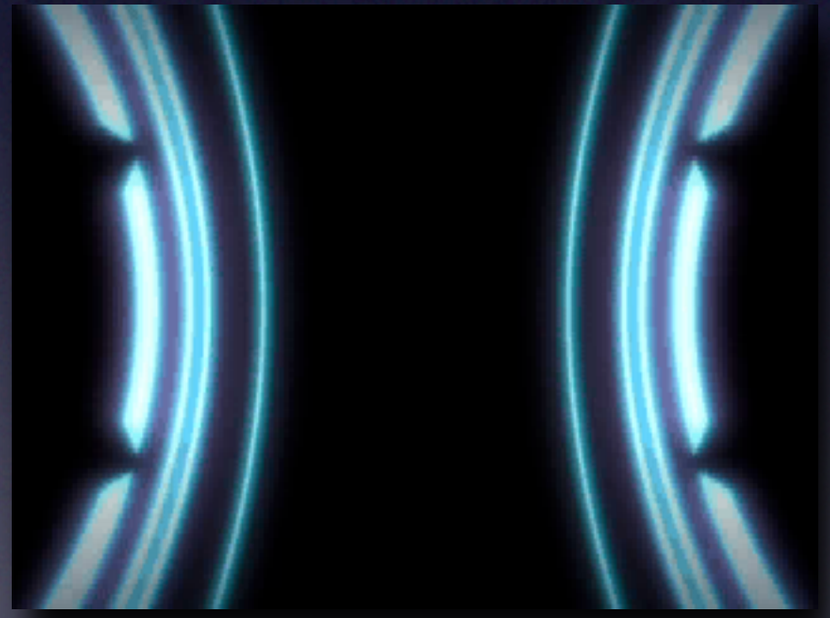
Semi-Strong Static Type Checking of Object-Oriented Query Languages

Michał Lentner[#], Krzysztof Stencel[*], Kazimierz Subieta[#]



[#] Polish-Japanese Institute of Information Technology, Poland (m.lentner@pjwstk.edu.pl, subieta@pjwstk.edu.pl)
[*] University of Warsaw, Poland (stencel@mimuw.edu.pl)

Motivations

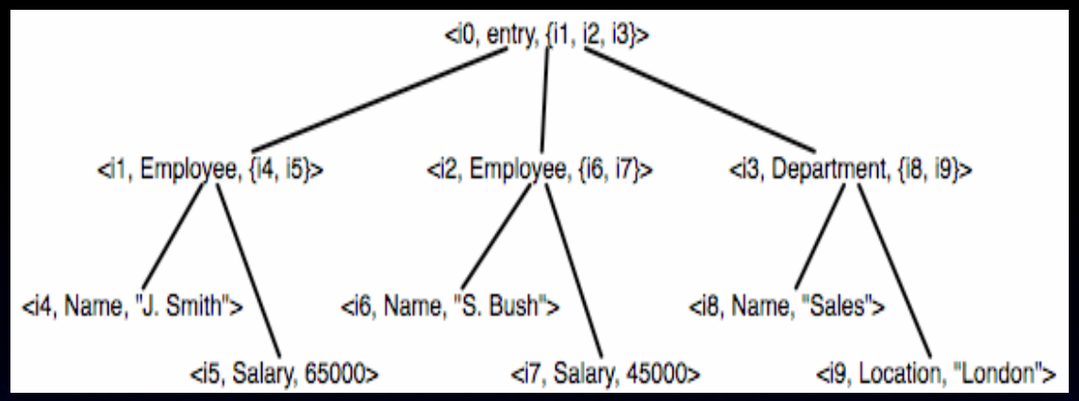
- There are features of query/programming languages and environments that make implementation of types extremely difficult. Among them are such notions as: mutability, collection cardinality constraints, collection kinds, single-, multi- and dynamic inheritance, modules, export/import lists, auxilliary names in queries, etc.
  Many type systems do not support these complex structures.

- Queries and: irregularities in data structures (null values, repeating data, variants/unions, unconstrained data names), ellipses and coercions, etc.
  Strong type checking should be relaxed to be efficient for programmers.

- Major issues in typing systems are not theoretical, but practical.
  They are concerned with development of solutions that meet the typical programmers' psychology.
  Theories are good guidelines, but eventually must be corrected by practical considerations.

- Everything in the area of type systems has already been done?
  If it is so good in theory, why is it so bad in practice?

