Automatic Reordering for Dataflow Safety of Datalog

Or how I stopped worrying about syntactic order of execution and love greedy scheduling

Mistral Contrastin Dominic Orchard Andrew Rice

UNIVERSITY OF CAMBRIDGE University of Kent

EPSRC
Engineering and Physical Sciences Research Council

The problem with syntactic order of execution

Version 1:

```
auth(User) :- hash(Pass,Hash), password(User,Pass), valid(User,Hash).
```

Version 2:

```
auth(User) :- password(User,Pass), hash(Pass,Hash), valid(User,Hash).
```

- Datalog recap
- Modes, adornments & well-modedness
- Intra- and inter-clausal analysis
- Properties of the analysis
- Future work

Datalog recap

- Good for deductive databases, AI, data integration, program analysis
- No function symbols, unlike Prolog, e.g., no lists
- Negation
- Aggregation
- Extralogical predicates, e.g., IO and foreign functions

Example Datalog program

- A set of extensional predicates and intensional predicates.

```
pc_predecessor(“Brigitte Pientka”, “Peter Thiemann”).
pc_predecessor(“Germán Vidal”, “Brigitte Pientka”).
pc_predecessor(“Elvira Albert”, “Germán Vidal”).
pc_predecessor(“Olivier Danvy”, “Elvira Albert”).
```

```
ancestor(X,Y) :- pc_predecessor(X,Y).
ancestor(X,Z) :- pc_predecessor(X,Y), ancestor(Y,Z).
```

Subgoal

Head

Body

Clause

- Queries to retrieve information

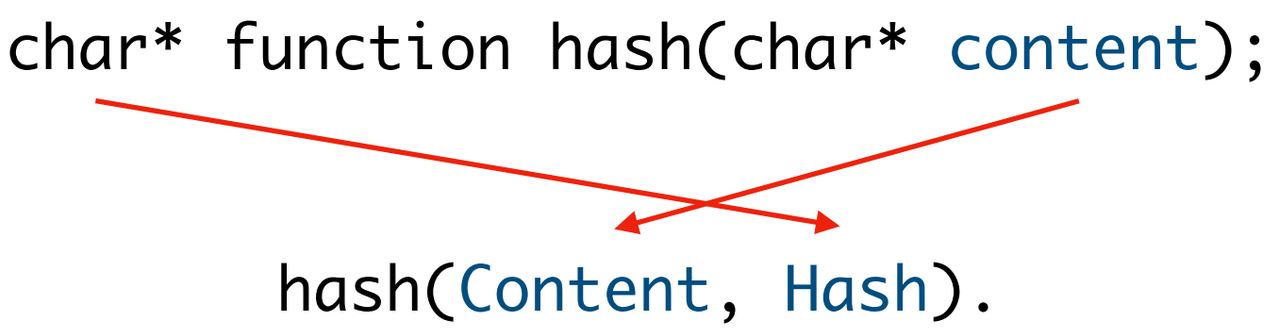
```
?- ancestor(X,“Peter Thiemann”).
```

- Datalog recap
- Modes, adornments & well-modedness
- Intra- and inter-clausal analysis
- Properties of the analysis
- Future work

Engineering and Physical Sciences Research Council

Practical applications require something *extra*

- Promoting a C function into a logical predicate:



Modes to capture *static* dataflow

- Use `+` to require the variable to be bound invocation time
- Use `?` to say you do not care if it is bound or not.
- Earlier predicate with mode annotation:
  `hash+?(Content,Hash)`
- Multiple implementations lead to multiple mode patterns.

Adornments to capture *dynamic* dataflow

- The binding of variables depends on the query
  - Bound variables are marked with b and free ones with f
  - Traditionally computed left-to-right in clause body

```
?-auth(“Rebecca”).
auth_b(User) :- hash+?_ff(Pass,Hash), password_bb(User,Pass), valid_bb(User,Hash).
```

Reordering changes binding pattern:

```
auth_b(User) :- password_bf(User,Pass), hash+?_bf(Pass,Hash), valid_bb(User,Hash).
```

Well-modedness

- Informally, a well-moded program's subgoals do not give invocation errors due to insufficient argument binding.

- Formally, an agreement between the mode patterns and the adornment of subgoals.

