Inference over strings

Boris Yangel

Outline

- Probabilistic programming
- Inference over strings: the idea
- Open problems

Probabilistic programming (PP): a reminder

- Specify probabilistic model by writing a sampler from the model
- Inference performed automatically based on the sampler code
- More powerful and flexible than graphical models
  - Ideally the language used to write samplers should be Turing-complete
- More natural for software engineers

PP: linear regression

```
float[,] x = GetFeatures();
float[] w = new float[x.GetLength(1)];
for (int m = 0; m < w.Length; ++m)
  w[m] = Gaussian.FromMeanAndVariance(0, 1);
float[] y = new float[x.GetLength(0)];
for (int n = 0; n < y.Length; ++n) {
  y[n] = Gaussian.FromMeanAndVariance(0, 0.1);
  for (int m = 0; m < w.Length; ++m)
    y[n] += w[m] * x[n, m];
}
Prob.Observe(y, GetOutcomes());
var distW = Prob.Infer(w);
```

Inference over strings

- Random variables in probabilistic models have mostly been of numeric types
- Strings are usually converted to a numeric representation: bags of words, HMMs, N-grams etc.
  - Such representations allow for very limited reasoning about string structure
  - Especially when using factorized approximations
- From probabilistic programming perspective, strings should be first-class citizens
  - Common string operations such as Substring, Concat etc. should be supported
  - Should be possible to combine different types within the same program

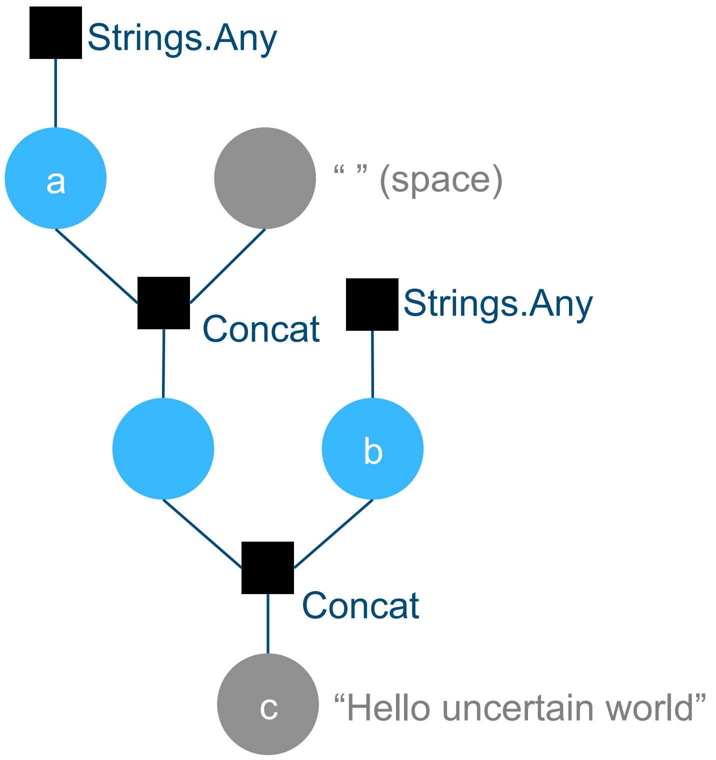
The goal

Provide first-class support for string variables and operations in our probabilistic programming framework (Infer.NET)

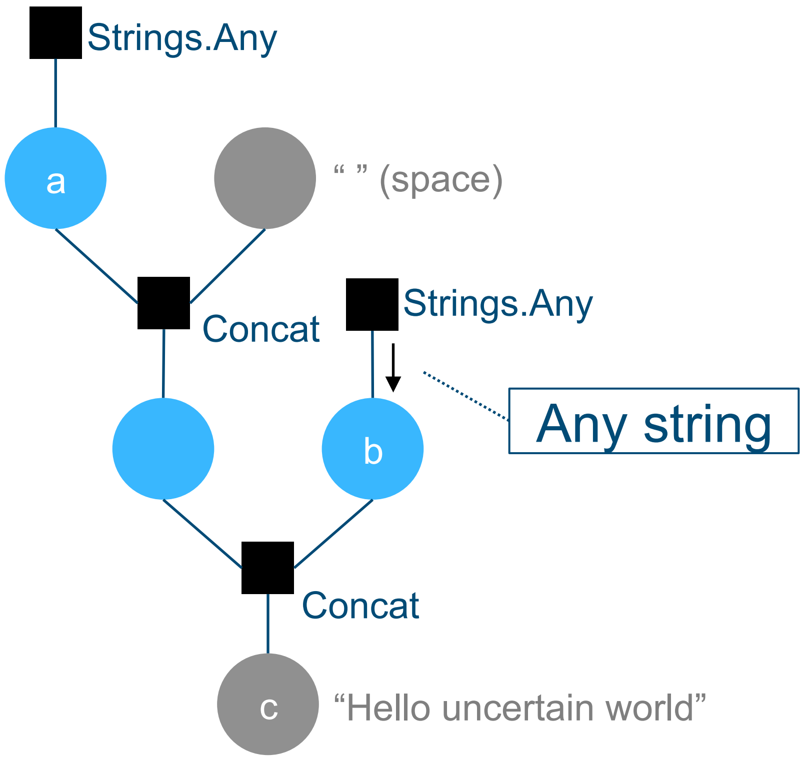
An example program

```
// Create two strings
string a = Strings.Any();
string b = Strings.Any();
// Format strings together into a new string
string c = a + “ ” + b;
// Observe result
Prob.Observe(c, “Hello uncertain world”);
// Infer one of the parts
var distA = Prob.Infer(a);
var distB = Prob.Infer(b);
```

