Programs = Logic + constraints + probabilities

Programs = Algorithms + Data-Structures
Algorithms = Logic + Control

Vijay Saraswat
IBM TJ Watson
Francesca Rossi
U Padua and Harvard University

Jan 2015
Era of Big Data (…. + Constraints…?)

- We are awash in data
  - Clustering algorithms
  - Regression
  - Frequent item set mining
- With open-ended set of applications
  - Personalized recommendations
  - Multi-language translation
  - Picture tagging
  - Face recognition
  - Speech understanding…

- And many techniques for dealing with it
  - Deep learning!

- The unreasonable effectiveness of constraints
**Example application: Constrained Markov sequences**

- **Constraint-based music (text, ...) synthesis**

- **Analyze corpora to generate a Markov model**
  - Impose constraints (unary, binary, k-ary...)
    - E.g. meter constraints
    - E.g. max-order constraints – synthesized string should not have substrings of length $\geq K$ in the corpora
  - Problem: Generate sequences from the posterior probability distribution

---

*Pachet et al*
**Example: Constrained Clustering**

- **Problem:** Cluster a large number of “points” into K groups (unsupervised learning)
  - An iterative “map reduce” algorithm, Lloyd-Warshall, for computing K means.

- **But what if you want to handle constraints?**
  - Two points must (not) be in the same cluster
  - The maximum diameter of a cluster should be minimized
  - A cluster should have at least (most) p elements in it
  - ...
But how do you scale this out?

- **How do you do this without constraints?**
  - Distribute points across places, replicate current K means
  - Use iterative algorithm (Lloyd-Warshall)
  - How do you design a solver in such a way that it can discover this structure on its own…?

- **Idea: Glass-box solvers (cc(FD), ‘94)**

  ➔ Divide data across places, run a constraint solver at each place, communicate partial results, iterate to convergence (cf SatX10, DDX10, Adaptive Local Solver)

  ➔ How do you ensure convergence?
Constraint Programming

- Constraint Programming is programming with (probabilistic) partial information, using logic-based combinators (implies, and, or, some, and, recursion…)
  - *contra* functional programming

- Constraint Programming is for general-purpose application programming
  - Not just constraint solving, combinatorial problem solving
  - Includes conventional data-structures (rails, arrays, hashmaps, …, recursion), types, inferencing, sketching…

Thesis: Time has come to design a new general-purpose constraint programming language for probabilistic analytic applications involving big data.
Desiderata

- Intended for (probabilistic) analytic applications involving big data
  - cf new implementations of R, Matlab, DML
  - Implementation must exploit available intra-node parallelism and scale to multiple nodes in a cluster
  - Implementation must recover automatically from node-failure
  - In-memory scalability (contra disk-level scalability) is ok

- Must support high productivity
  - Must have high-level, declarative abstractions
  - Strongly type-checked, support type inference
  - Determinate by design (no race conditions, deadlock)
  - Must not require explicit concurrency or distribution constructs (annotations are ok)
  - Must support high-level (declarative) debugging
(Concurrent) Constraint Programming

(Agents) $A ::= c \mid \text{if } G \ A \mid A;A \mid X^A \mid D$

(Goals) $G ::= c \mid G;G \mid X^G$

- Discrete, continuous time ('93-'96)
- Defaults ('96)
- Probabilistic Computation ('96, '97)
- Recursive goals ('05)
- Optimization (functions)
- Preferences

CCP as a powerful, declarative, constraint-based framework for probabilistic programming
Significant amount of additional theory work

- **Proving properties of CC programs**  
  - de Boer, Gabbrielli, Marchiori, Palamidessi *TOPLAS 1997*; Etalli, Gabbrielli, Meo *TOPLAS 2001*

- **Abstract interpretation**  
  - Falaschi, Olarte, Valencia, PPDP 09; Falaschi, Olarte, Palamidessi, 2011

- **Declarative debugging**  
  - Fromherz, 1995

- **Abstract diagnosis**  
  - Comini et al, TPLP 2011, Titolo PhD thesis

See also survey paper: Gabbrielli, Palamidessi, Valencia “Concurrent and Reactive Constraint Programming”, 2010
Example: Clustering

// I(c) is the index of the representative of cluster c
val I = new IntExprRail(K, 0n, (N-1n), "I");