- Consider the two adornments of `hash`:

`hash`$^{+?}_{ff}$`(Pass,Hash)` ❌ `hash`$^{+?}_{bf}$`(Pass,Hash)` ✅

Global reordering is needed

- Recall the different orderings of authentication clauses

```
auth_b(User) :- hash+?_ff(Pass,Hash), password_bb(User,Pass), valid_bb(User,Hash).
auth_b(User) :- password_bf(User,Pass), hash+?_bf(Pass,Hash), valid_bb(User,Hash).
```

- What if it was written this way?

```
auth_b(U) :- check_bf(U,P), password_bb(U,P).
check_bf(U,P) :- hash+?_ff(P,H), valid_bb(U,H).
```

- Reordering the caller help well-moding the subgoals of callee!

```
auth_b(U) :- password_bf(U,P), check_bb(U,P).
check_bb(U,P) :- hash+?_bf(P,H), valid_bb(U,H).
```

- Datalog recap
- Modes, adornments & well-modedness
- Intra- and inter-clausal analysis
- Properties of the analysis
- Future work

Mode analysis in two parts

- *Intra-clausal analysis* determines an ordering constraint for each clause based on its subgoals and known constraints alone

- *Inter-clausal analysis* updates constraints until they stabilise (a fixpoint is reached)

Intra-clausal analysis

- Be greedy and schedule *easy* subgoals ASAP
- Exploit shared variables between subgoals
- Produce orderings using a graph construction that encodes orderings as paths

Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```

A =({X, Y, Z, W}
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) }
, {})

Cost to pay

Alternatives & their costs

Cost paid

A

EPSRC Engineering and Physical Sciences Research Council

Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```

A =({X, Y, Z, W} ← Cost to pay
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) } ← Alternatives & their costs
, {}) ← Cost paid

B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

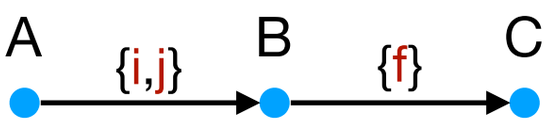
Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```

A =({X, Y, Z, W} ← Cost to pay
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) } ← Alternatives & their costs
, {}) ← Cost paid

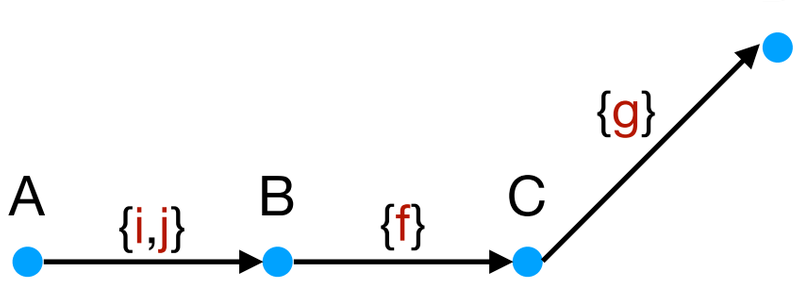
B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

C =({Y, Z}, {(g, {Y}), (g, {Z}), (h, {Z})}, {})



Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```



A =({X, Y, Z, W} ← Cost to pay
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) } ← Alternatives & their costs
, {}) ← Cost paid

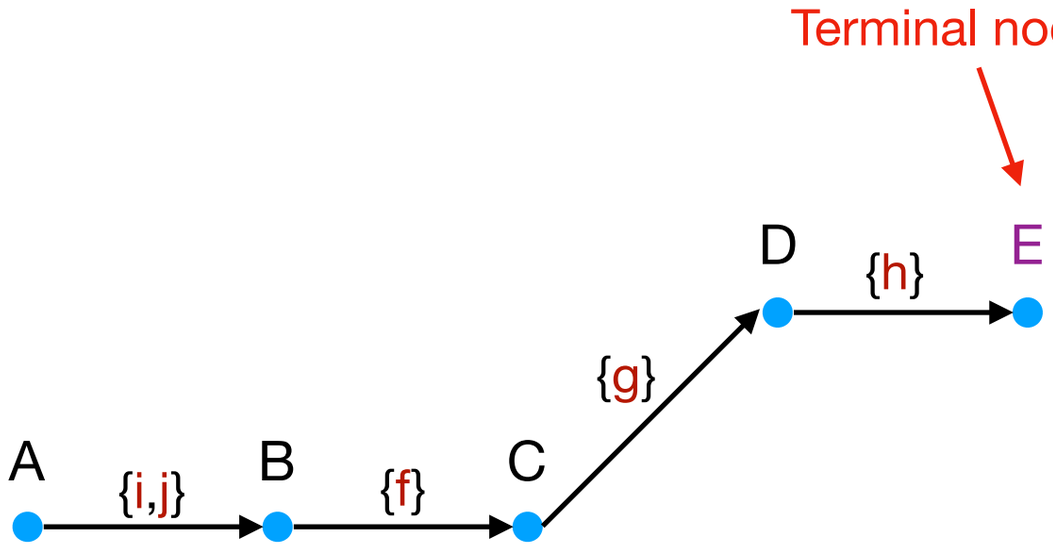
B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

