $f_0(\omega)$  is  $x$ , and vice versa.

# RUNTIME FUNCTION

$$\rho_a : \Omega \rightarrow \mathbb{R}_\infty^+$$

$$\rho_a(\omega) = \inf\{t \mid f_a^{term}(\omega, t) = 1\}$$

$\rho_a(\omega)$  gives the smallest time at which the algorithm  $a$  produces its final output given input  $\omega$ .

# ONE-ANSWER ALGORITHMS

$$\mathcal{A}_1 = \{a \mid \forall \omega \in \Omega, \rho_a(\omega) < \infty \wedge \forall t < \rho_a(\omega), f_a^{res}(\omega, t) = \perp\}$$

A one-answer algorithm is an algorithm that arrives at its final output in finite time, and given any shorter amount of time, it results in no output.

# DEFECT ASSOCIATED WITH AN ALGORITHM AT A TIME POINT

$$\delta : \mathcal{A} \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$$

$$\delta(\mathbf{a}, t) = \sum_{\omega \in \Omega} e(f_{\mathbf{a}}^{\text{res}}(\omega, t), f_0(\omega)) P(\omega)$$

$\delta(\mathbf{a}, t)$  denotes the defect associated with an algorithm  $\mathbf{a} \in \mathcal{A}$  at a time point  $t$ . It is essentially the average error of a given algorithm for a given amount of computation time.

# ULTIMATE DEFECT and MORE PRECISE THAN

$$\delta : \mathcal{A} \rightarrow \mathbb{R}_{\infty}^{+}$$

$$\delta(a) = \limsup_{t \rightarrow \infty} \delta(a, t)$$

$\delta(a)$  denotes the defect associated with an algorithm  $a \in \mathcal{A}$  over an arbitrarily long amount of time.

For  $a, b \in \mathcal{A}$ ,  $a$  is said to be MORE PRECISE THAN  $b$  if  $\delta(a) \leq \delta(b)$ .

## AVERAGE RUNTIME and QUICKER THAN

$$\alpha(a) = \sum_{\omega \in \Omega} \rho_a(\omega) P(\omega)$$

$\alpha(a)$  gives the average runtime of an algorithm  $a$ .

For  $a, b \in \mathcal{A}$ ,  $a$  is said to be QUICKER THAN  $b$  if  $\alpha(a) \leq \alpha(b)$ .

# STRONGLY BETTER THAN

For  $a, b \in \mathcal{A}$ , we say that  $a$  is STRONGLY BETTER THAN  $b$   
if  $a$  is more precise than  $b$  and  $a$  is quicker than  $b$ .

# MORE PRECISE THAN AT TIME POINT and REALISES A DEFECTLESS APPROXIMATION

For  $a, b \in \mathcal{A}$ , we say that  $a$  is MORE PRECISE THAN  $b$   
AT TIME POINT  $t$  if  $\delta(a, t) \leq \delta(b, t)$ .

We say that  $a \in \mathcal{A}$  REALISES A DEFECTLESS APPROXIMATION if

$$\lim_{t \rightarrow \infty} \delta(a, t) = 0.$$

# ANYTIME ALGORITHM and MONOTONIC ANYTIME ALGORITHM

We say that an algorithm  $a \in \mathcal{A}$  is an ANYTIME ALGORITHM if it realizes a defectless approximation.

We say that it is a MONOTONIC ANYTIME ALGORITHM if it is an anytime algorithm and furthermore  $\delta(a, t)$  is monotonically decreasing in  $t$ , i.e. if  $\delta(a, \cdot) \searrow 0$ .

# BETTER THAN

$a$  IS BETTER THAN  $b$  if

$$\sum_{\omega \in \Omega} P(\omega) \int_0^T \left( e(f_a^{\text{res}}(\omega, t), f_0(\omega)) - e(f_b^{\text{res}}(\omega, t), f_0(\omega)) \right) \bar{f}(t) dt \leq 0$$

for  $T = \max\{\rho_a(\omega), \rho_b(\omega)\}$

$$\bar{f} : \mathbb{R}^+ \rightarrow \mathbb{R}^+$$

$\bar{f}$  is a density function used to weight the importance of  
different computation times.