Runtime Query Evaluation



Employee where Name = "J. Smith" and Salary > 10000

1. Initialize ENVS and QRES.

Employee(i1), Employee(i2), Department(i3)	

2. Execute **bind**(Employee)

Employee(i1), Employee(i2), Department(i3)	bag(i1, i2)

2. Pop one element from QRES.

Employee(i1), Employee(i2), Department(i3)	

3. Create a new ENVS section. Execute **nested**(i1).

Name(i4), Salary(i5)	
Employee(i1), Employee(i2), Department(i3)	

4. Execute **bind**(Name)

Name(i4), Salary(i5)	
Employee(i1), Employee(i2), Department(i3)	i4

5. push("J. Smith")

Name(i4), Salary(i5)	"J. Smith"
Employee(i1), Employee(i2), Department(i3)	i4

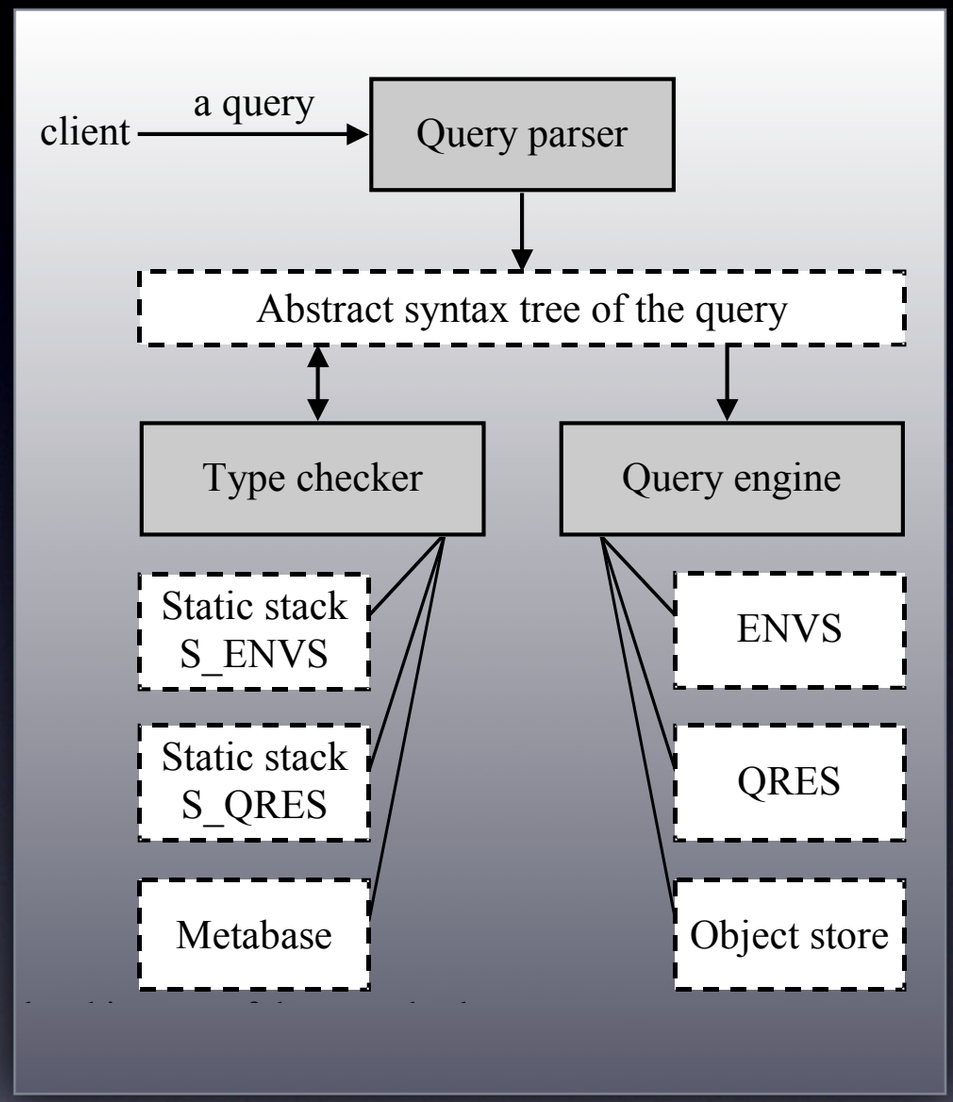
6. Pop two elements, dereference i4, compare "J. Smith" and "J. Smith", push true.