C =({Y, Z}, {(g, {Y}), (g, {Z}), (h, {Z})}, {})

D =({}, {(h, {})}, {Y})

Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```



A =({X, Y, Z, W}
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) }
, {})

Cost to pay

Alternatives & their costs

Cost paid

B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

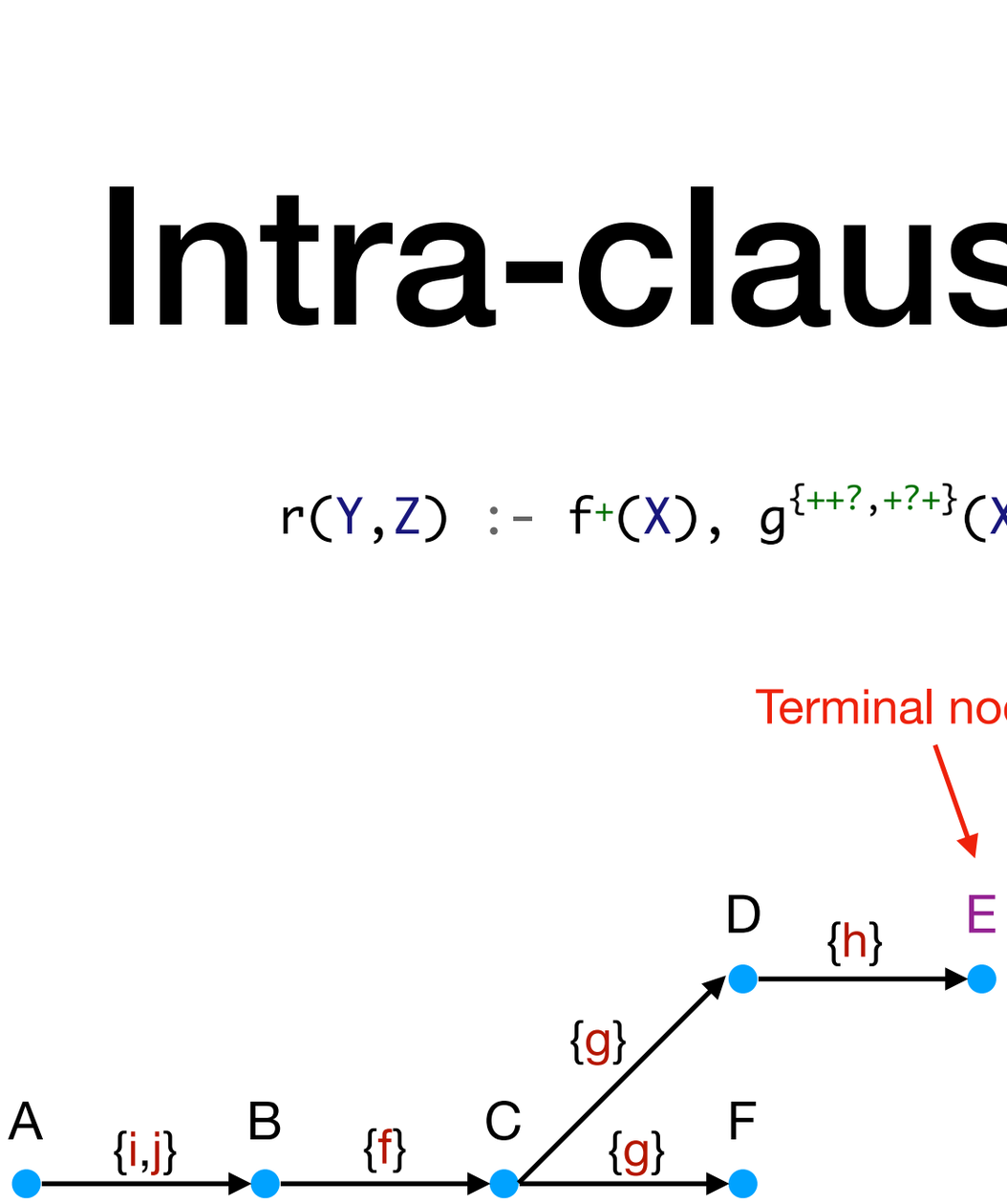
C =({Y, Z}, {(g, {Y}), (g, {Z}), (h, {Z})}, {})