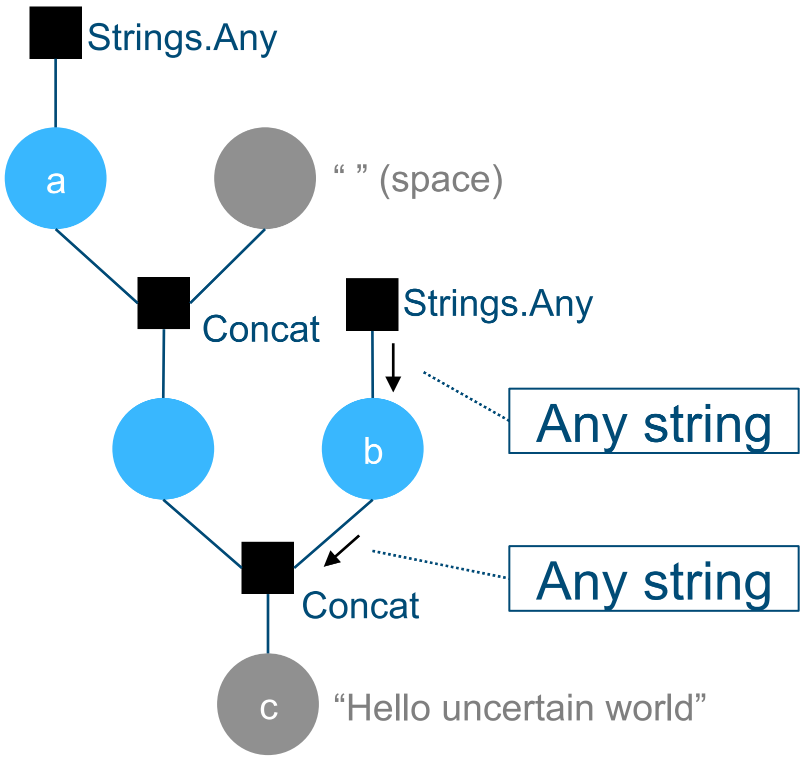
Inference via belief propagation



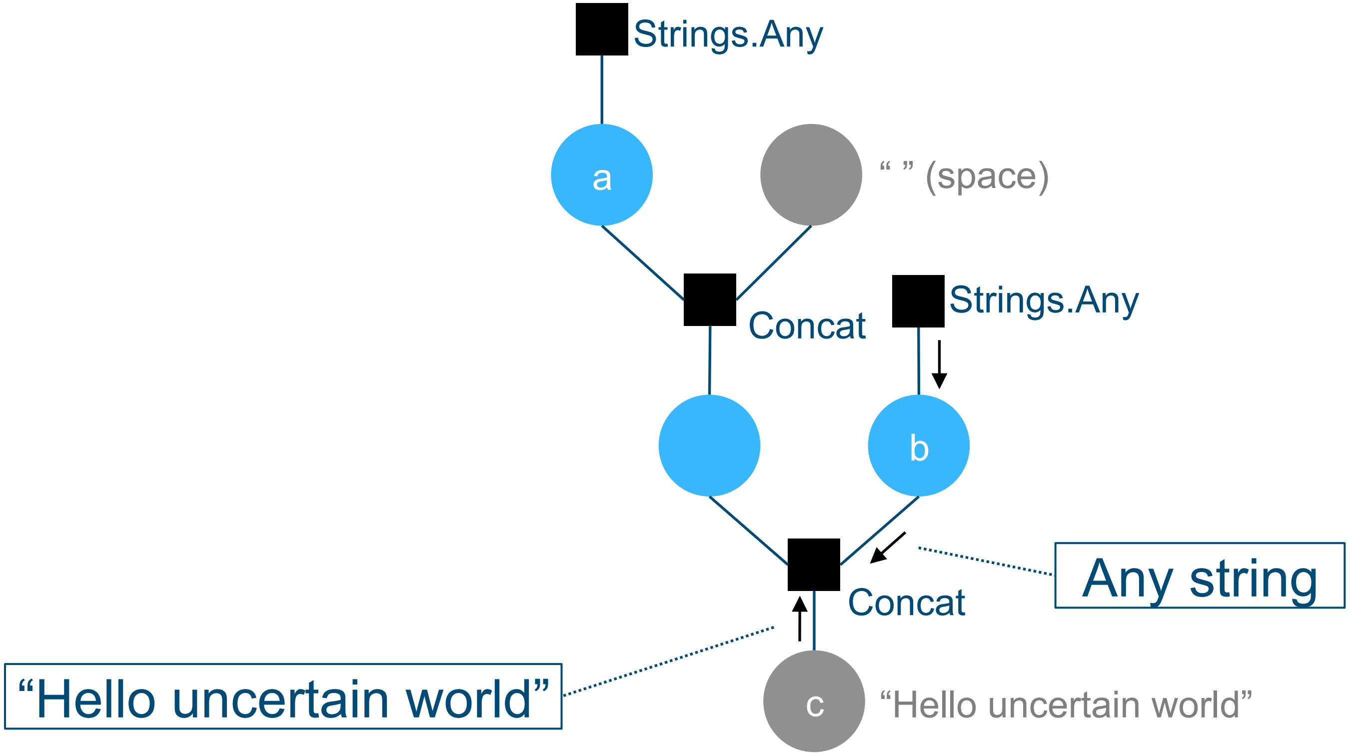
Inference via belief propagation



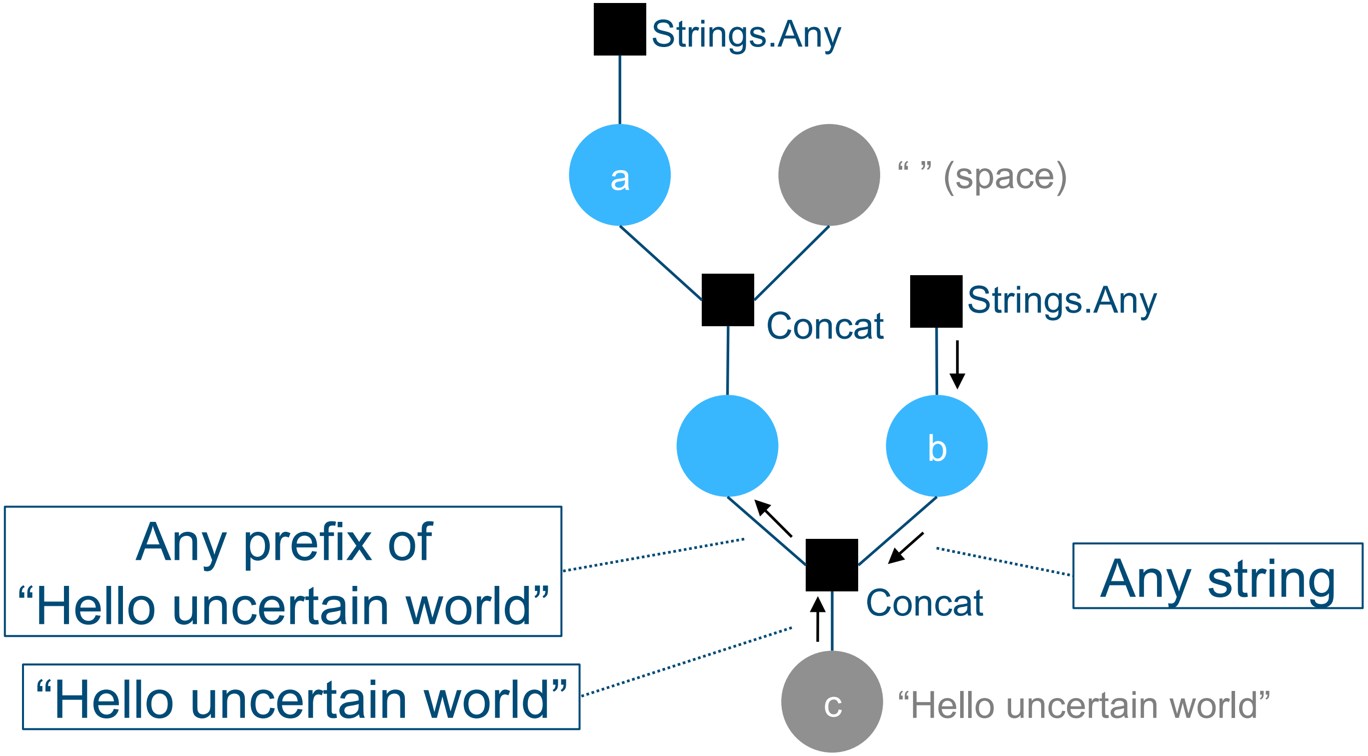
Inference via belief propagation



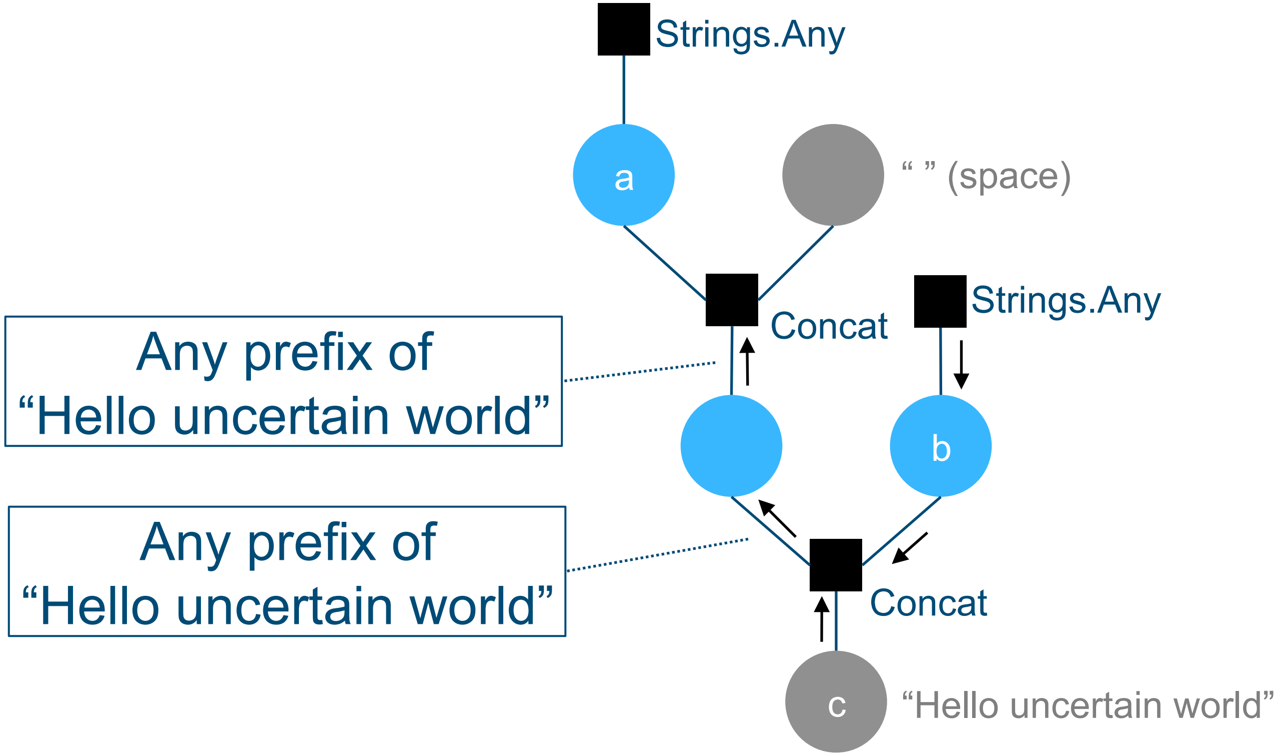
Inference via belief propagation



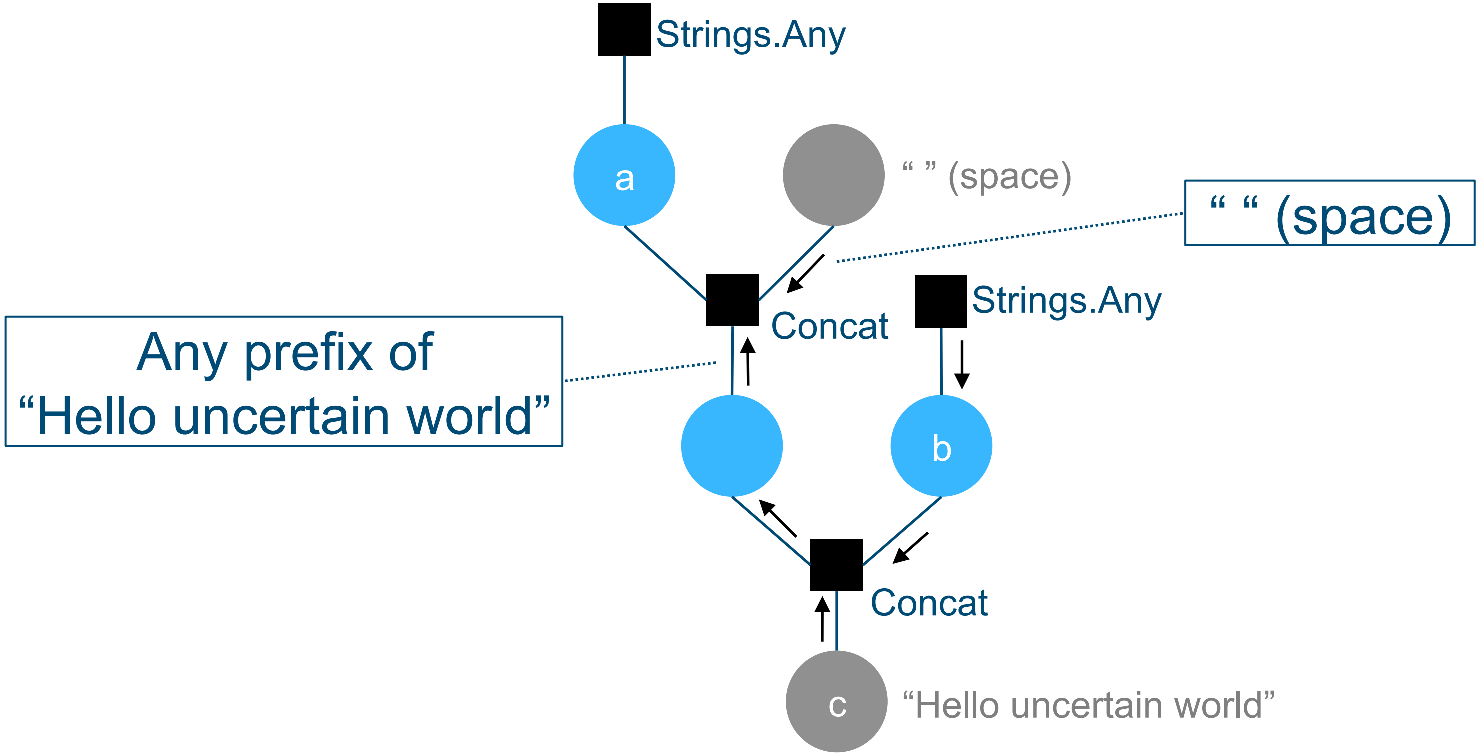
Inference via belief propagation



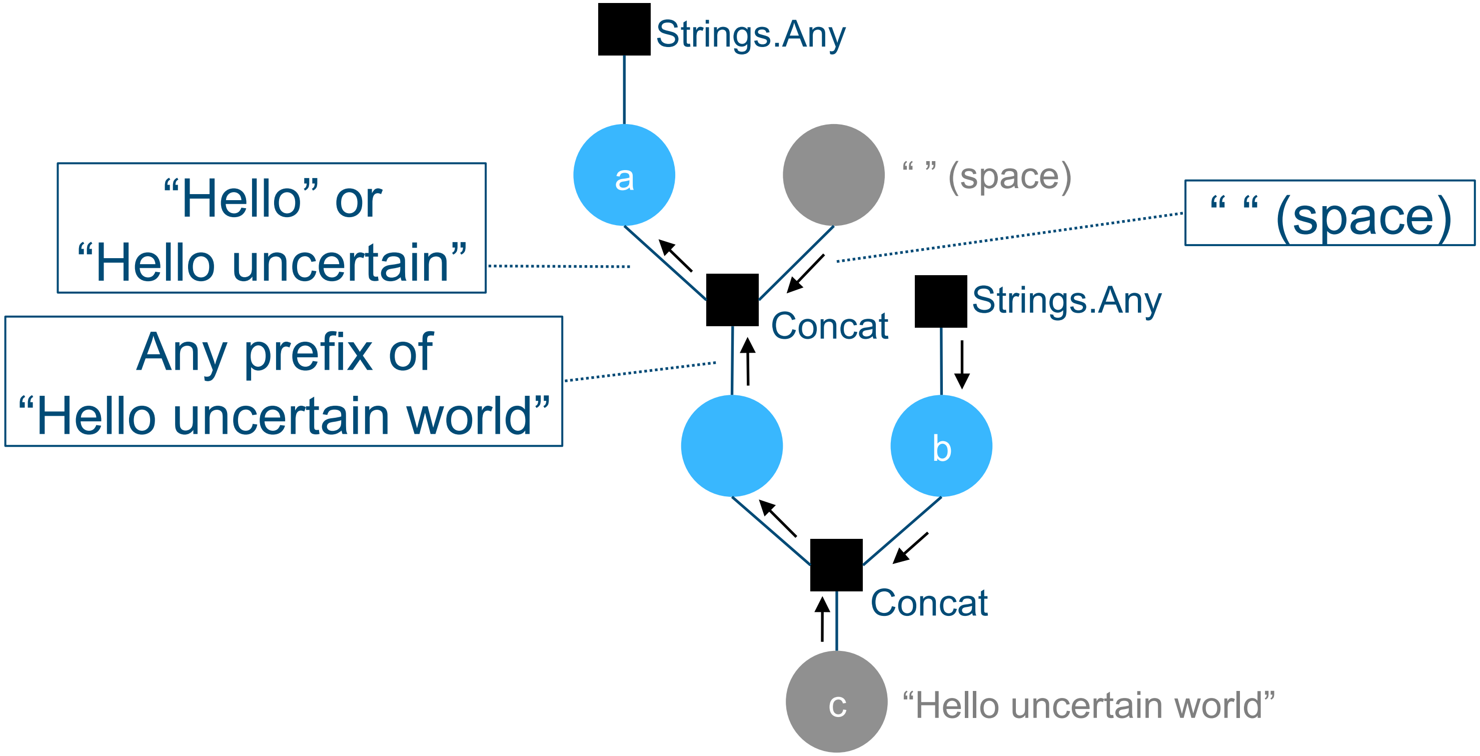
Inference via belief propagation



