Froglingo, a Database Programming Language

Kevin Xu
Bigravity Business Software
khxu@bigravity.com

At Microsoft Research Cambridge
May 19th, 2006

What is Froglingo

- FrogLingo is a database programming language. It is based on the lambda calculus. One can uniformly express both data and application logic.
- The FrogLingo system is a computer system that implements the FrogLingo language. It has a single database storing both data and application logic.
- The website www.froglingo.com for more.

Why Froglingo?

Among all the database programming languages, Froglingo is looking for:

- Higher Productivity.
- Best Performance.

Agenda

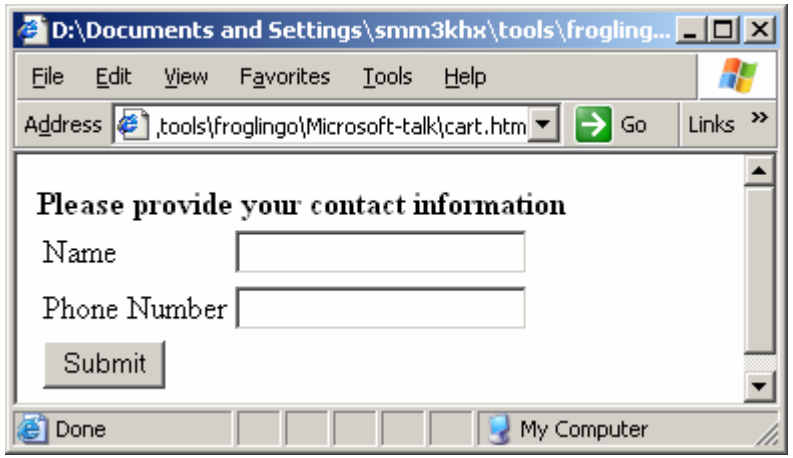
- Sample Expressions
- Concepts
- Productivity
- Time Complexity
- Related Work
- Future Work

Froglingo – Sample Expressions

```
> 5;
-- 5
> "HelloWorld";
-- "HelloWorld"
> HelloWorld;
-- null
> create John salary = 1500;
-- successful
> John salary + 5;
-- 1505
> create tax $money=($money * 0.3);
-- successful
> tax (John salary);
-- 450
> create fac 0 = 1;
-- successful
> create fac n:[n>0]=(n*(fac (n-
   1)));
-- successful
> fac 3;
-- 6
```

```
> create Smith salary = 3000;
> Select $person, $person salary,
    tax ($person salary) where
    $person salary >= 1500;
-- John, 1500, 450
-- Smith, 3000, 900
> create a1=100;
> create a2=300;
> create transfer $money =
 (update a2 = (a2 - $money)),
 (update a1 = (a1 + $money));
> transfer 35;
--successful
> a1;
-- 135;
> a2;
-- 265
```

Froglingo – Sample Expressions




```
<html><body>
<b>Please provide your contact information </b><br>
<table>
 <form action="../servlet/epserv">
 <tr>
    <td> Name </td>
    <td> <input type=text name = “name">  </td>
 </tr>
 <tr>
    <td> Phone Number </td>
    <td> <input type=text name = "visitor @name phone"></td>
 </tr>
 <tr>
   <td colspan=2> <input type=submit value = "Submit"></td>
  </tr>
     <input type=hidden name=epFun value=“add_visitor">
     <input type=hidden name=epPara value="name">
  </form>
</table>
</body></html>
```

```
add_visitor $name =
    “<html><body>Welcome “, $name,
    did_u_visit (visitor $name) $name
    ;
did_u_visit null $name =
    servlet {
          <“funname”, epCreate>,
          <“name”, $name>
    },
    “. You are a new visitor
    </body></html>”
    ;
Did_u_visit $visitor $name =
     “. You have been here
    before.</body></html>”
    ;
```

Froglingo – Concepts

Terms:

- constant ∈ term. e.g. 33, “string”.
- identifier ∈ term. e.g. id, John.
- variable ∈term. e.g. $x.
- term term ∈ term. e.g. John, John salary.
- term ‘,’ term ∈ term.