Name(i4), Salary(i5)	
Employee(i1), Employee(i2), Department(i3)	true

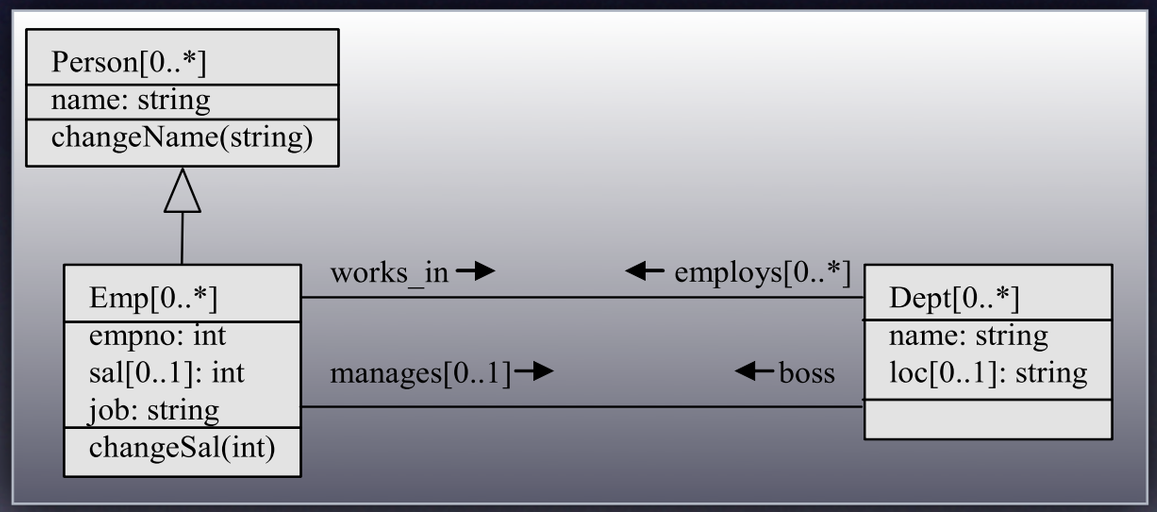
and so on...