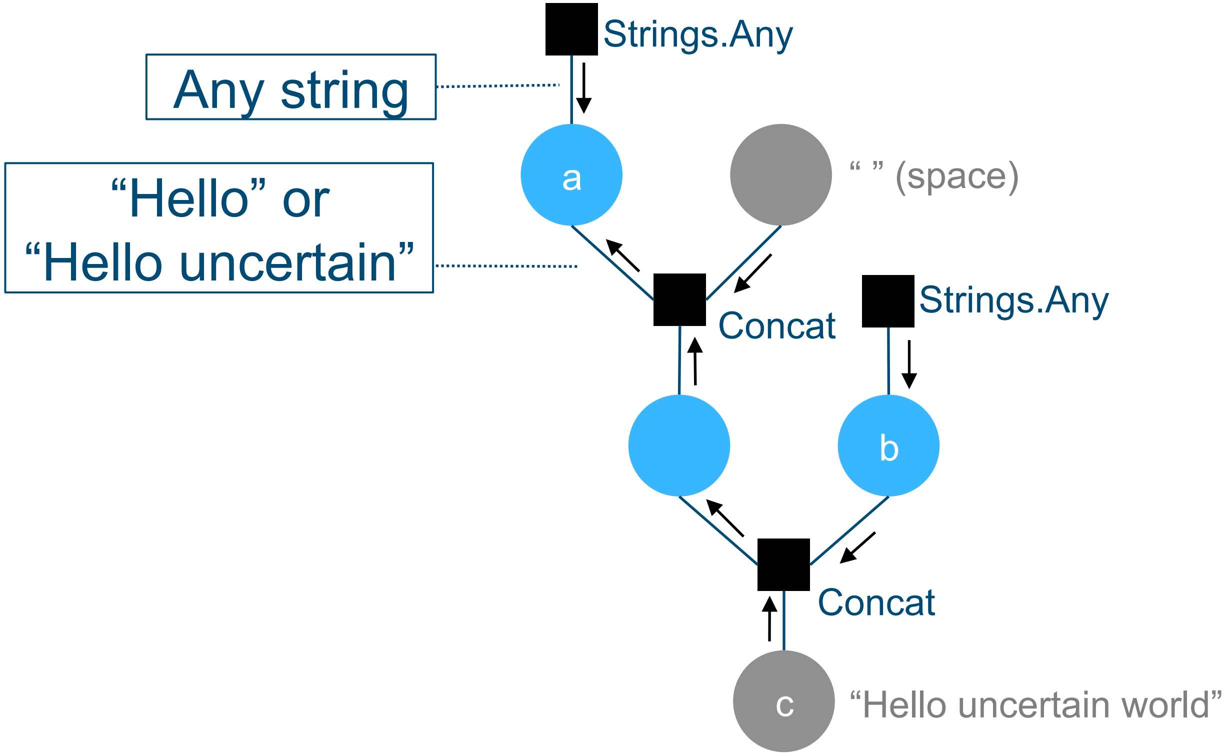
Inference via belief propagation



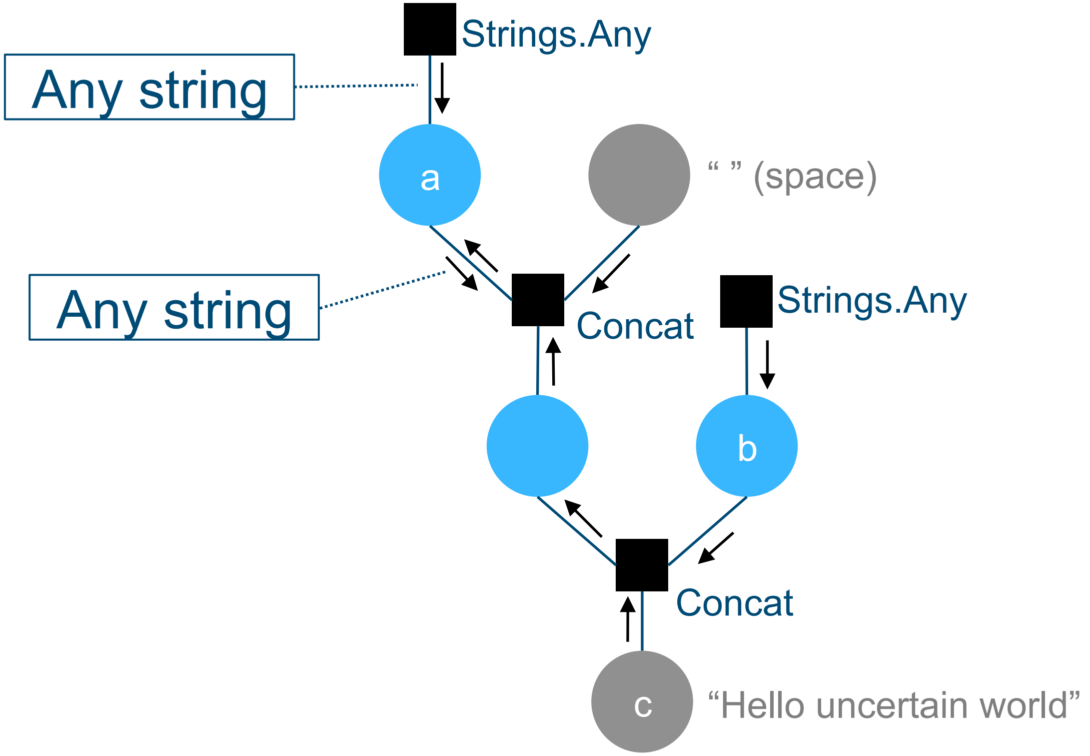
Inference via belief propagation



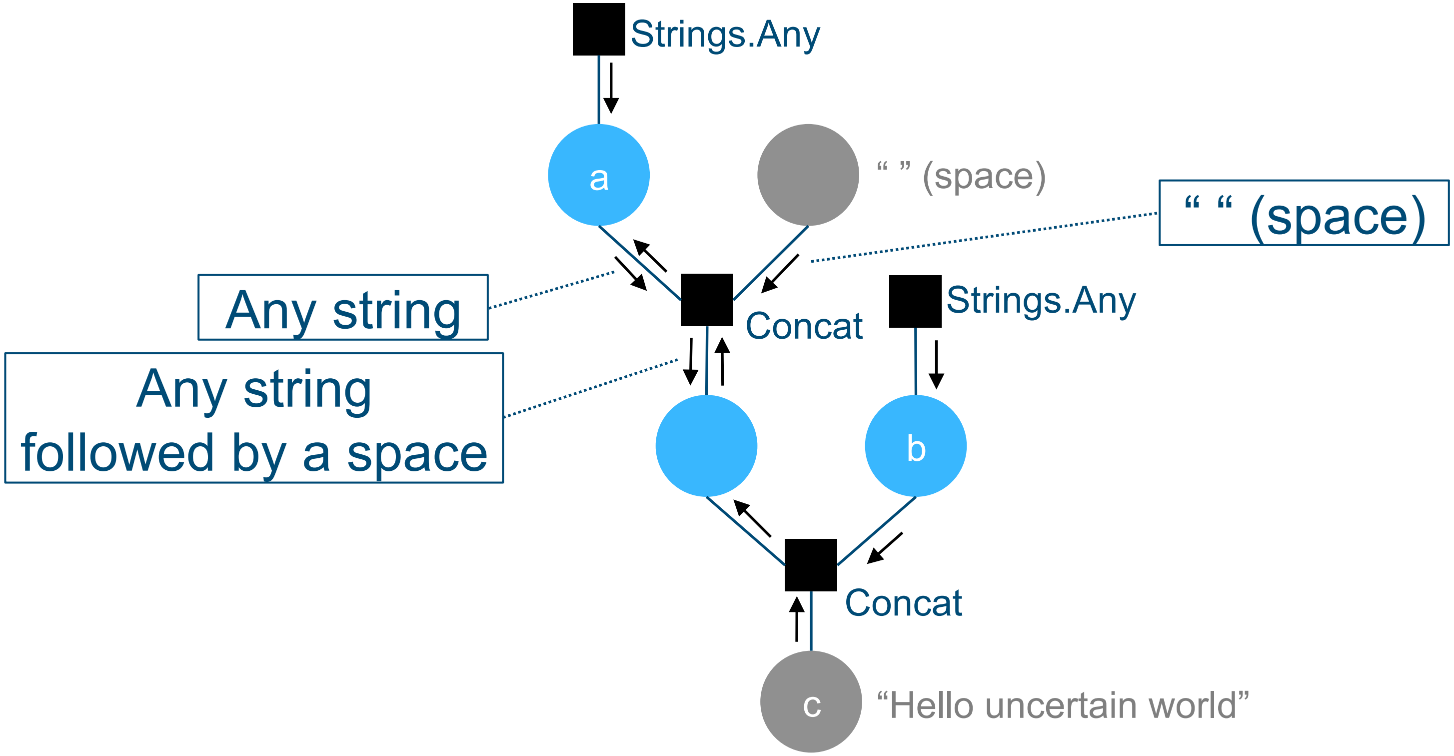
Inference via belief propagation



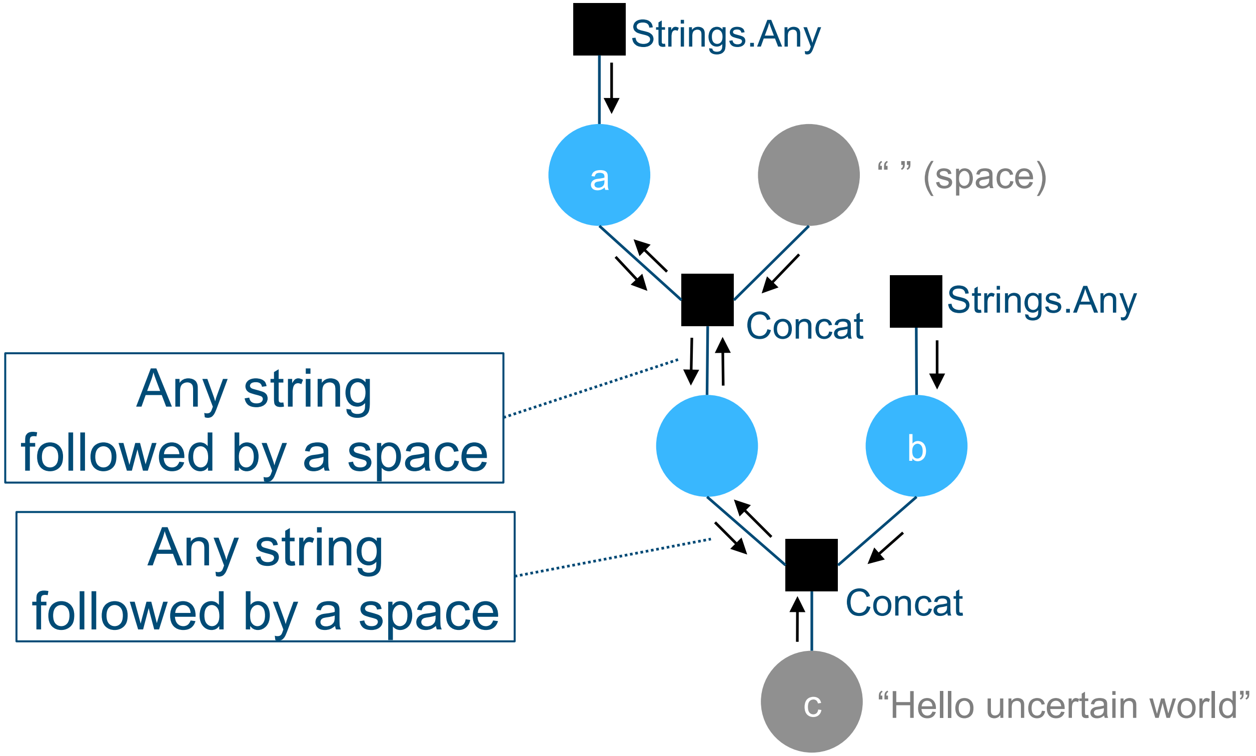
Inference via belief propagation



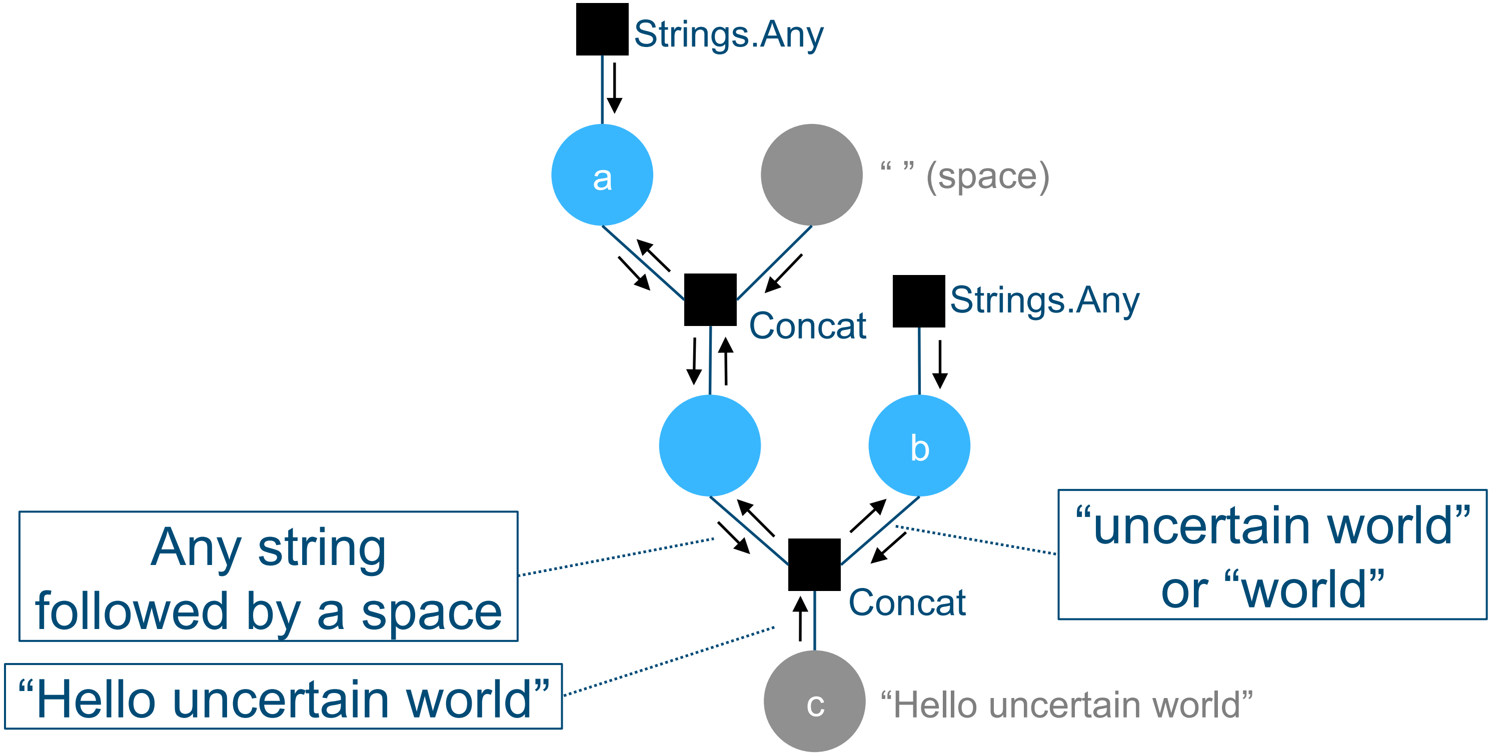
Inference via belief propagation



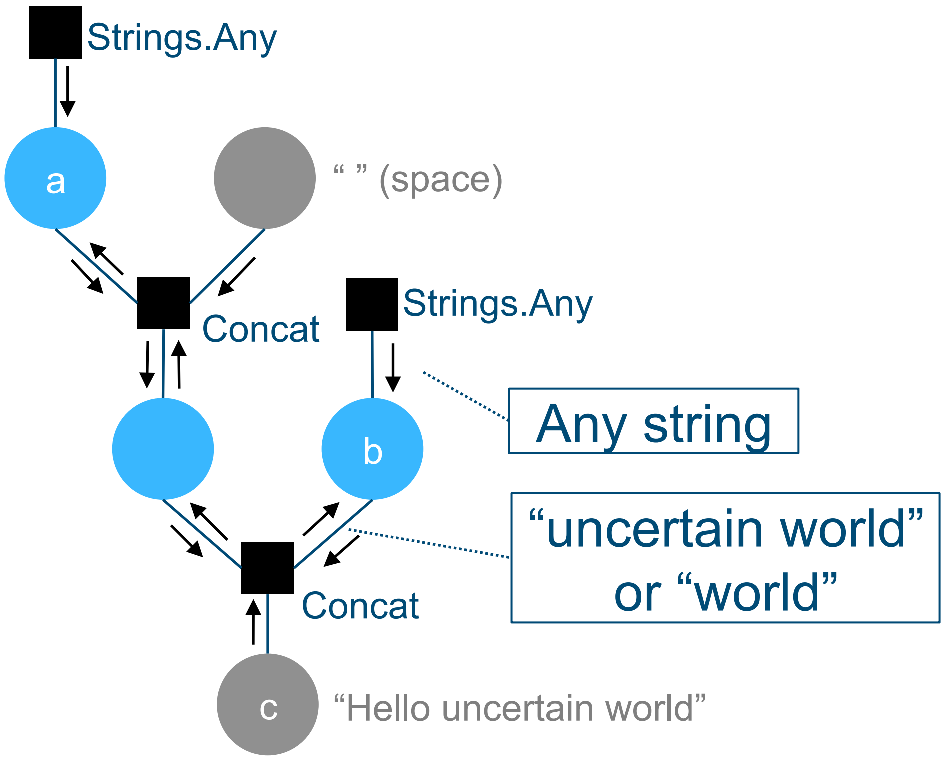
Inference via belief propagation



