ChaseFUN: a Data Exchange Engine for Functional Dependencies at Scale

Angela Bonifati, Ioana Ileana, Michele Linardi

1. ChaseFUN: motivation and goals
ChaseFUN is the first Data Exchange (DE) engine targeting
• piecemeal process and parallelization of DE constraints during the chase by playing with constraint ordering and interaction; and
• interplay and interaction among constraints in order to reduce the size of intermediate results of the chase while providing granular insight into it.

As such, ChaseFUN addresses the coverage and efficient support of target functional dependencies lacking in available DE engines.

2. Classical DE Setting via an example
Process of taking data structured under a target schema, by exploiting the dependencies of the schemas.

<table>
<thead>
<tr>
<th>Active Actors</th>
<th>Awarded Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name surname age</td>
<td>name surname age</td>
</tr>
<tr>
<td>Leonardo Di Caprio</td>
<td>Matthew David</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>Matthew David</td>
</tr>
</tbody>
</table>

Chase steps with tgds add assignments to a target assignment set
- s-t tgds heads with the values in the assignments’ images.
- The target assignment set, i.e. producing atoms in the materializing insight into it.

3. The Interleaved Chase for efficient and scalable DE
Big Classical DE Chase Issue: fd-induced overhead, due to the often very large fd application scope (pre-solution...)

The Interleaved Chase, at the heart of ChaseFUN, mitigates fd-induced overhead by cleverly taming fd application scope.

The Interleaved Chase plays on s-t tgds assignments = mappings from s-t tgds variables to constants or labeled nulls. Assignments are constructed in an initial form for every DE scenario, and then chased:
- Chase steps with tgds add assignments to a target assignment set
- Steps with egds (fds) change assignments in the target set.
- The (intermediate) target solution can be obtained at any point by materializing the target assignment set, i.e. producing atoms in the s-t tgds heads with the values in the assignments’ images.

4. Saturation Sets and Overlaps
Fds apply on the target assignment set - how can one keep the target set low across the chase, so as to tame fd scope?
IDEA: split the chase into small, independent units!

Saturation Sets are subsets $S$ of the set of tgt assignments $A$ for a scenario such that a given assignment in $S$ is guaranteed never to interact via fd application with an assignment in $A - S$. Saturation Sets are independent chase units!

The Interleaved Chase (and ChaseFUN)’s mission: building small Saturation Sets, by grouping together assignments that interact now or are suspected to interact later:
- Overlap of two assignments = pairs of (equal or prone to be equal) variables and corresponding involved fd
- Two overlapping assignments are placed in the same Saturation Set.

Example: the assignments:
- $a_1 = (\{n,Leonardo, s\mid Di Caprio, a\} \{Y_1, N_1, Y_2, N_2\})$
- $a_2 = (\{n',Leonardo, s\mid Di Caprio, n',Matthew, n',David, e, E_1, N_1, N_2\})$

overlap on $\langle n, n', (s, s'), e \rangle$ and $e_1$.

5. The Conflict Graph and parallelization
The Conflict Graph is a structure accounting for constraints interaction and a helper for finding overlapping assignments. Conflict Graph nodes correspond to s-t tgds, whereas an edge means that the two tgds may have overlapping assignments, further characterized by conflict areas adorning nodes.

Areas ($v_j$) = $\langle v_j \rangle = \langle n, s, e_j \rangle$

Areas ($v_j$) = $\langle v_j \rangle = \langle n', s, e_j \rangle$

Areas ($v_j$) = $\langle v_j \rangle = \langle n, n', s, e_j \rangle$

Areas ($v_j$) = $\langle v_j \rangle = \langle n, n', s, e_j \rangle$

Nice news: a Saturation Set cannot span two distinct connected components of the graph. Besides speeding up Saturation Set construction, the Conflict Graph thus provides ChaseFUN with a parallelization opportunity, enhancing its scalability and overall speed!

6. Comparative performance assessment
We stress-tested ChaseFUN using several large, iBench generated scenarios:
- OF scenarios (iBench Object Fusion), OF+ scenarios (OF + iBench Vertical Partitioning), OF++ scenarios (augmented OF+). On those, we have moreover compared to one of the fastest state-of-the-art DE engines: the Lunatic engine.

References