D =({}, {(h, {})}, {Y})

E =({}, {}, {Y})

Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```



A =({X, Y, Z, W}
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) }
, {})

Cost to pay

Alternatives & their costs

Cost paid

B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

C =({Y, Z}, {(g, {Y}), (g, {Z}), (h, {Z})}, {})

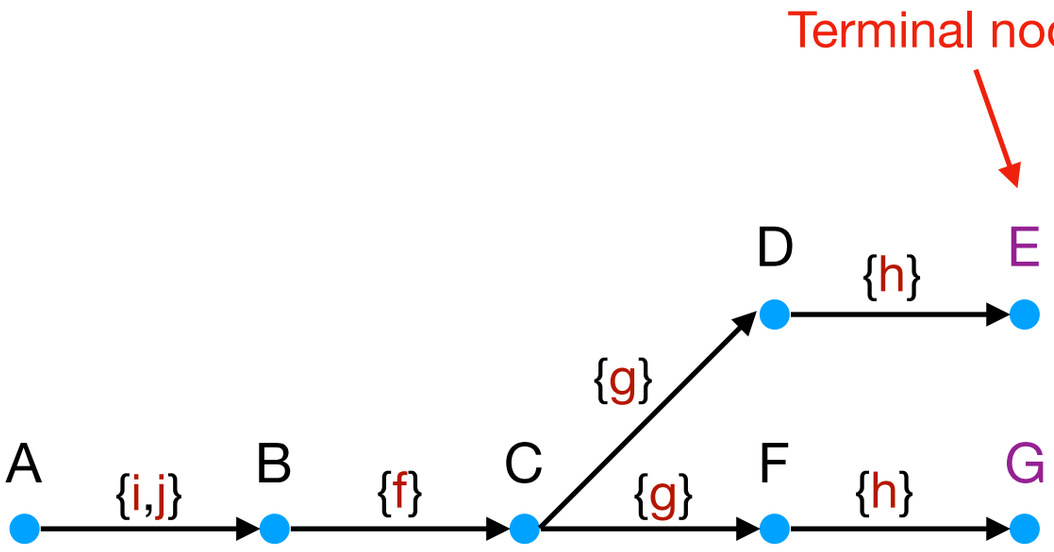
D =({}, {(h, {})}, {Y})

E =({}, {}, {Y})

F =({}, {(h, {})}, {Z})

Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```



Terminal node

A =({X, Y, Z, W}
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) }
, {})

Cost to pay

Alternatives & their costs

Cost paid

B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

C =({Y, Z}, {(g, {Y}), (g, {Z}), (h, {Z})}, {})

D =({}, {(h, {})}, {Y})