Inference via belief propagation



Inference via belief propagation



Representing beliefs

- Can we represent beliefs like <*any string*>, <*any prefix of a certain string*> etc. efficiently?
- Can we compute BP messages in this representation?

$$\mu_{f \to x}(x) = \sum_{X \setminus \{x\}} f(X) \prod_{x' \in X \setminus \{x\}} \mu_{x' \to f}(x')$$

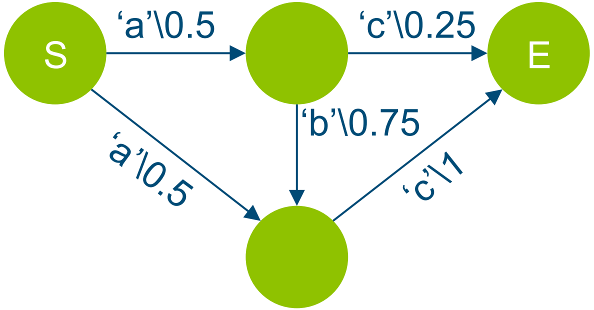
$$\mu_{x \to f}(x) = \prod_{f' \neq f} \mu_{f' \to x}(x)$$

- Requires support for sum and summing out variables
- Mixture models (gates) also require support for sum

Weighted finite state transducers (WFST)

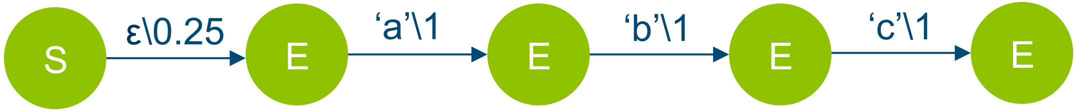
- N-way functions mapping sequences to real numbers
- 1-way transducers known as weighted finite state automata (WFSA)
- Can be normalized to represent distributions
- Closed w.r.t. sum, product and summing out a variable
- All these operations are efficiently computable!

