## Provenance Polynomials

**Unifying framework** (Green et al.) that captures the semantics of incomplete information and uncertain databases, query evaluation under set/bag semantics, annotation propagation for why- and how-provenance. In provenance polynomials, we denote provenance of input tuples by **variables**, a join of tuples by a **product** of their provenance, and a union of tuples by a **sum** of their provenance.

### Example Database

<table>
<thead>
<tr>
<th>Order</th>
<th>Store</th>
<th>Location</th>
<th>Operator</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 Print</td>
<td>s1</td>
<td>Depot1 Printer</td>
<td>s2</td>
<td>Joe抽烟1</td>
</tr>
<tr>
<td>02 Plotter</td>
<td>01 Printer</td>
<td></td>
<td>03 Plotter</td>
<td>Ink</td>
</tr>
<tr>
<td>04 Printer</td>
<td>s2</td>
<td>Depot2 Printer</td>
<td>05 Ink</td>
<td>Dan</td>
</tr>
<tr>
<td>06 Printer</td>
<td>s3</td>
<td>StoreA Ink</td>
<td>07 Plotter</td>
<td>Joe</td>
</tr>
</tbody>
</table>

### Example Query

\(<\text{Order} \land \text{Store} \land \text{Location} \land \text{Operator} \land \text{Item}\> = \{01, 02, 03, 04, 05, 06, 07\}

### Provenance Polynomial of the Query Result

\begin{align*}
\Phi_{Q} = & (s_1 + s_2)(s_1 + s_3) + s_2 s_3, \\
\Phi_{Q} = & (s_1 + s_2)(s_1 + s_3) + s_2 s_3, \\
\Phi_{Q} = & (s_1 + s_2)(s_1 + s_3) + s_2 s_3, \\
\Phi_{Q} = & (s_1 + s_2)(s_1 + s_3) + s_2 s_3.
\end{align*}

**Special cases:**
- **Boolean semiring** \((\mathbb{B}, \land, \lor)\): Each variable encodes the presence of its input tuple.
- **Semiring over natural numbers** \((\mathbb{N}, +, \cdot)\): Each variable encodes tuple multiplicity.
- **Bag semantics** positive queries: If the variables encode the tuples themselves, the provenance polynomial encodes the whole query result.

### Factorisation of Provenance Polynomials

**Algebraic factorisation** of \(\Phi_{Q}\):
\[
\Phi_{Q} = (a_1 + a_2)(s_1(a_1 + a_2) + s_2(a_1 + a_2)) + (a_3 + a_4)s_3s_4
\]

(expresses explicitly how groups of input tuples combine and thus shows the nested structure of the query result and its provenance.)

**Factorisations** can be more **informative** and **exponentially more succinct** than flat representations.

The monomials can be extracted from the factorisation with polynomial delay.

---

## Challenge: Queries with Factorised Polynomials of Bounded Size

**Classification of queries based on:**
- The minimal size of the factorised polynomials of query results for any input database.
- Result polynomials with factorisations of **bounded readability** for any input database.
  - Polynomial \(\Phi\) is **read-k** if each variable occurs at most \(k\) times in \(\Phi\).
  - Polynomial \(\Phi\) has **readability** \(k\) if \(k\) is the smallest number such that there is a read-k polynomial equivalent to \(\Phi\).

**Examples:**
- The readability of the query \[\text{Order} \land \text{Store} \land \text{Location} \land \text{Operator} \land \text{Item}\] is one for any database.
  - In our example, the factorised polynomial is \(\Phi_{Q} = (s_1 + s_2)(s_1 + s_3) + s_2 s_3\).
  - For each location, we get a product of sums of distinct variables.
  - The query \[\text{Order} \land \text{Store} \land \text{Location} \land \text{Operator} \land \text{Item}\] is dependent on the input database size.

### Challenge: Efficient Computation of Factorised Polynomials

- **Compute factorisations** of low/minimal readability for any polynomial.
- **Minimality** may be with respect to a restricted class of factorisations.
- For a query and a database, compute the factorised polynomial of the query result
  - without first computing the flat polynomial of the query result.

### Challenge: Querying Factorised Relations and Polynomials

- **Assume that variables** in polynomials carry the input tuples.
- **Evaluate queries** directly on factorised polynomials.