// G(i) is the index of the cluster to which i belongs*/
val G = new IntExprRail(N, 0n, (K-1n), "G");

// Require that i and j are in the same cluster.
def mustLink(i:XInt, j:XInt) { G(i) ~ G(j);} 

// Require that i and j cannot be in the same cluster.
def cannotLink(i:XInt, j:XInt) { G(i) !~ G(j);} 

/** Require the size of each cluster (i.e. the 
* number of entries in G which equal c) is no less than 
* v. */
def minSize(v:Int) throws ConstraintEx {
  for (c in 0n..(K-1n)) G.count(c) >= v;
}

def minDensity(v:Int, epsilon:XInt,count:XInt) throws ConstraintEx {
  for (i in 0n..(N-1n)) {
    val ns = IntExprRail.makeBoolVarRail(N, "for" + i);
    for (j in 0n..(N-1n)) {
      (Constraint.EQ(G(i),G(j)) &
       (d(P(i),P(j)) < epsilon) as Constraint).ifThenElse(
         Constraint.EQ(1n,ns(j)),
         Constraint.EQ(0n,ns(j)));
    }
    ns.count(1n) >= count; }
}

/** Points within delta of each other are in the same cluster. */
def minimizeMaxDiameter(delta:XInt) throws ConstraintEx {
  val D=new Int(0n,delta,"D");
  for (i in 0n..(N-1n)) for (j in i..(N-1n))
    Constraint.GT(d(P(i),P(j)), D) -> Constraint.NEQ(G(i), G(j));
  D.minimize();}
Probabilistic CCP (Concur 96, POPL 99)

- "External View": Add agent $X \sim \text{PD}$
  - Sampling interpretation: Sample PD to get a value $t$, add $X=t$ to store. Discard runs with inconsistent store. Report $c_1 \sim p_1 \& \ldots \& c_n \sim p_n$ (the $p_i$ are proportions).

- Conditioning agent $A$ with observations (=constraints on visible variables) $O$:
  - run $(A, O)$

- Marginalizing variable $X$:
  - Run $X^A$

MAP queries?

- Semantics: Once values are sample, program reduces to a CCP (= set of constraints $S$)
- Hence: denotation is a pair $(S, f)$, where $f$ is a probability lattice on $S$
  - (= map from $L(S)$, the free profinite lattice on $S$, to $[0, 1]$).
- All combinators definable, parallel composition is still set intersection;
  - POPL 99 handles the recursive case

- Generalizes appropriately to TCC
  - not yet worked out for HCC

Bayes’ networks

$$PD ::= v_1 \sim p_1 \ || \ ... \ || \ v_k \sim p_k$$

| switch(X) of {case v1: pv1; ...; case vk:pvk}

```scala
class BackAche {
   val True = Boolean.TRUE, False = Boolean.FALSE.
   def backache(Chair:Boolean, Sport:Boolean, Worker:Boolean, Back:Boolean, Ache:Boolean) {
      Chair ~ True~0.8||False~0.2,
      Sport ~ True~0.02||False~0.98,
      Worker ~ switch(Chair) {case True: True~0.9 || False~0.1;
         case False: True~0.01||False~0.99}
      Back ~ switch(Chair*Sport) {case (True, True): True~0.9 || False~0.1;
         case (True, False): True~0.2 || False~0.8;
         case (False, False): True~0.9 || False~0.1;
         case (False, False): True~0.01 || False~0.199},
      Ache ~ switch(Back) {case True: True~0.7 || False~0.3;
         case False: True~0.1||False~0.9}
   }
   agent run() {
      backache(True, True, Boolean("Worker"), Boolean("Back"), Boolean("Ache"))
   }
}
```

Bayes network represented as a conjunction of agent $X_i \sim P_{Di}$, one per node, existentially quantified over all latent variables.
Markov fields

Each factor (clique) represented as an object, with as many fields as variables, equality constraints to represent edges between cliques

AB ~ PD1,
BC ~ PD2,
AC ~ PD3,
CD ~ PD4,
AB.a = AC.a, AB.b=BD.b, AC.c=CD.c, BD.d=CD.d
(+ constraints on AB, AC, BD, CD as needed)