Examples of WFSA



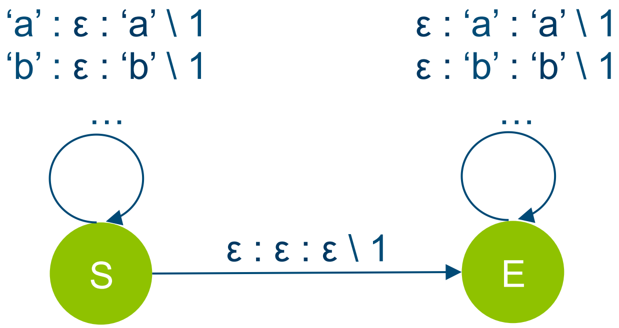
f("")=f("a")=f("aa")=…=1

f("ac")=0.5*0.25 + 0.5*1=0.625
f("abc")=0.5*0.75*1=0.375



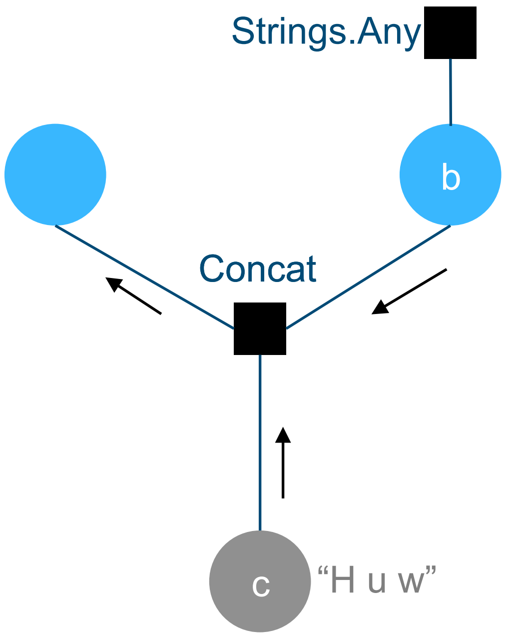
f("")=f("a")=f("ab")=f("abc")=0.25

An example of WFST

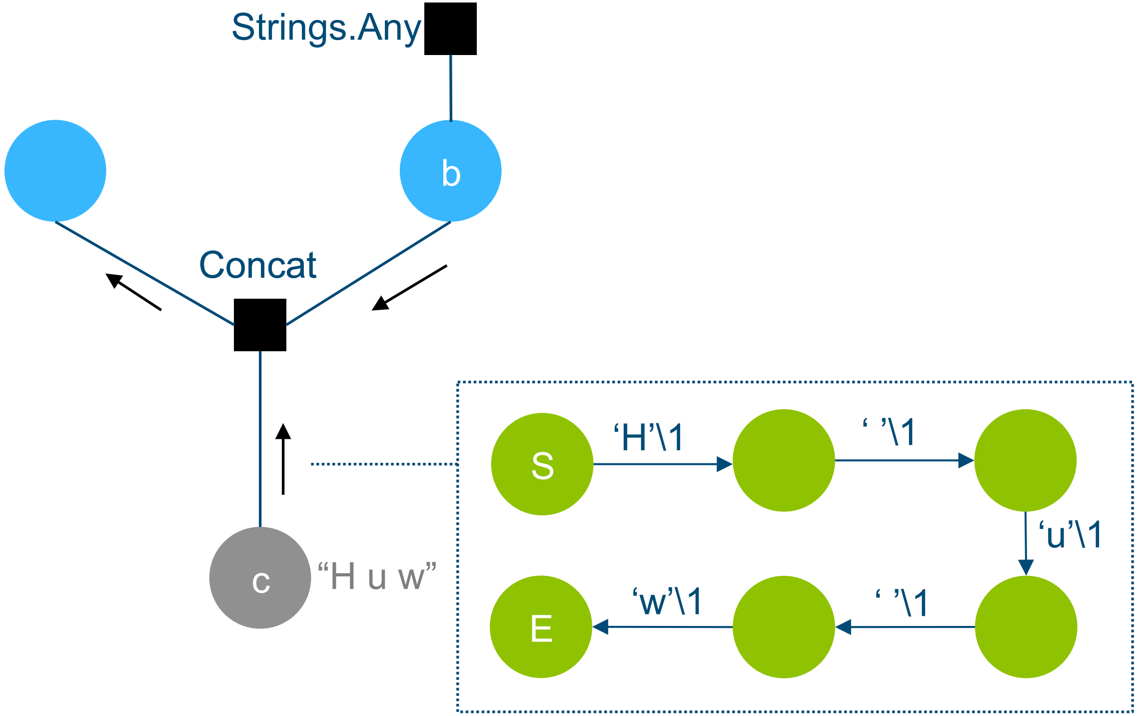


Concatenation:
f(a, b, c)=1 if c=ab, 0 otherwise

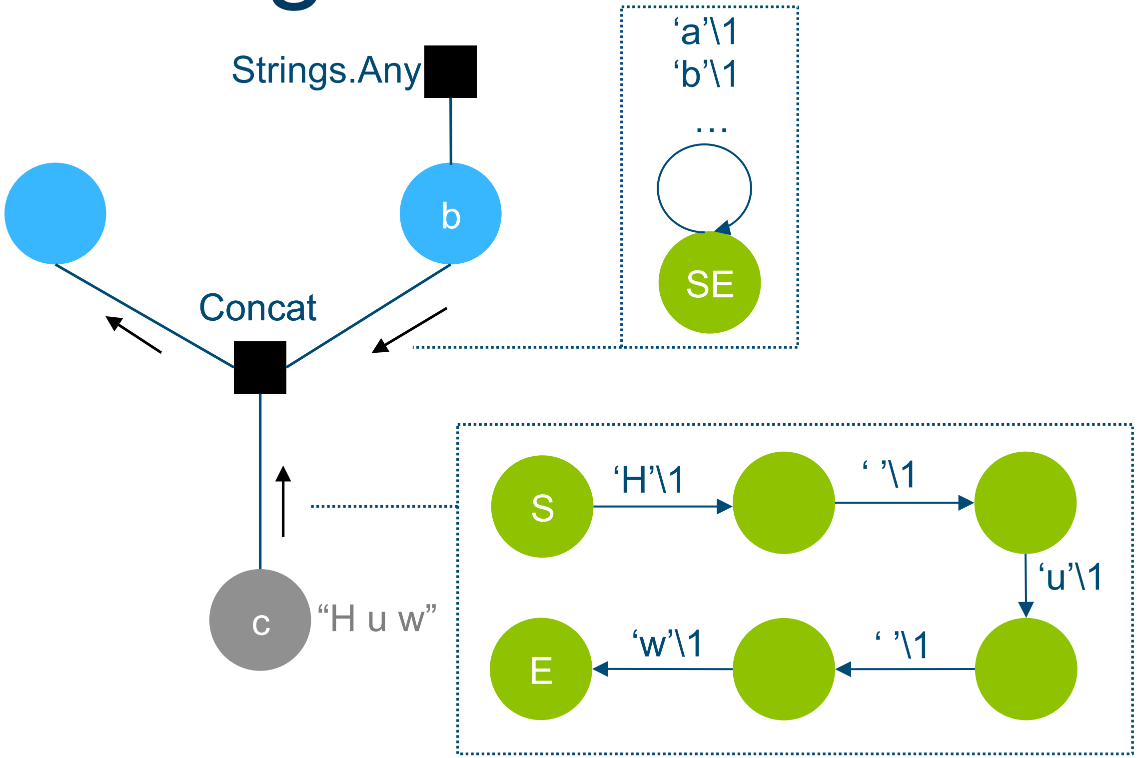
Message passing with WFST



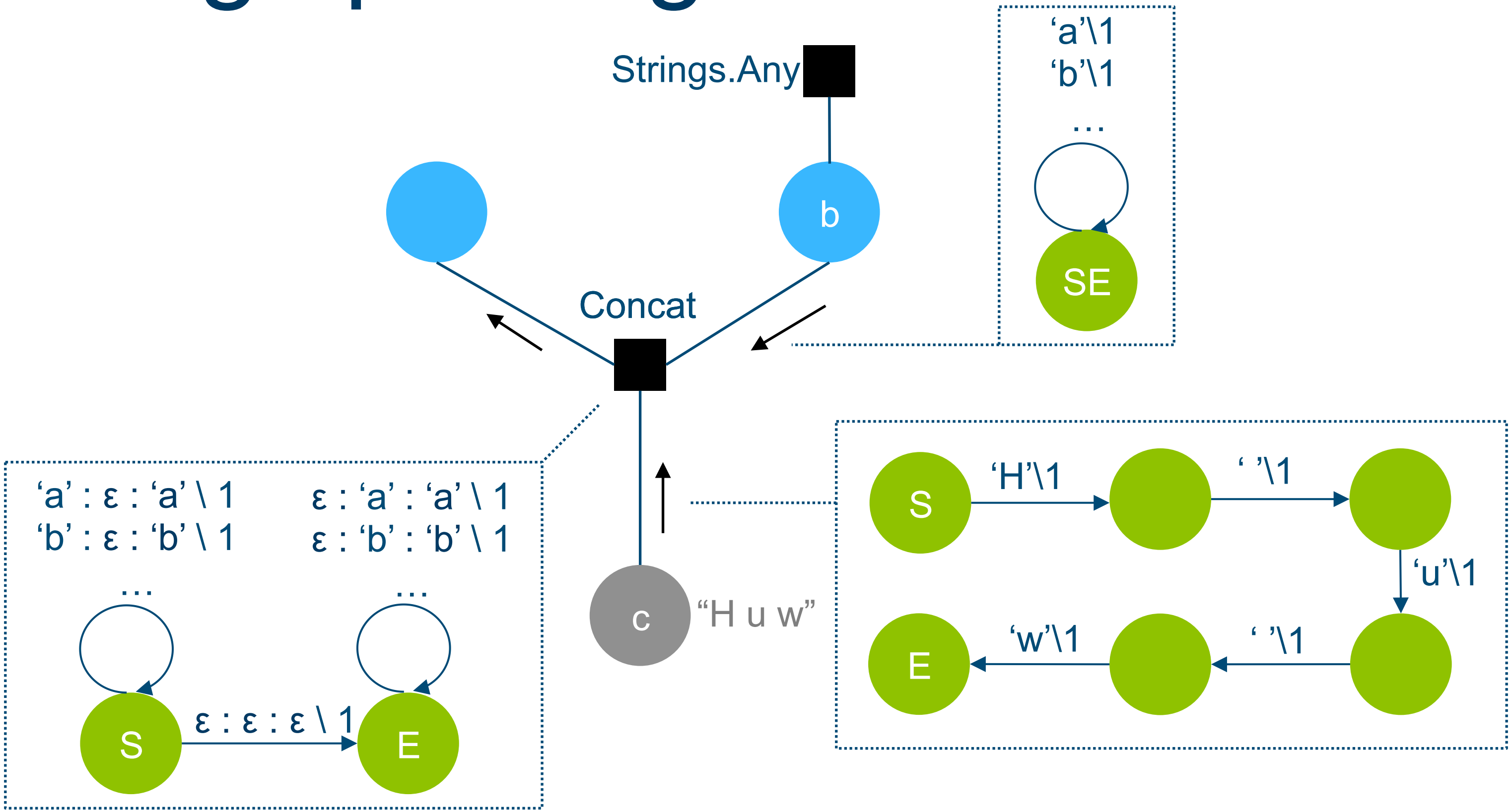
Message passing with WFST



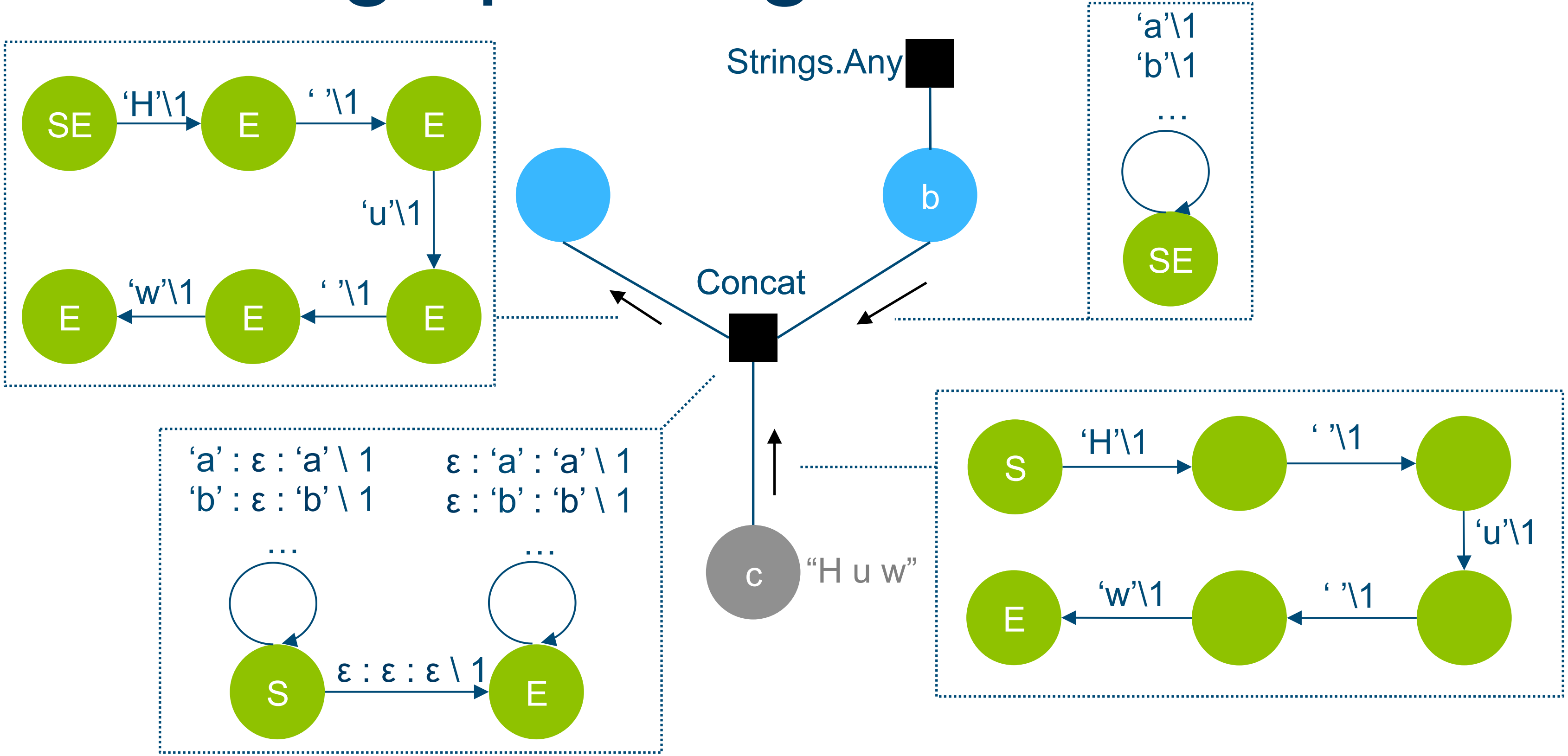
Message passing with WFST



Message passing with WFST



Message passing with WFST



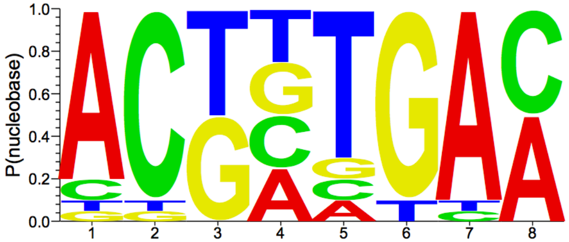
What do we have so far?

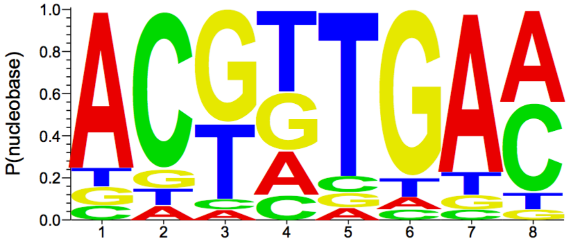
- Experimental support for automata-based distributions in Infer.NET
  - Efficient implementations of all basic operations: sum, product, projection, normalization, simplification etc.
  - A set of factors for commonly used string operations: Concat, Substring, Format, ArrayToString etc.
- Docs and tutorials
  - Motif finder in a few lines of code!
- A paper with more details
- Available in the latest (2.6) public Infer.NET release

Motif finder

```
var c = new Range(motifLen);
var motifProbs = Array<Vector>(c);
