J-Logic: Logical Foundations for JSON Querying

Jan Hidders
Free U. Brussels

Jan Paredaens
U. Antwerp

Jan Van den Bussche
U. Hasselt

VRIJE
UNIVERSITEIT
BRUSSEL

UHASSELT
KNOWLEDGE IN ACTION

uniquelyphenomenal.be

Logical Foundations for JSON Querying

- **Object descriptions:** sets of path–value pairs
- **Path variables**
- **Generating new keys** by packing
- Testing the **object-to-object property**
- **Containment** testing
- Open problems

- JSON-to-JSON queries,
  easy access to deeply nested objects

Describing a JSON object…

```
{ name:john,
  children: { 1:{name:anne,age:12},
              2:{name:bob,age:18},
              3:{name:chris,age:24} } }
```

…as a set of path–value pairs:

```
{ name:john,
  children.1.name:anne,
  children.1.age:12,
  children.2.name:bob,
  children.2.age:18,
  children.3.name:chris,
  children.3.age:24 }
```

“object description”

J-Logic ingredients

- Object descriptions as binary relations
- Input as well as output
- Path variables `$x` , concatenation `.`
- Atomic variables `@x`
- Atomic value constants `john` , `42`
- Logic-based language (Datalog)

[Sequence Datalog, Bonner&Mecca]

Hello World in J-Logic

```
T(hello:world).
> T={hello:world}
```

My first real J-Logic program

```
> Sales={ 1:{1999:40,2017:35},
          2:{1999:70,2001:80},
          3:{2001:90,2002:70,2017:33}}
ByYear(@y.@p:@s) :- Sales(@p.@y:@s).
> ByYear = { 1999:{1:40,2:70},
             2001:{2:80,3:90},
             2002:{3:70},
             2017:{1:35,3:33}}
```

```
> Sales={ 1:{1999:40,2017:35},
          2:{1999:70,2001:80},
          3:{2001:90,2002:70,2017:33}}
> ByYear = { 1999:{1:40,2:70},
             2001:{2:80,3:90},
             2002:{3:70},
             2017:{1:35,3:33}}
```

object descriptions!

```
ByYear(@y.@p:@s) :- Sales(@p.@y:@s).
```

```
ByYear={1999.1:40,      Sales={1.1999:40,
        2017.1:35,             1.2017:35,
        1999.2:70,             2.1999:70,
        2001.2:80,             2.2001:80,
        2001.3:90,             3.2001:90,
        2002.3:70,             3.2002:70,
        2017.3:33}             3.2017:33}
```

Deep equality in J-Logic

```
> R={ a: o1 ,
      b: o2 }
// Are o1 and o2 equal?
T(answer:no) :- R(a.$x:@v), not R(b.$x:@v).
T(answer:no) :- R(b.$x:@v), not R(a.$x:@v).
Q(answer:no) :- T(answer:no).
Q(answer:yes) :- not T(answer:no).
```

Key generation by packing

```
// input: R, a deeply nested object
// retrieve all subobjects with loc:chicago
Q(<$x>.$z:@v) :- R($x.loc:chicago),
                 R($x.$z:@v).
```

```
// input: R, a deeply nested object
// retrieve all subobjects with loc:chicago
Q(<$x>.$z:@v) :- R($x.loc:chicago),
                  R($x.$z:@v).
> S={a:1,b:2}
> T={a:3,b:4}
// cartesian product of JSON objects
Cart(<@x.@y>.s.@x:@u) :- S(@x:@u),T(@y:@v).
Cart(<@x.@y>.t.@y:@v) :- S(@x:@u),T(@y:@v).
> Cart={<a.a>:{s:{a:1},t:{a:3}},
        <a.b>:{s:{a:1},t:{b:4}},
        <b.a>:{s:{b:2},t:{a:3}},
        <b.b>:{s:{b:2},t:{b:4}}}
```

Cartesian product by packing

Proper objects

- JSON objects are unordered
- Are `{loc:raleigh,loc:chicago}` and `{loc:chicago,loc:raleigh}` the same?
- Not in JavaScript!
- Objects where keys are not keys (!) are called **improper**
- Input objects are normally proper
- We should avoid outputting improper objects
- **Object-object property:** **proper inputs** are mapped to **proper outputs**