## Composition (1)

$$c : (X \times \tau)^n \rightarrow X \times \tau$$

$c$  is an ORACLE ALGORITHM that combines a vector of outputs and yields a single output.

Consider algorithms  $a_i$  ( $i = 1, \dots, n$ ). The invocation of all  $a_i$  in parallel and the subsequent call of the oracle algorithm  $c$  yield a new algorithm  $a_{1,\dots,n,c}$  with IO-function

$$f_{a_{1,\dots,n,c}}(\omega, t) = c(f_{a_1}(\omega, t), \dots, f_{a_n}(\omega, t))$$

## Composition (2)

$$c((x_1, 1), \dots, (x_n, 1)) = (x, 1)$$
$$\text{for } x \in \{x_1, \dots, x_n\} \subseteq X$$

This constraint requires that  $c$  select an output from the algorithms of which it's composed if they all complete (i.e., come to their final output). This effectively constrains the runtime of  $a_{1,\dots,n,c}$ .

$$\rho_{a_{1,\dots,n,c}}(\omega) \leq \max\{\rho_{a_1}(\omega), \dots, \rho_{a_n}(\omega)\}$$

# Combining One-Answer Algorithms into an Anytime Algorithm

Let  $a_1, \dots, a_{n-1}$  be one-answer algorithms and  $a_n$  be a (always terminating) sound and complete algorithm. Let  $c$  be such that

$$c(f_{a_1}(\omega, t), \dots, f_{a_{n-1}}(\omega, t), f_{a_n}(\omega, t)) = (f_{a_n}^{res}(\omega), 1)$$

for  $t \geq \rho_{a_n}(\omega)$ . This ensures that the sound and complete result is used if available given time  $t$ . This means that  $a_{1,\dots,n,c}$  realizes a defectless approximation and is therefore an anytime algorithm.

## More Specific Combiner (1)

Let inputs consist of class descriptions  $C$  over some description logic  $L$ , and let outputs consist of pairs  $(A, B)$  of sets of (named) individuals.  $A$  consists of only individuals belonging to the extension of  $C$  while  $B$  is guaranteed to contain all individuals belonging to the extension of  $C$ . Let  $a_1, \dots, a_n$  be sound but incomplete one-answer algorithms, and let  $b_1, \dots, b_m$  be complete but unsound one-answer algorithms. Additionally, let  $a$  be complete and sound.

## More Specific Combiner (2)

$$f_{a_1, \dots, a_n, b_1, \dots, b_m, a, c}(C, t) = \begin{cases} \left( (f_a^{res}(C, t), f_a^{res}(C, t)), 1 \right) & \text{for } t \geq \rho_a(C), \\ ((lower, upper), term) & \text{for } t < \rho_a(C) \end{cases}$$

$$\text{where } lower = \bigcup_{(A_i, B_i, 1) = f_{a_i}(C, t)} A_i,$$

$$upper = \bigcap_{(A_j, B_j, 1) = f_{b_j}(C, t)} B_j,$$

$term = 1$  if  $lower = upper$ , otherwise 0.

# Querying for Class Instances in Wine Ontology

- ▶ Reasoning of SHIQ description logic.

# Querying for Class Instances in Wine Ontology

- ▶ Reasoning of SHIQ description logic.
- ▶ Three one-answer algorithms used: SCREECH-ALL (complete but unsound), SCREECH-NONE (sound but incomplete), and KAON2 (sound and complete).

# Querying for Class Instances in Wine Ontology

- ▶ Reasoning of SHIQ description logic.
- ▶ Three one-answer algorithms used: SCREECH-ALL (complete but unsound), SCREECH-NONE (sound but incomplete), and KAON2 (sound and complete).
- ▶ The query space is the set of named classes (e.g, Chardonnay, StEmilion, Grape).