Assignments:

- term ‘=‘ term

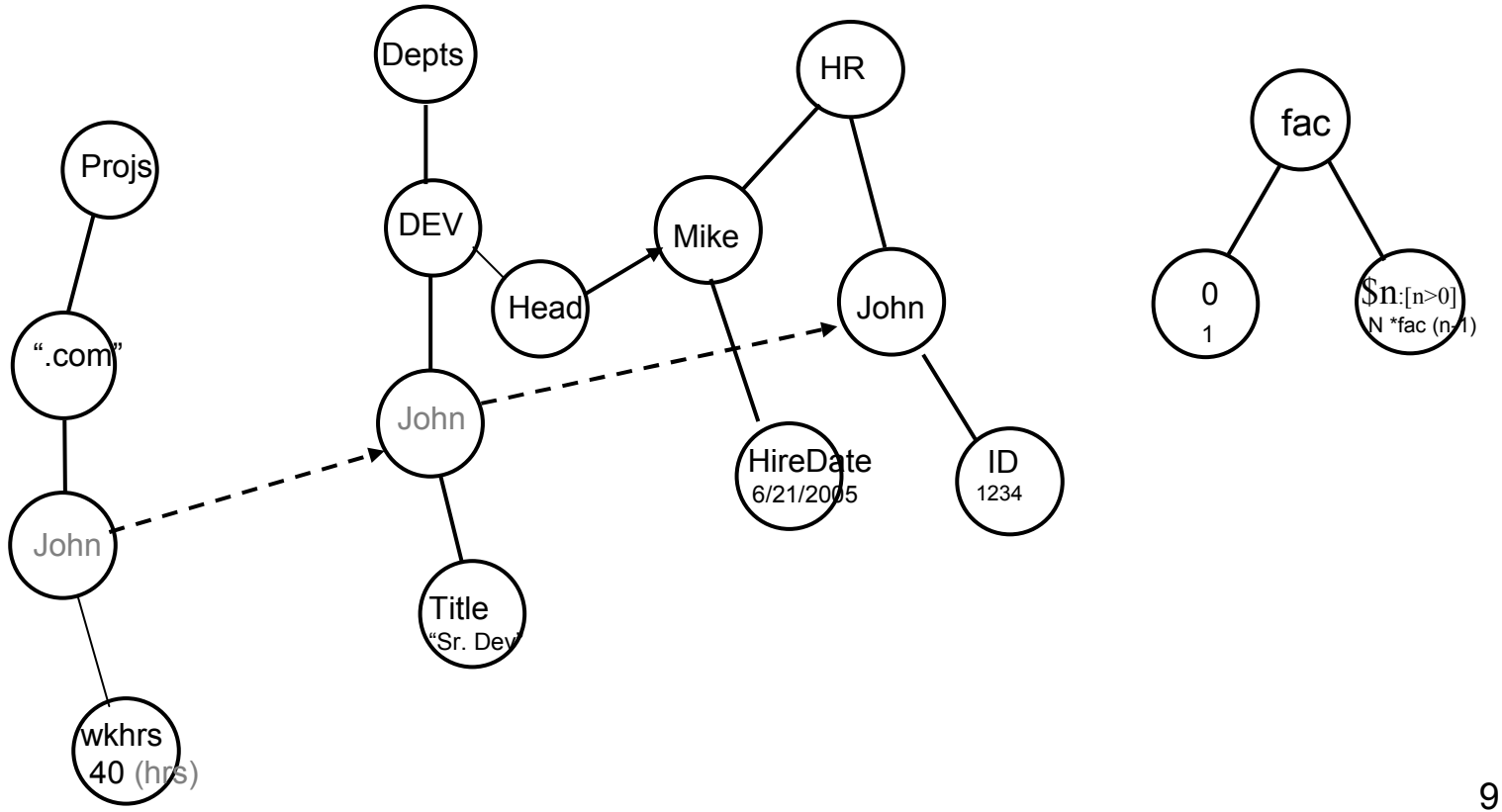
Froglingo - Concepts

Database - is a finite set of assignments.

