ChaseFUN: a Data Exchange Engine for Functional Dependencies at Scale

Angela Bonifati[⋆,†] Ioana Ileana[◇,‡] Michele Linardi[‡]

angela.bonifati@univ-lyon1.fr ioana.ileana@parisdescartes.fr michele.linardi@parisdescartes.fr

[⋆]CNRS LIRIS [†]Univ Lyon 1 [◇]INSERM IDS [‡]Univ Paris Descartes

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 **source schema** and transforming it into data structured under a **target schema**, by exploiting the dependencies of the schemas.

Active_Actors

name	*surname*	*age*
Leonardo	Di Caprio	40
John	Redmayne	33

Awarded_Actor

name	*surname*	*oscarName*	*year*
John	Redmayne	Best Actor	2014
Wallace	Beery	Best Actor	1932
Fredric	March	Best Actor	1932
Marlon	Brando Jr.	Best Actor	1954
Marlon	Brando Jr.	Best Actor	1972

Actor_Collaboration

$name_1$	$surname_1$	$name_2$	$surname_2$
Leonardo	Di Caprio	Matthew	David
Fredric	March	Miriam	Hopkins

S-t tgds:

$\mathbf{m}_1 : Active\_Actor(n, s, a) \rightarrow Actor(n, s, Y_1, Y_2)$

$\mathbf{m}_2 : Awarded\_Actor(n', s', p', w') \rightarrow Actor(n', s', T, T_1) \wedge Oscar\_Prize(p', w', T)$

$\mathbf{m}_3 : Actor\_Coll(n', s', n'', s'') \rightarrow Actor(n', s', E_1, E_2) \wedge Actor(n'', s'', E_3, E_2)$

Target fds:

$\mathbf{e}_1 : Actor(n, s, p, w) \wedge Actor(n, s, p', w') \rightarrow (p = p') \wedge (w = w')$

$\mathbf{e}_2 : Oscar\_Prize(p, w, z) \wedge Oscar\_Prize(p, w, z') \rightarrow (z = z')$

Classical DE Chase phase 1: Apply s-t tgds

$\mathbf{m}_1 : Active\_Actor(n, s, a) \rightarrow Actor(n, s, Y_1, Y_2)$

Pre-solution - **Actor** table

name	*surname*	*idRewarding*	*idClub*
Leonardo	Di Caprio	N_1	N_2
John	Redmayne	N_3	N_4
...	...	...	...

Classical DE Chase phase 2: Apply target fds

Pre-solution - **Actor** table

name	*surname*	*idRewarding*	*idClub*
Leonardo	Di Caprio	N_1	N_2
Leonardo	Di Caprio	N_{15}	N_{16}
...	...	...	...

Target fd $\mathbf{e}_1 : Actor(n, s, p, w) \wedge Actor(n, s, p', w') \rightarrow (p = p') \wedge (w = w')$

Solution - **Actor** table

name	*surname*	*idRewarding*	*idClub*
Leonardo	Di Caprio	N_{15}	N_{16}
...	...	...	...

Actor

name	*surname*	*idReward*	*idClub*
John	Redmayne	N_5	N_6
Wallace	Beery	N_7	N_8
Marlon	Brando Jr.	N_{13}	N_{14}
Leonardo	Di Caprio	N_{15}	N_{16}
Matthew	David	N_{17}	N_{16}
Fredric	March	N_{18}	N_{19}
Miriam	Hopkins	N_{20}	N_{19}

Oscar_Prize

oscarName	*year*	*idActor*
Best Actor	2014	N_5
Best Actor	1932	N_7
Best Actor	1954	N_{13}
Best Actor	1972	N_{13}

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 tgd assignments A for a scenario, such that an assignment in S **is guaranteed never to interact via fd application** with an assignment in $A - S$. Saturation Sets are **independent chase units!**

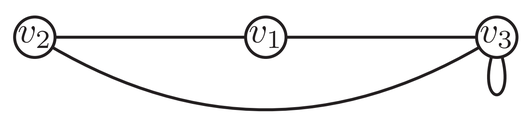
The Interleaved Chase (and ChaseFUN)'s mission: building small Saturation Sets, by **grouping together assignments that interact now or are suspected to interact later**:

- **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:
$a_1^{m1} = \{n{:}Leonardo, s{:}Di\ Caprio, a{:}42, Y_1{:}N_1, Y_2{:}N_2\}$
$a_1^{m3} = \{n'{:}Leonardo, s'{:}Di\ Caprio, n''{:}Matthew, s''{:}David, E_1{:}N_{15}, E_2{:}N_{16}, E_3{:}N_{17}\}$
overlap on $\langle n, n'\rangle, \langle s, s'\rangle$ **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.



$\mathbf{Areas}(\mathbf{v}_1) = \{ca_1^1 = \langle (n, s), e_1\rangle\}$.
$\mathbf{Areas}(\mathbf{v}_2) = \{ca_2^1 = \langle (n', s'), e_1\rangle\}$.
$\mathbf{Areas}(\mathbf{v}_3) = \{ca_3^1 = \langle (n'', s''), e_1\rangle, ca_3^2 = \langle (n''', s'''), e_1\rangle\}$.

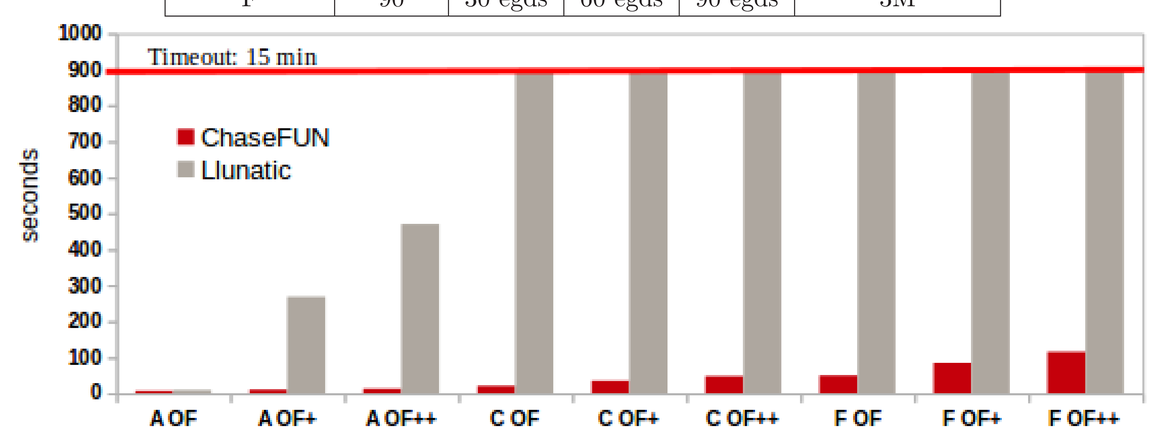
Nice news: **a Saturation Set cannot span two distinct connected components of the graph**. Besides **speeding up Saturation Set construction**, the Conflict Graph thus provides ChaseFUN with a *parallelization opportunity*, enhacing its scalability and overall speed!

6. Comparative performance assesment

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 Llunatic engine.

Scenarios

SCENARIO	*s-t tgds*	*OF*	*OF+*	*OF++*	*# source tuples*
A	15	5 egds	10 egds	15 egds	500K
C	45	15 egds	30 egds	45 egds	1.5M
F	90	30 egds	60 egds	90 egds	3M



References

[1] A. Bonifati, I. Ileana, M. Linardi. Functional Dependencies Unleashed for Scalable Data Exchange. In Proc. of SSDBM, 2016 https://arxiv.org/abs/1602.00563