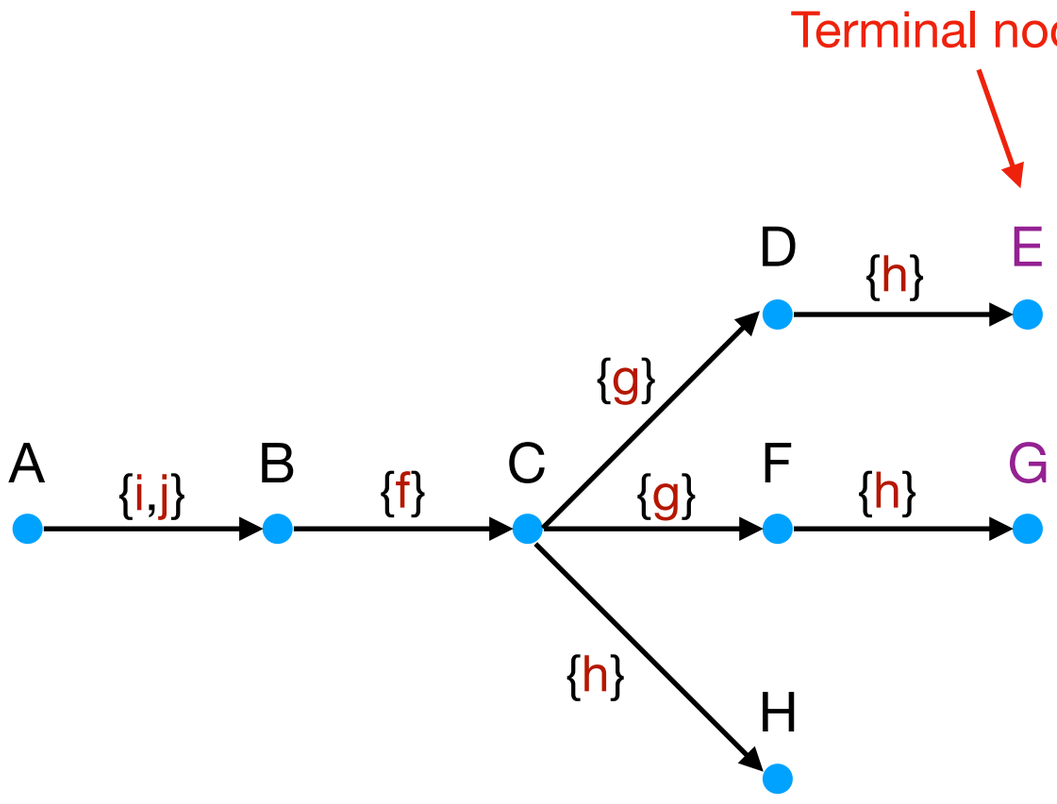
E =({}, {}, {Y})

F =({}, {(h, {})}, {Z})

G =({}, {}, {Z})

Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```



Terminal node

A =({X, Y, Z, W}
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) }
, {})

Cost to pay

Alternatives & their costs

Cost paid

B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

C =({Y, Z}, {(g, {Y}), (g, {Z}), (h, {Z})}, {})

D =({}, {(h, {})}, {Y})

E =({}, {}, {Y})

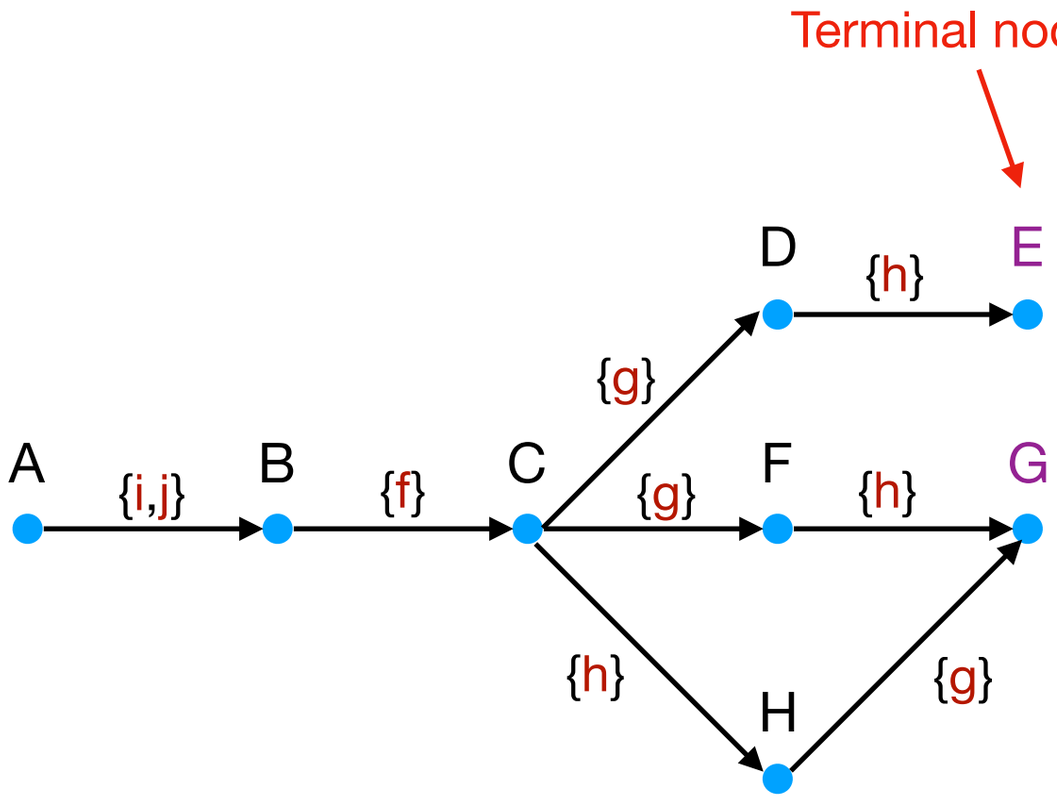
F =({}, {(h, {})}, {Z})

