Violating dataflow and getting away with it

Mistral Contrastin  Andrew Rice

Mistral.Contrastin@cl.cam.ac.uk
dodisturb.me
@madgen_
Datalog syntax in this talk

advisor(“Euler”, “Lagrange”).
advisor(“Lagrange”, “Fourier”).
advisor(“Lagrange”, “Poisson”).

academicAncestor(X, Y) :- advisor(X, Y).
academicAncestor(X, Y) :- academicAncestor(X, Z), advisor(Z, Y).

?- academicAncestor(X, “Fourier”).
Frustrating Datalog: programmer’s misery

\[ \text{lt100}(X) :- X < 100. \]

?- \text{in}(X,1,50), \text{lt100}(X). \]
Frustrating Datalog: programmer’s misery

pure(E) :- ! vars(E,_).
?- exp(E), pure(E).
Frustrating Datalog: programmer’s misery

?- exp(E₁), exp(E₂), independent(E₁,E₂).

independent(E₁,E₂) :- pure(E₁).
independent(E₁,E₂) :- pure(E₂).
independent(E₁,E₂) :-
    vars(E₁,V₁), vars(E₂,V₂), ! intersect(V₁,V₂,\_).
Frustrating Datalog: compiler writer’s misery

?- exp(E₁), exp(E₂),
   (pure(E₁);
   pure(E₂);
   vars(E₁,V₁), vars(E₂,V₂), ! intersect(V₁,V₂,\_)).
Overview

- Dataflow restrictions (well-moding & range restriction)
- Information flow (modes & adornment)
- Dataflow repair
- Synthesising control (reorder atoms)
Dataflow restrictions
Range restriction

- Every variable in the head must appear in the body

- Why?
  - Domain independence
  - Finiteness
  - Termination
  - Easy (not lazy) implementation

```cpp
int p(int i) { result = 42; }
```

$p(I, 42)$. 
Well-moding

Not every predicate is created equal

- \( \leq \) (as implemented in QL) is extralogical; \( \text{in} \) is logical

- Foreign predicates, parameters indexed in a database, semantically one way functions, etc.
Information flow
Mode = Binding requirement

- + means “needs to be bound”
- ? means “we don’t care”

- Static property independent of where the predicate occurs

- $\text{in}^{++}(I, \text{Lower}, \text{Upper})$
  
  \[
  A \leq^{++} B
  \]

- $\text{sha256}^{++?(\text{Content}, \text{Hash})}$

- $\text{plus}^{++?, +?, ?++}(A, B, C)$
In QL, **binding sets** seem to be the equivalent notion

```plaintext
plus++,++?,+++(A,B,C)
```

```plaintext
bindingset[a,b]
bindingset[b,c]
bindingset[a,c]
predicate plus(int a, int b, int c) {
    a + b = c
}
```
Adornment = active binding

- $b$ for bound
- $f$ for free
- Dynamic property depending on the context

?- $p_{fb}(X, 1)$, $q_{bfb}(X, Y, 0)$. 
Well-modable: Intuition

?- lt100(X).
lt100(X) :- X < 100.

?- in(X,1,50), lt100(X).
lt100(X) :- X < 100.
Well-modable: Formalisation

Need a b for every +

‣ ?- \text{lt100}_f(X).
  \text{lt100}_f(X) :- X <_{fb^{++}} 100.

‣ ?- \text{in}_{fbb}(X, 1, 50), \text{lt100}_b(X).
  \text{lt100}_b(X) :- X <_{bb^{++}} 100.
Well-modable vs well-moded

- Semantically clause heads and atoms are related

```prolog
?- in_{\text{fbb}}(X,1,50), \text{lt100}_b(X).
\text{lt100}_b(X) :- X <_{\text{bb}++} 100.
```

- But the evaluator treats each clause independently

```prolog
?- \text{in}_{\text{fbb}}(X,1,50), \text{lt100}_b(X).
\text{lt100}_f(X) :- X <_{\text{fb}++} 100.
```

- Well-modability is a global property, while well-modedness is a local one
Dataflow repair:
Transforming well-modable to well-moded and making clauses range-restricted
Repairing basic ill-modedness

?- in_{fbb}(X,1,50), lt100_0b(X).
lt100_0f(X) :- X <_{fb}^{++} 100.

?- in_{fbb}(X,1,50), lt100_0b(X).
lt100_0f(X) :-
in_{fbb}(X,1,50), X <_{bb}^{++} 100.
Repairing basic range-restriction

\[ \begin{align*}
\text{independent}(E_1, E_2) & \iff \text{pure}(E_1). \\
\text{independent}(E_1, E_2) & \iff \text{pure}(E_2).
\end{align*} \]

\[ \begin{align*}
\text{independent}(E_1, E_2) & \iff \text{exp}(E_2), \text{pure}(E_1). \\
\text{independent}(E_1, E_2) & \iff \text{exp}(E_1), \text{pure}(E_2).
\end{align*} \]
Not so basic dataflow 1

?- pClo(X,1).
pClo(X,Y) :- p?+(X,Y).

› Is this well-modeled?
Not so basic dataflow 2

\[ ?- p(X, 1), r(Y), p(X, Y). \]
\[ p(X, Y) :- q(X). \]

What flows into \( Y \) of \( p \)?
What are we trying to find?

‣ For well-modedness, target is body atom parameters. What flows into $X$ of $X <^{++} 100$?