The object-object (O2O) property

```
> R={raleigh:loc,chicago:loc}
Q(@v:@k) :- R(@k:@v). // not O2O
> Q={loc:raleigh,loc:chicago}
```

- All other example programs we have seen so far are O2O
- **Theorem:** It is decidable (in EXPTIME) whether a given **positive, recursion-free** J-Logic program is O2O
- "Unions of conjunctive queries" for J-Logic

Deciding the O2O property

- "JSON atomic equality-generating dependencies" (**jaegd**)
- D is proper if it satisfies these two jaegds:

```
D($x:@u),D($x:@v) ⟶ @u=@v
D($x:@u),D($x.$y:@v) ⟶ false
```

- Plan:

1. Reduce O2O decision to the **implication problem for jaegds**
2. Solve the jaegd implication problem

Chasing jaegds?

- Complications due to two kinds of variables
- Consider Σ:

```
R(@x:42) ⟶ false
R(<$x>:42) ⟶ false
R($x.$y:42) ⟶ false
```

- Then Σ logically implies σ:

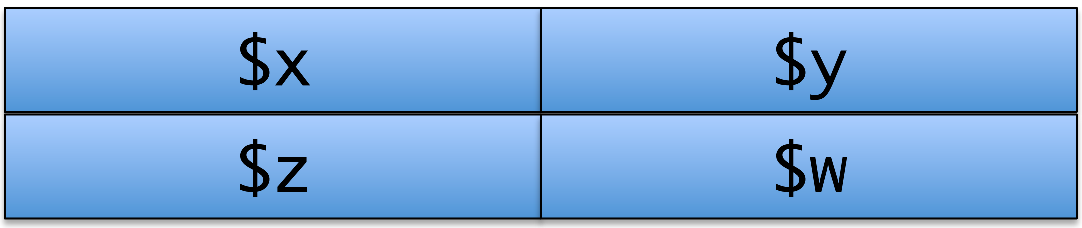
```
R($x:42) ⟶ false
```

- However, chasing σ with Σ does not fail
- Fortunately, dependencies expressing properness do not have this problem

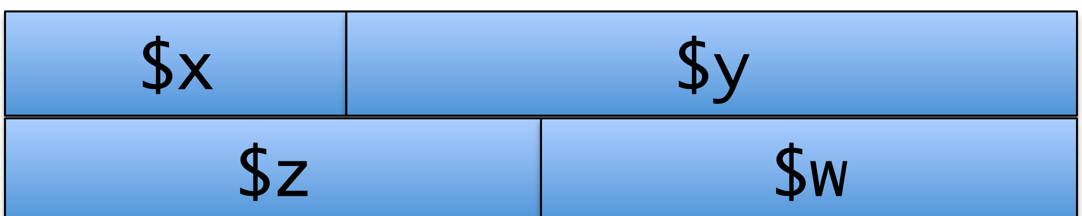
Unification in J-Logic

E.g. `$x.$y = $z.$w`

Split rule 3 ways:



`$x/$z, $y/$w`



`$x.$s/$z, $s.$w/$y`

`$z.$s/$x, $s.$y/$w`

The containment problem for J-Logic

- **Theorem:** Containment of a conjunctive J-Logic query in a union of conjunctive J-Logic queries, **over flat instances,** is decidable (Π-P-2)
- Flat instances have no packed keys
- We solve the **inclusion problem** for **pattern languages** (over an infinite alphabet) extended with atomic variables.

Conclusions

- Path variables & packing, useful for declarative JSON querying
- Further research:
  - Query processing
  - Theory of egds–tgds for J-Logic
  - Precise complexity of O2O problem, containment
  - Containment over non-flat instances?
  - Is packing necessary for recursion-free **flat-flat** queries?

uniquelyphenomenal.be

Pattern inclusion

- `$x.$y` included in `@x.$y`
- No homomorphism from right to left!
- Replace left pattern by four variants:
  - $@x_1.@y_1$
  - $@x_1.@x_2.@y_1$
  - $@x_1.@y_1.@y_2$
  - $@x_1.@x_2.@y_1.@y_2$