G =({}, {}, {Z})

H =({Y},{(g,{})},{Z})

Intra-clausal example

```
r(Y,Z) :- f+(X), g{++?,+?+}(X,Y,Z), h+(Z), i(X), j(X,W).
```



Terminal node

A =({X, Y, Z, W}
, { (f, {X}), (g, {X, Y}), (g, {X, Z})
, (h, {Z}), (i, {}), (j, {}) }
, {})

Cost to pay

Alternatives & their costs

Cost paid

B =({Y, Z}, {(f, {}), (g, {Y}), (g, {Z}), (h, {Z})}, {})

C =({Y, Z}, {(g, {Y}), (g, {Z}), (h, {Z})}, {})

D =({}, {(h, {})}, {Y})

E =({}, {}, {Y})

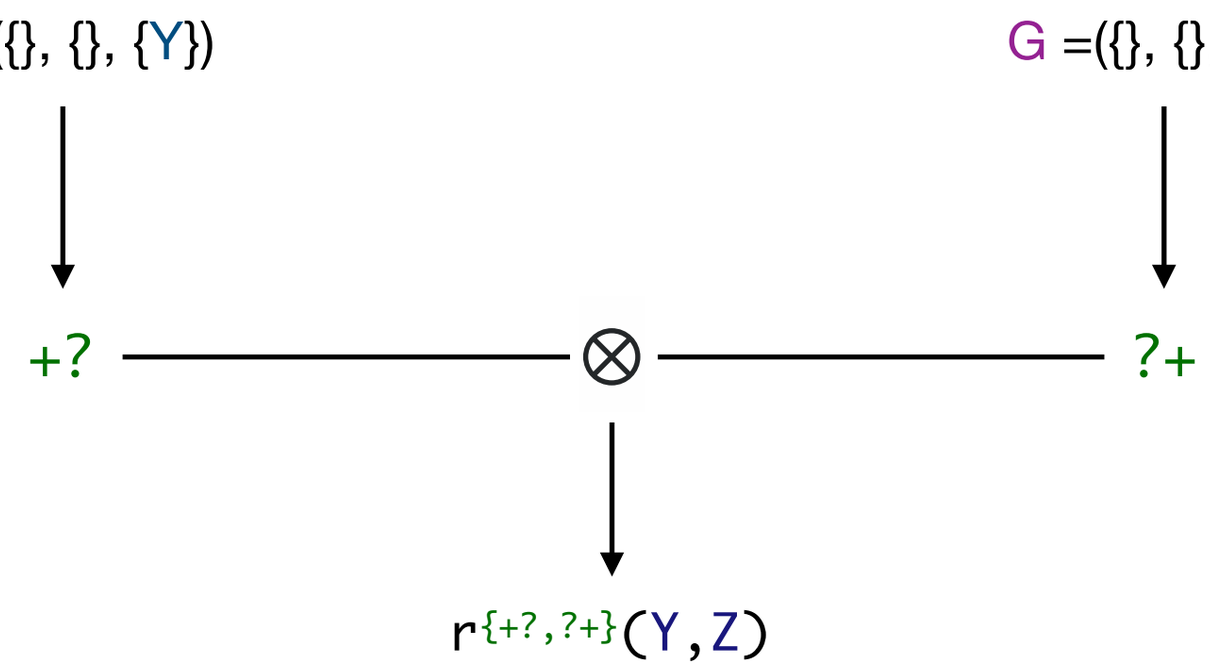
F =({}, {(h, {})}, {Z})