Conditional Random Fields…?
Problem on how to compute the normalization factor $Z(X)$ “naturally”
Propagating constraints on PDs

- What if we treat $X \sim PD$ as a constraint? Now store can make many inferences:
  - $X \sim \text{Uniform}(M,N), X \geq L \models X \sim \text{Uniform}(\max(M,L), N)$
  - $X \sim \text{Gaussian}(M_1, S_1^2), Y \sim \text{Gaussian}(M_2, S_2^2) \models X+Y \sim \text{Gaussian}(M_1 + M_2, S_1^2 + S_2^2)$

- And, in some cases (e.g. Bayesian networks), we can execute programs without sampling
  - $X \sim (a \sim p : b \sim (1-p)) \models \text{switch}(X)\{\text{case True: } pv1; \text{ case False: } pv2\} = X ? pv1 * p : pv2 * (1-p)$

Building powerful deductions into the constraint solver
Functions as set comprehensions

- **Key insight:** (partial) functions implicitly represent a bag (their range).
  - Functions with finite domains represent finite bags
  - Reduction operations (e.g. `max`) can be directly applied to functions
  - Functions with `groupby` clauses represent Rails (maps) of sets

```python
def delta(A: Vector, B: Vector) =
    max((i:A.domain)=>Math.abs(A(i)-B(i))).

def histogram(N:Int, A:Rail[Int(1,N)]) =
    sum ((i:A.domain)=> 1 groupby A(i)).
```
KMeans – a clustering algorithm

class KMeans(N:Int, P:Int, K:Int, pts:Rail[Vector(N)](P)) {
  type Vector = Rail[Double](N).
  def delta(A:Vector, B:Vector)=max(i=>Math.abs(A(i)-B(i))).

  def kmeans():Rail[Vector](K)={
    T = avg (i =>
      pts(i) groupby
      argmin(j => delta(pts(i), old(j))),
    means=delta(old, T) < epsilon? T: kmeans(T)
  }"}
Open Problems – Theory

- **Theory**
  - Develop declarative debugging for timed programs.
  - Develop extended static checking for CCP
  - Develop implementations of abstract interpretation for (T)CC (Falaschi et al)
  - Develop theory of determinate default programs
  - Integrate “soft constraints”, preferences into CCP theory.
  - Develop theory of “sketching” (another use for symbolic execution engine)

C10 is a very ambitious attempt to develop a modern constraint language. Please join us!
Background
X10
X10 2.2: An APGAS language

- Class-based single-inheritance OO
- Structs
- Closures
- True Generic types (no erasures)
- Constrained Types (OOPSLA 08)

```java
class HelloWholeWorld {
    public static def main(s:Array[String]) {
        finish
        for (p in Place.places())
            async
                at (p)
                    Console.OUT.println("(At " + p + ") " + s(0));
    }
}
```

Java-like productivity, MPI-like performance

- Basic model is now well established
- PPoPP 2011 paper shows best known speedup numbers for UTS upto 3K cores.
- Global Matrix Library shows substantial speedup over Hadoop for data analytics kernels.
- Similar performance improvement for Main Memory Map Reduce engine (M3R) over Hadoop.
- SATX10 – better than plingeling on 8 cores, significant perf improvement at 16,32,64,128 cores (x86 multicore, P7 cluster).
Selected Bibliography

- Saraswat, Rinard, Panangaden “Semantics of Concurrent Constraint Programming”, POPL 1991
- Falaschi, Gabbrielli, Marriott, Palamidessi “Compositional analysis for CCP”, LICS 1993
- Fromherz “Towards declarative debugging of CCP”, 1995
- de Boer, Gabbrielli, Marchiori, Palamidessi “Proving concurrent constraint programs correct”, TOPLAS 1997
- Etalli, Gabbrielli, Meo “Transformations of CCP programs”, TOPLAS 2001
- Falaschi, Olarte, Valencia “Framework for abstract interpretation for Timed CCP”, PPDP 09
- Gabbrielli, Palamidessi, Valencia “Concurrent and Reactive Constraint Programming”, 2010
- Recent PhD theses – Carlos Olarte (LiX) (universal TCC), Sophia Knight (LiX), Laura Titolo (U Udine) “Abstract Interpretation Framework for Diagnosis … of Timed CC languages”
HCC references