HR John ID = 1234;
HR Smith hireDate = ‘6/21/2006’;
Depts DEV Head = HR Smith;
Depts DEV (HR John) title = “Sr. Dev”;
Projs “.com” (Depts DEV (HR John)) wkhrs = 40;
fac 0 = 1;
fac n:[n>0]=(n*(fac (n–1)));

Froglingo - Concepts

Tree-like Structure – equivalent of database



Froglingo - Higher Productivity

- Terms serve as the global names for data. It minimizes the need for intermediate variables commonly used in imperative languages. In other words, it has no need of a set-valued operation to find a single value as SQL does.

Froglingo – Higher Productivity

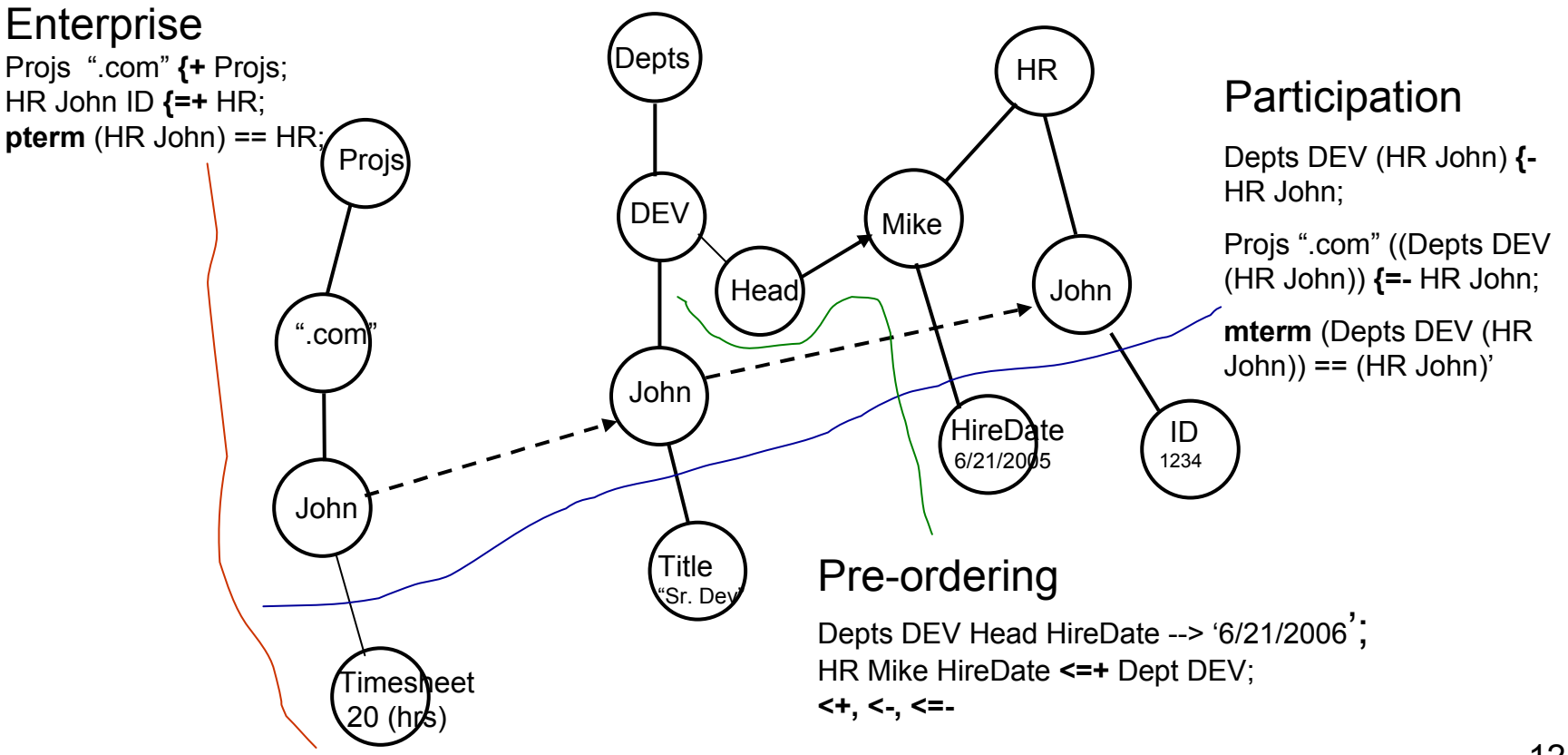
Code comparison between C# and Froglingo
Example: displaying the phone number of a given visitor

```
<!– a html file embedding ASP.NET code →
<html><body>
Your Phone Number: <asp:Label id="lblPhone" runat="server" />
</body></html>

<!-- C# code feeding phone number to the above ASP.NET code  -->
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="System.Data" %>

string GetPhoneNumber(string name){
        string PhoneNumber= null;
        System.Data.SqlClient.SqlConnection conn = new
        System.Data.SqlClient.SqlConnection(
                ConfigurationSettings.AppSettings[ "LocalConnStr" ] +
                ReturnPassValue() );
        conn.Open();
        System.Data.SqlClient.SqlCommand dc = new
        System.Data.SqlClient.SqlCommand("select phone where name=" +
        name, conn );
        try {
          PhoneNumber = System.Convert.ToString( dc.ExecuteScalar() );
        } catch  {
          PhoneNumber = "Systems error";
        }
        return phoneNumber;
}
void display_phone_page(Object s, DataListCommandEventArgs e) {
        pnlRequestPhone.Visible = false;
        pnlViewPhone.Visible = true;
        lblPhone.Text = getPhoneNumber(visitorName.text);
}
```

```
<!– a html file embedding Froglingo code →
<html><body>
Your Phone Number: <frog> visitor @name phone </frog>
</body><html>

<!– No need for extra function. The above html code pulls data by itself-→
```