# Querying for Class Instances in Wine Ontology

- ▶ Reasoning of SHIQ description logic.
- ▶ Three one-answer algorithms used: SCREECH-ALL (complete but unsound), SCREECH-NONE (sound but incomplete), and KAON2 (sound and complete).
- ▶ The query space is the set of named classes (e.g, Chardonnay, StEmilion, Grape).
- ▶ All queries given equal probability of being asked.

# Querying for Class Instances in Wine Ontology

- ▶ Reasoning of SHIQ description logic.
- ▶ Three one-answer algorithms used: SCREECH-ALL (complete but unsound), SCREECH-NONE (sound but incomplete), and KAON2 (sound and complete).
- ▶ The query space is the set of named classes (e.g, Chardonnay, StEmilion, Grape).
- ▶ All queries given equal probability of being asked.
- ▶ Error function is inverted f-measure.

## Formalization

$$\mathcal{A} = \{KAON2, \text{SCREECH-ALL}, \text{SCREECH-NONE}\}$$

$$\Omega = \mathcal{C}$$

$$P(\omega) = \frac{1}{|\mathcal{C}|} \quad \text{for all } \omega \in \Omega$$

$$e(I, I_0) = 1 - \frac{2 \cdot \textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

$$e(\perp, I_0) = 1$$

$$\textit{precision} = \frac{|I \cap I_0|}{|I|}$$

$$\textit{recall} = \frac{|I \cap I_0|}{|I_0|}$$

$$f_a(C, t) = (\emptyset, 0) \text{ if } t < \rho_a(C), \text{ otherwise } (I_C, 1)$$

# Defect Over Time

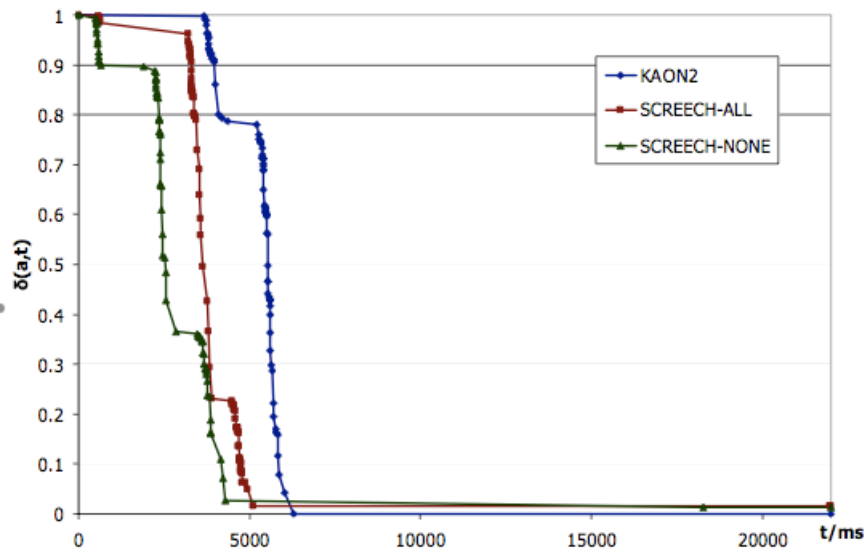


Fig. 1. Defect over time.

# Conclusions

- ▶ A mathematical framework is presented for assessment and comparison of approximate reasoning algorithms with respect to correctness, runtime, and anytime behavior.

# Conclusions

- ▶ A mathematical framework is presented for assessment and comparison of approximate reasoning algorithms with respect to correctness, runtime, and anytime behavior.
- ▶ In most practical cases, it will be unfeasible to measure the whole input space. Use of statistical considerations or rough heuristics could be effectively used.

# Conclusions

- ▶ A mathematical framework is presented for assessment and comparison of approximate reasoning algorithms with respect to correctness, runtime, and anytime behavior.
- ▶ In most practical cases, it will be unfeasible to measure the whole input space. Use of statistical considerations or rough heuristics could be effectively used.
- ▶ Combination of algorithms can provide anytime reasoning and is made possible with parallelization. This approach could be conceived as a somewhat exotic approach to distributed reasoning.