motifProbs[c] = Dirichlet(motifProbPrior);
var s = new Range(sequenceCount);
var motifPos = Array<int>(s);
var hasMotif = Array<bool>(s);
var sequences = Array<string>(s);
using (ForEach(s)) {
  motifPos[s] = DiscreteUniform(
    seqLen - motifLen + 1);
  hasMotif[s] = Bernoulli(hasMotifProb);
  using (If(hasMotif[s])) {
    var motifChars = Array<char>(c);
    motifChars[c] = Char(motifProbs[c]);
    var motif = StrFromArray(motifChars);
    var bgL = StrOfLen(motifPos[s],bgDist);
    var bgR = StrOfLen(
      seqLen - motifLen - motifPos[s],
      bgDist);
    sequences[s] = bgL + motif + bgR;
  }
  using (IfNot(hasMotif[s])) {
    sequences[s] = StrOfLen(
      seqLength, bgDist);
  }
}
```

	Phas	*Pmode*	*Ptruth*
AGTTTGCGTTGAACCGCGTGATATA	0.95	0.95	
CCTGGGGGCCGCAATACGGCTGAAC	0.32	0.35	
CTACTTTGACCATGCAACACTCAGG	0.99	0.99	
GTTGGTCATAATGGAACTTGGTCGG	0.62	0.64	
GATTAGAAATTATCAACCCTCTGTT	0.22		
AGCCGTTGTTGTATTTCGAGGTCGC	0.29		
GCTGTTCGGTTACGGTGAAATCACA	0.99	0.99	
TTAGGACAGTCGTTTGTTAGTCGCG	0.07		
GGAAGTTTGATGTAGCTACGGTGAA	0.98	0.97	
AGCGAACTCGACGGTGACGTGTTAC	0.99	0.99	
GCAAGTACTCCGGCGCATATAAGCA	0.05		
CTGGGTACTGCTTTCTTCGTTGACG	0.96	0.96	
TCGGAATTCCTGTGATGCGGTTGAA	0.63	0.47	0.20
TTCCGATCACAATAATACTTTGAAG	0.99	0.99	