G =({}, {}, {Z})

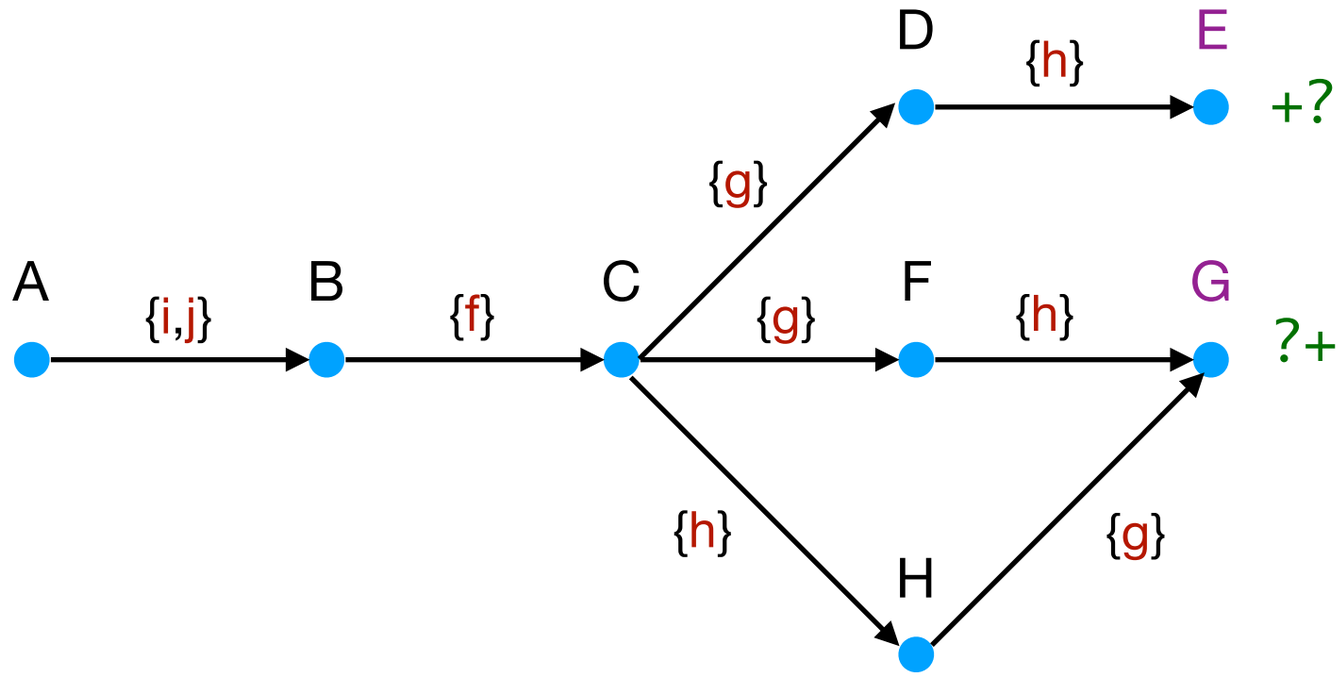
H =({Y},{(g,{})},{Z})

Intra-clausal example

Head predicate: r(Y,Z)