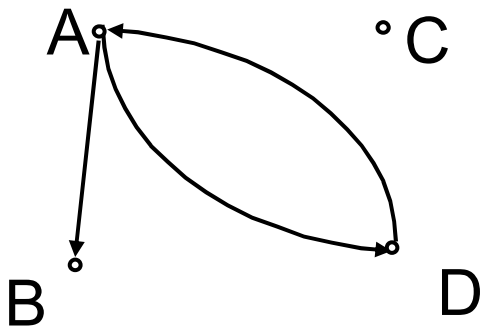
Froglingo – Higher Productivity

- Richer and more expressive built-in operators

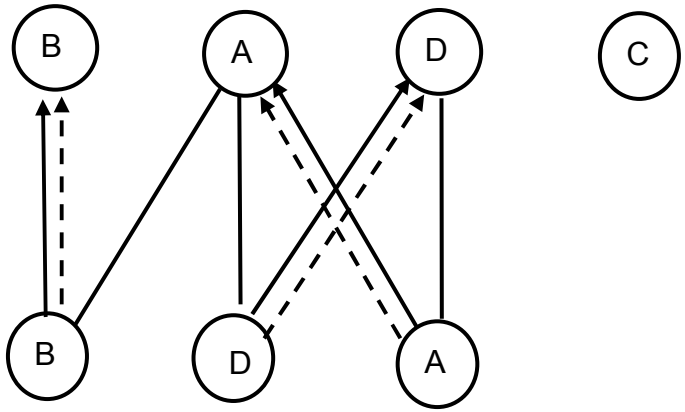


Froglingo – Higher Productivity

Example “Is there a path from A to Z in a directed graph?”