Open problems

- Mixing variable types
- Messages can become too large
  - redundancy in the representation of beliefs
  - exactness of belief representation
  - inability of WFSA to represent certain beliefs efficiently
- Improper beliefs can arise during inference
  - and make computations ill-defined

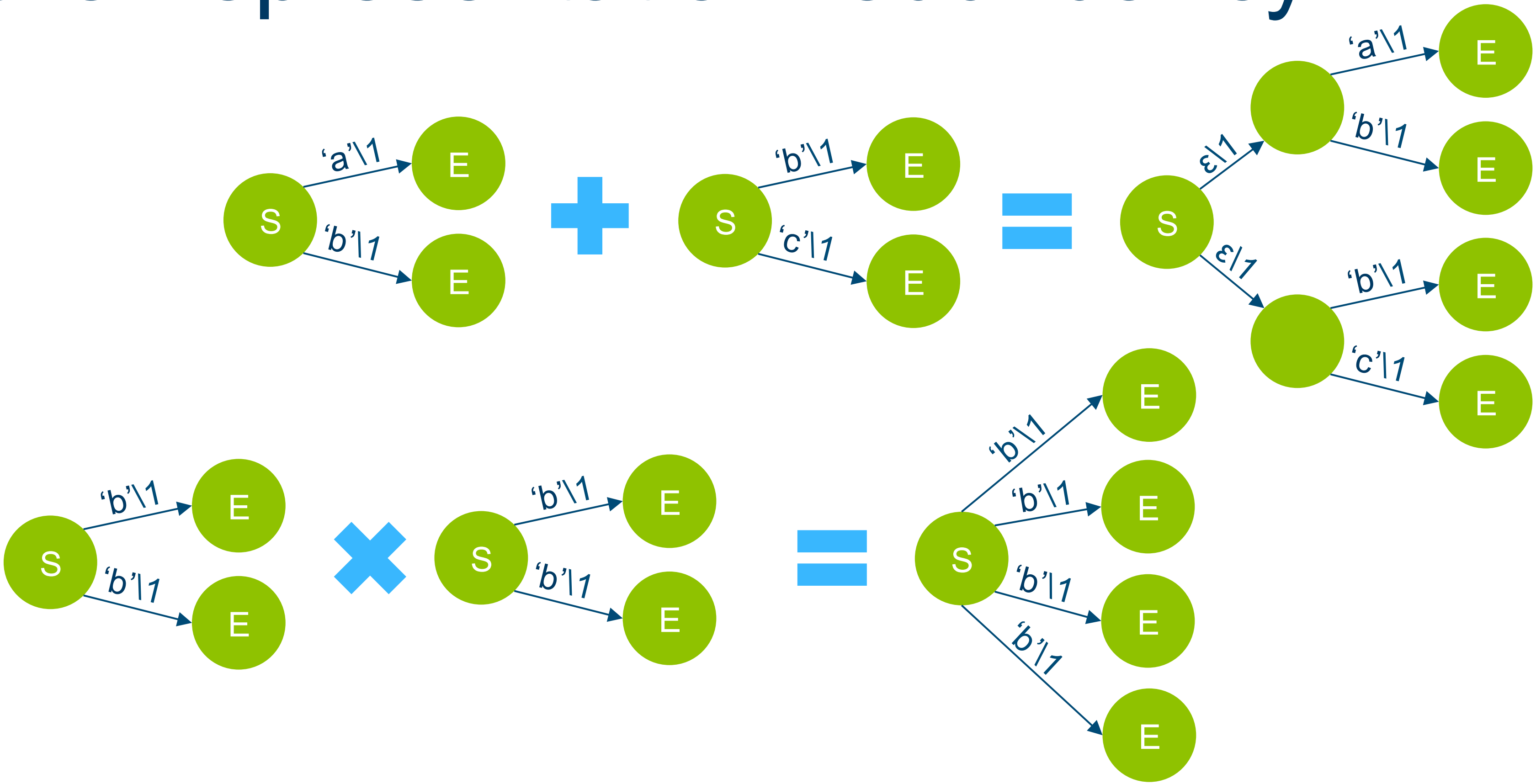
Mixing variable types

- What if a factor has non-string arguments?
  - `substr = string.Substring(str, pos, length)`
- Can usually sum out non-string variables and represent result as a transducer

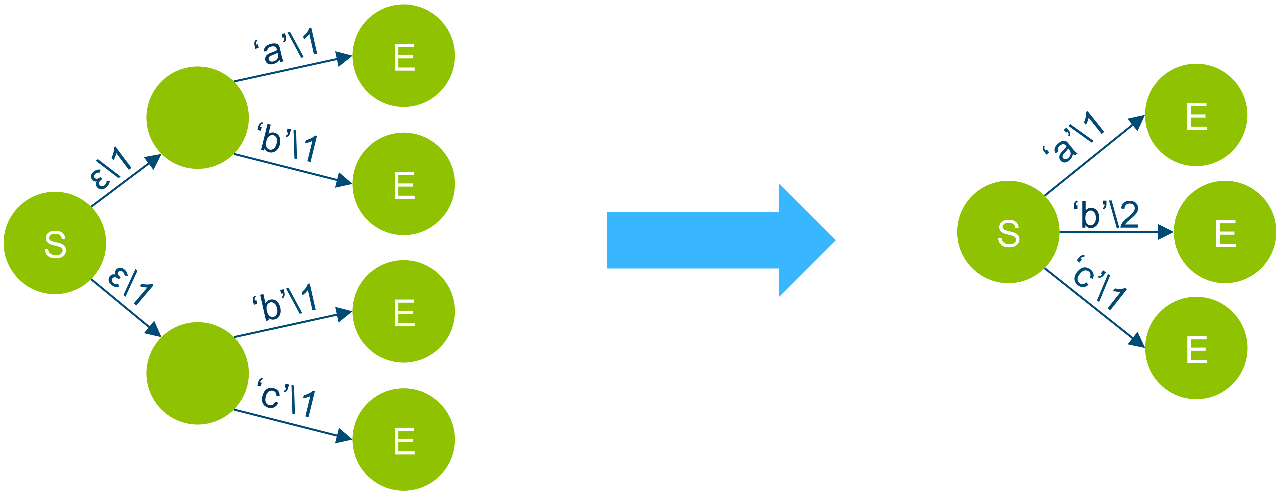
$$f'(str, substr) = \sum_{pos,length} f(str, substr, pos, length)\mu(pos)\mu(length)$$

  - Allows computing messages to string variables
  - Each case requires special treatment for efficiency
- Computing messages into non-string variables
  - Custom logic for each factor so far

Belief representation redundancy



Determinization



Determinization

- Cannot always be applied if an automaton has loops
  - Unlike the non-weighted case
- Possible alternatives
  - Partial determinization
  - Disambiguation [Mohri, 2012]
  - Approximate determinization [Boker et. al., 2012]

Exactness of belief representation

- If there are no loops, marginals are computed exactly
- Each observed data point can potentially influence marginals
- Marginals might “remember” the training set
- Potential solution: EP instead of BP
  - Project messages onto a simpler family (limited number of states)
  - State merging [Carrasco et al., 1997], [Thollard et al., 2000]

Inefficient representation of beliefs

- What can be said about the result of

```
string.Format(template, arg0, arg1, ..., argN)
```

if `template` mentions each argument once?

- Can be any permutation of args with text in between!
- Requires O(|arg|N!) states to represent
- Using EP instead of BP might also help here
  - But projection needs to be built on-the-fly

Improper distributions

- Can easily arise during inference
  - Observe that a string starts with an “A”
- Make some message computations ill-defined
  - Sum of factor times message over a variable might diverge
  - Especially affect models with gates
- A possible solution: constrain maximum string length
  - Not clear how to do automata calculus efficiently with this constraint
  - Maybe some good approximations can be made

Questions?

References

- Thollard, Franck, Pierre Dupont, and Colin de la Higuera. **"Probabilistic DFA inference using Kullback-Leibler divergence and minimality."** *ICML*. 2000.
- Carrasco, Rafael C. **"Accurate computation of the relative entropy between stochastic regular grammars."** *ITA* 31.5 (1997): 437-444.
- Mohri, Mehryar. **"A disambiguation algorithm for finite automata and functional transducers."** *Implementation and Application of Automata*. Springer Berlin Heidelberg, 2012. 265-277.
- Boker, Udi, and Thomas A. Henzinger. **"Approximate determinization of quantitative automata."** *LIPIcs-Leibniz International Proceedings in Informatics*. Vol. 18. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.