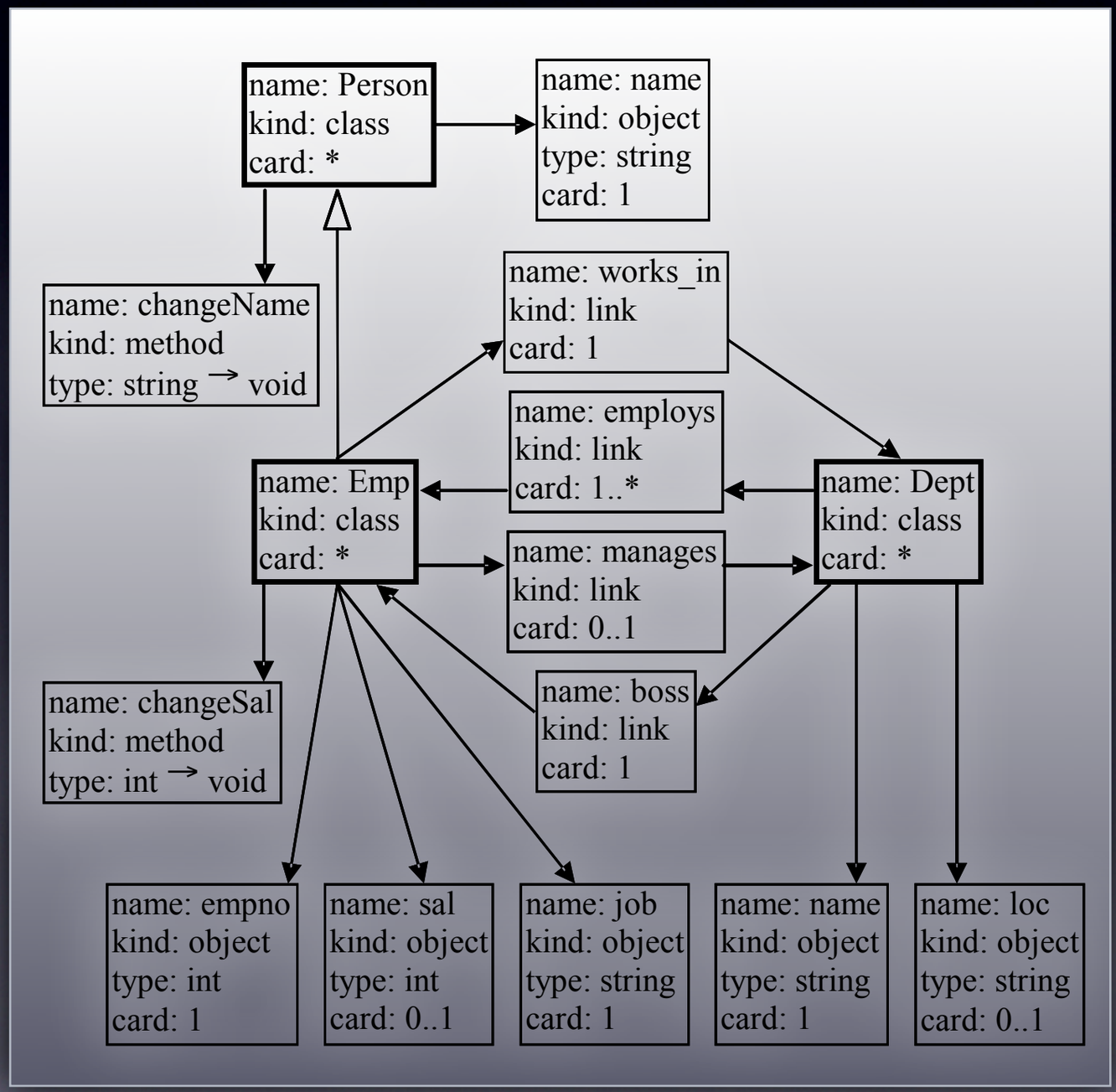
Our approach to type checking

- Runtime structures of SBA:
  - Data Store
    Represents objects stored in the database.
  - Environment Stack
    Responsible for name binding, enforcing scope rules, supporting non-algebraic operators, procedure calls, etc.
  - Query Result Stack
    Responsible for storing temporary and final query results.
- Compile-time structures:
  - Metabase
    Represents the database schema.
  - Static Environment Stack
    Responsible for name binding, enforcing scope rules, supporting non-algebraic operators, procedure calls, etc.
  - Static Query Result Stack
    Responsible for storing temporary and final signatures.
  - Type inference tables
    Information about operators, signatures of arguments, and signatures of results.



Database Schema and Metabase





Signatures

S - set of signatures, SC - set of signature components

- Names of atomic types (e.g. integer, real, etc.) ∈ SC
- Static identifiers of the metabase graph nodes ∈ SC
- Static binders: pairs *name*(s) ∈ SC, where s ∈ S
- Static structures: { s_1, s_2, ... } ∈ SC, where s_1, s_2, ... ∈ S
- $\forall_{(s \in SC)}$ s[c]o ∈ S , where:

  c - cardinality ([0..1], [1..1], [0..*], [1..*], etc.),
  o - ordering (bag or sequence)

Type Inference Decision Tables

Addition

q1	q2	q1 + q2
int[1..1]bag	int[1..1]bag	int[1..1]bag
int[1..1]bag	real[1..1]bag	real[1..1]bag
int[x..y]bag, x != 1 or y != 1	int[x..y]bag, x != 1 or y != 1	type error
...	...	...

Union

q1	q2	q1 union q2
T[a..b]bag	*T*[x..y]bag	*T*[a + x ... b + y]bag
...	...	...

Projection

q1	q2	q1.q2
T_1[a..b]bag	T_2[x..y]bag	T_2[a * x ... b * y]bag
...	...	...

Selection

q1	q2	q1 where q2
T[x..y]bag	bool[1..1]bag	*T*[0..y]bag
...	...	...

etc...

Static Type Checking

The type checker simulates runtime query evaluation using the metabase, the static environment stack, the query result stack and type inference decision tables. It performs the computation on signatures just as if they were actual data. It also:

- Checks names used in the query
- Generates messages on type errors
- Restores the process of type checking after a type error has been encountered
- Augments the syntactic tree with new nodes in order to resolve ellipses
- Augments the syntactic tree with dynamic type checks, if the type correctness cannot be asserted statically

Thank you!

More information about SBA:
http://www.ipipan.waw.pl/~subieta/