?- in(X,1,50), lt100(X).
lt100(X) :- X <^{++} 100.

‣ For range restriction, target is predicate parameters. What flows into second parameter of independent?

?- exp(E₁), exp(E₂), independent(E₁,E₂).
independent(E₁,E₂) :- pure(E₁).
What are we really trying to find?

- Enumerable domains for variables!
- QL makes this much easier with its **characteristic predicates**
- In pure Datalog, we must look hard for them
How to find these domains? (sketch)

1. Extract all local flow into a graph

2. Compute the shortest paths from data sources (constants, positive atoms) to data sinks (offending atom/predicate parameters)

3. Ensure data paths to offending sinks are closed

4. Extract domain into new predicate and prepend it to the problematic clause body
Implementation

- Feature extraction can be done in Datalog via MetaDL
- Flow path computation is about ~15 pure Datalog clauses
- Transformation cannot (yet) be done in Datalog!
- Similar analysis required for Magic Set Rewriting, potential infrastructure reuse
Mode inference and atom reordering
Adornments are contextual

Reminder: need a \texttt{b} for every \texttt{+}

- \texttt{auth}_f(\texttt{U}) \leftarrow \texttt{hash}_{ff^+}(\texttt{P},\texttt{H}), \texttt{password}_{bf}(\texttt{U},\texttt{P}), \texttt{valid}_{bb}(\texttt{U},\texttt{H}). \xmark

- \texttt{auth}_f(\texttt{U}) \leftarrow \texttt{password}_{ff}(\texttt{U},\texttt{P}), \texttt{hash}_{bf^+}(\texttt{P},\texttt{H}), \texttt{valid}_{bb}(\texttt{U},\texttt{H}). \checkmark
Naïvely enumerate orderings?

- $\text{auth}_f(U) :\text{- hash}_{ff}^+(P,H), \text{password}_{bf}(U,P), \text{valid}_{bb}(U,H)$. ❌
- $\text{auth}_f(U) :\text{- hash}_{ff}^+(P,H), \text{valid}_{fb}(U,H), \text{password}_{bb}(U,P)$. ❌
- $\text{auth}_f(U) :\text{- password}_{ff}(U,P), \text{hash}_{bf}^+(P,H), \text{valid}_{bb}(U,H)$. ✅
- $\text{auth}_f(U) :\text{- password}_{ff}(U,P), \text{valid}_{bf}(U,H), \text{hash}_{bb}^+(P,H)$. ✅
- $\text{auth}_f(U) :\text{- valid}_{ff}(U,H), \text{password}_{bf}(U,P), \text{hash}_{bb}^+(P,H)$. ✅
- $\text{auth}_f(U) :\text{- valid}_{ff}(U,H), \text{hash}_{fb}^+(P,H), \text{password}_{bb}(U,P)$. ✅
It’s a global reordering problem

- ?- authf(U)
  authf(U) :- checkff(U,P), passwordbb(U,P).
  checkff(U,P) :- hashff+(P,H), validbb(U,H).
  ❌

- ?- authf(U)
  authf(U) :- passwordff(U,P), checkbb(U,P).
  checkbb(U,P) :- hashbf+(P,H), validbb(U,H).
  ✅

- Combinatorial explosion! Naïve won’t do.
What are we really trying to compute?

- Orderings, but also a summary of binding requirements
- Mode inference for derived predicates
- What should the ideal mode of check is?

\[ \text{check}_{bb}(U,P) :\!-\! \text{hash}_{bf}^{+?}(P,H), \text{valid}_{bb}(U,H). \]

- +? or ??? ?? is strictly better
- We have submoding! Only want the most general mode
- Each mode is coupled to orderings that lead to it
Multiple modes due to ordering

\[
\begin{align*}
\text{clientCheck}(P) & : -\ \text{weak}(P,H). \\
\text{serverCheck}(H) & : -\ \text{weak}(P,H). \\
\text{weak}(P,H) & : -\ \text{hash}^+?(P,H),\ \text{rainbow}^+?(H,P).
\end{align*}
\]

\[\text{The mode of weak is } +?,?+\]
Mode inference algorithm
(sketch)

1. Start with input modes, assume others are pure

2. For each clause, construct a greedy scheduling graph

3. Find the most relaxed modes and their orderings

4. Combine modes of clauses sharing head predicates

5. Go to (2) repeat until convergence
Two heuristics against naïveté

- Pure atoms can always be scheduled first in any order
- Scheduling might make other atoms pure

?- ▪ p⁺(Z), q⁺?(X,Z), r??(X,Y).
?- r??(X,Y), ▪ p⁺(Z), q??(X,Z).
?- r??(X,Y), q??(X,Z), ▪ p?(Z).
?- r??(X,Y), q??(X,Z), p?(Z).
Properties

- Sound and complete
- Incremental
- Highly parallelisable
- Closely mimics properties of Datalog evaluation
- Still exponential in degenerate case but only locally
  \[ ?- \ p^+(X), \ q^+(Y), \ r^+(Z). \]
Conclusion

‣ Well-modedness and range restriction are sources of "is not bound to a value" errors

‣ They can be fixed via program transformations

‣ Dataflow restricted atoms can be used as if purely logical

‣ Modes (or binding sets) can always be inferred

‣ Datalog analysis is nice because Datalog is nice!
Thanks. Questions?

Mistral.Contrastin@cl.cam.ac.uk
dodisturb.me
@madgen_