Intra-clausal example: Path extraction

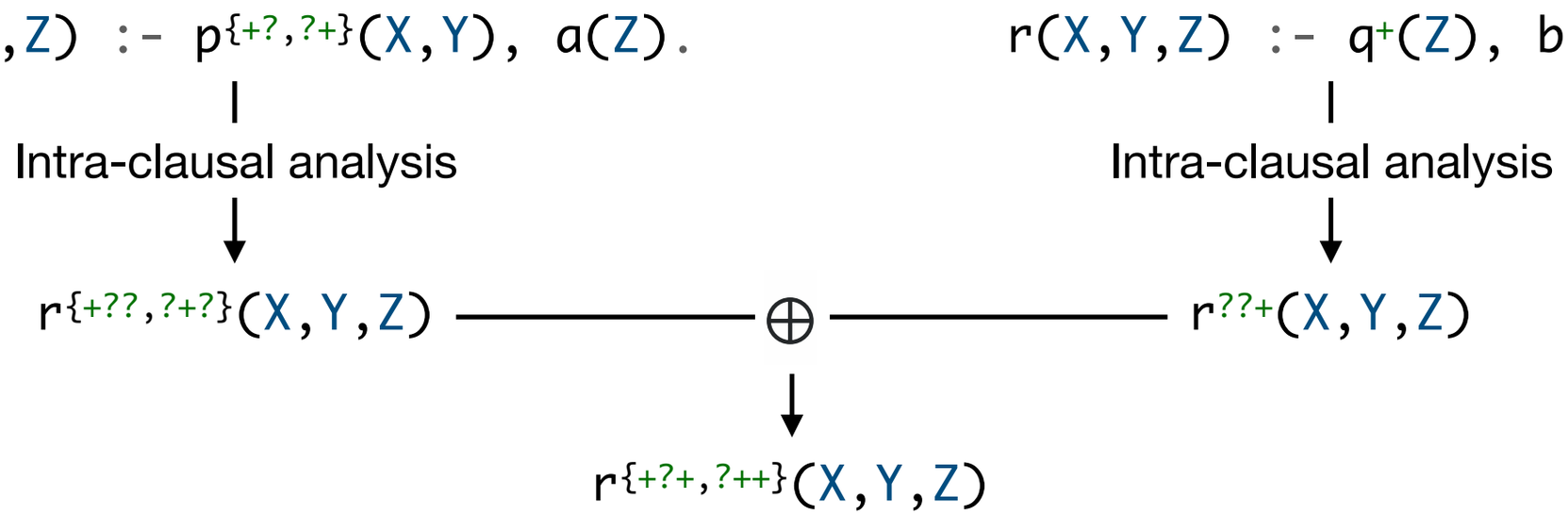


Orderings of subgoals leading to ?+

```
r(Y,Z) :- i(X), j(X,W), f+(X), g{++?,+?+}(X,Y,Z), h+(Z).
r(Y,Z) :- j(X,W), i(X), f+(X), g{++?,+?+}(X,Y,Z), h+(Z).
r(Y,Z) :- j(X,W), i(X), f+(X), h+(Z), g{++?,+?+}(X,Y,Z).
r(Y,Z) :- i(X), j(X,W), f+(X), h+(Z), g{++?,+?+}(X,Y,Z).
```

Inter-clausal analysis

- Constraint of a predicate respects the constraint of each of its clauses



- Update the constraints for each predicate
- Rinse and repeat until a fixpoint is reached

- Datalog recap
- Modes, adornments & well-modedness
- Intra- and inter-clausal analysis
- Properties of the analysis
- Future work

Sound and complete

- **Soundness** says if the algorithm finds an ordering for all clauses, there will not be invocation errors.

- **Completeness** says if there is an ordering of subgoals that eliminates invocation errors, the analysis will find it.

Incremental analysis

- Datalog is interactive, do not want to recompute.
- Addition of rules **never** invalidates previous analysis.
- When new rules do not extend existing predicates, it suffices to analyse **just** the new rules. Good for libraries.
- A query requires a **single** intra-clausal analysis round.

- Datalog recap
- Modes, adornments & well-modedness
- Intra- and inter-clausal analysis
- Properties of the analysis
- Future work

Engineering and Physical Sciences Research Council

Future work

- User given mode annotations for intentional predicates
- Analysis graphs contain other useful dataflow information
- ⊕ and ⊗ form Martelli's semiring suggesting analysis might be reduce to matrix operations
- Inlining and similar optimisations provide further well-moding opportunities

Recap

- Imperative programming tries to sneak in to declarative programming, we can do better!

- Well-modedness for Datalog is fully captured by adornments and simple modes

- It is possible to do better than brute-force search whilst remaining sound and complete

Thanks. Questions?

EPSRC
Engineering and Physical Sciences Research Council