**Example:**
**Equivalent factorisations** of the result of \[\text{Order} \land \text{Store} \land \text{Location} \land \text{Operator} \land \text{Item}\] emp:
\[
\begin{align*}
\Phi_{Q} & = (a_1 + a_2)(s_1(a_1 + a_2) + s_2(a_1 + a_2)) + (a_3 + a_4)(s_1(a_1 + a_2) + s_2(a_1 + a_2)). \\
\Phi_{Q} & = (a_1 + a_2)(s_1(a_1 + a_2) + s_2(a_1 + a_2)) + (a_3 + a_4)(s_1(a_1 + a_2) + s_2(a_1 + a_2)).
\end{align*}
\]

Here, variables \(a_i\) are annotated with tuples from Order (Emp).

**\(\Phi_{Q}(\Phi_{\text{Emp}})\) is suitable for joining on Order (Emp) without unfolding.

### Challenge: Approximation by Factorised Polynomials

- **Given** a polynomial \(\Phi\), find **lower** and **upper bounds** \(\Phi_L, \Phi_U\) with lower readability.
- **Definition of lower and upper bounds** depends on the semiring.
  - In the Boolean semiring: \(\Phi_L \leq \Phi \leq \Phi_U\)
  - In the semiring over natural numbers: \(\Phi_L \leq \Phi \leq \Phi_U\)
  - For all semirings: Drop (add) monomials for lower (upper) bounds
- **Lower bound** \(\Phi_L\): \(\Phi_L = (a_1 + a_2)(s_1(a_1 + a_2) + s_2(a_1 + a_2)) + (a_3 + a_4)s_3s_4\)
- **Upper bound** \(\Phi_U\): \(\Phi_U = (a_1 + a_2)(s_1(a_1 + a_2) + s_2(a_1 + a_2)) + (a_3 + a_4)s_3s_4\)
- **Search for closest bounds** in a given class \(c\) of well factorisable polynomials.
  - \(c\) could be the class of polynomials with readability one.

**Query approximation:**
- Approximate a query \(Q\) by lower and upper bound queries \(Q_L\) and \(Q_U\).
  - For any database, the polynomials \(\Phi_L, \Phi_U\) of \(Q_L\) and \(Q_U\) are lower and upper bounds for the polynomial \(\Phi\) of \(Q\) and have lower readability.

---

## Results: Queries with Factorisations of Bounded Size

- We introduce **factorisation trees** which are statically derived from a query \(Q\).
  - are independent of the input database,
  - define a factorisation of the polynomial of \(Q(D), \) for any database \(D\).

### Characterisation of Conjunctive Queries

- For any query \(Q\), there is a rational number \(t(Q)\) such that for any database \(D\), \(Q(D)\) has a factorised polynomial of
  - with readability \(O(D^{(t(Q))/2})\),
  - with size at most \(D^{(t(Q))/2}\).
- Moreover, \(t(Q)\) is the smallest such number when restricted to factorisations defined by factorisation trees.
- A query satisfies \(t(Q) = 0\) if it is **hierarchical**. Then the polynomial of any \(Q(D)\) has a factorisation with **bounded readability**,
  - with size linear in the sizes of input database and query.
  - For hierarchical queries \(Q\) self-joins is also known that
  - in probabilistic databases, their exact probability can be computed in polynomial time,
  - in the finite cursor machine model, they can be evaluated in just one pass over the database.

### Results: Efficient Computation of Factorisations

- For any \(Q\) and \(D\), we compute a factorisation of readability \(O(D^{(t(Q))/2})\) and size at most \(D^{(t(Q))/2-1}\) in time \(O(D^{(t(Q))/2})\).
  - .. without computing the flat polynomial!

### Results: Approximation by Factorised Polynomials

- Approximation by polynomials of readability one, over the
  - Boolean semiring.
  - Equivalent syntactic and model-theoretic characterisations of lower and upper bounds.
  - Algorithms to enumerate bounds with polynomial delay.

### Selected Publications

- **On Factorisation of Provenance Polynomials**
- **Factorised Representations of Query Results**
  - Also arXiv report 1104.0867.
- **On the Optimal Approximation of Queries Using Tractable Propositional Languages**
  - R. Fink and D. Olteanu. In ICDT, 2011.