A Directed Graph



The EP Database

Define: A B = B;
A D = D;
D A = A;

Froglingo – Higher Productivity

```
Algorithm Is_a_path (G, v, z);
Input: G = (V, E) (a directed graph), v (a vertex of G), and
       z (a vertex of G).
Output: return true if there is a path from v to z, or false.
Begin
       mark v;
       if v is z itself, then return true;
       for all the directed edges (v, w) do
                  if w is unmarked then
                     return Is_a_path (G, w, z);
       return false;
End

Call Is_a_path (G, A, Z);
```

```
Z <=+ A
```

The built-in operators {=+, {=-, <=+, and <=- are more expressive than SQL, Datalog, and path-expression in graph-oriented structures. Datalog can do the path problem, but no gurantee of termination.

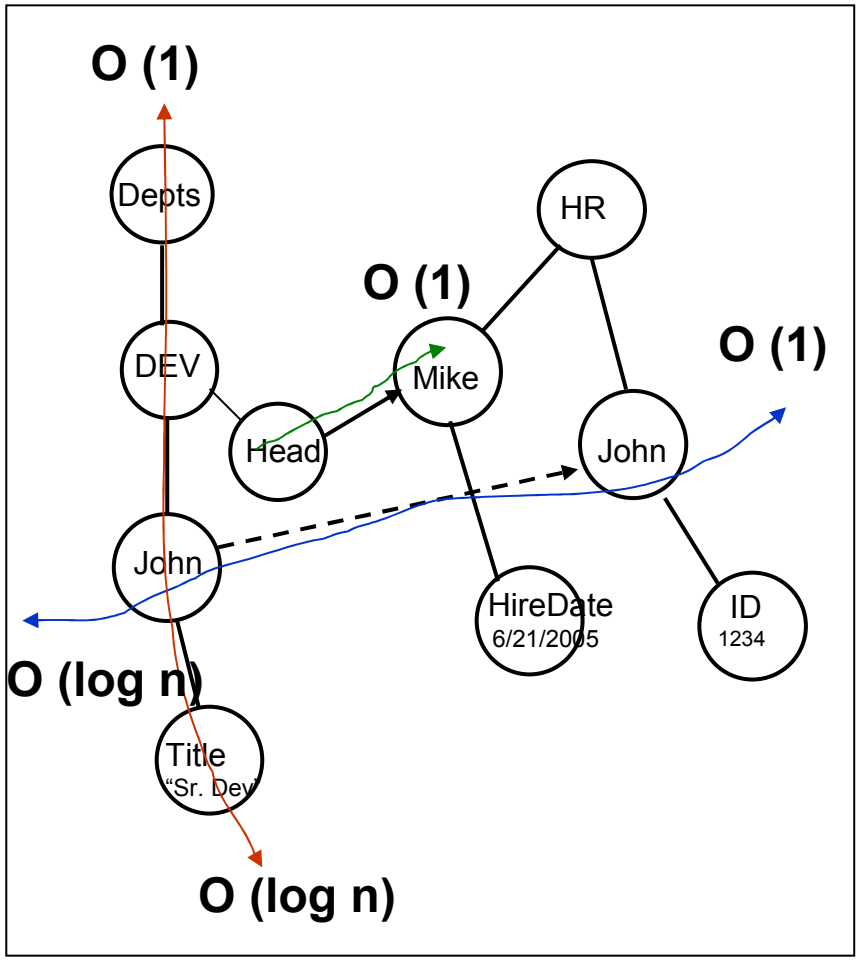
The path problem is just a classical example. The Froglingo built-in operators are extensively used in set-valued operations.

Froglingo – Higher Productivity

- Uniform language – term is the primary concept. C# has many: table, file, connection, binding, Label, Repeater, DataList, DataGrid, DataSet, ….
- Uniform storage. Reduce operation & maintenance effort.
- Functional programming declarativeness
  - Easy in error handling
  - For example: blog careless → null

Froglingo – Time Complexity

- Traveling along the trees-like structure is optimized.
- Arranging data differently costs less for many queries than SQL does.
- The time complexity for many queries not SQL-expressible is the best.



Froglingo - Time complexity

Case 1: Queries on m-to-m relationships using SQL join

- Example: a report of all the projects including member names.
- SQL spends O (n log (n)). Then Denormalization
- Froglingo spends O (n)

Emps

empId	name
1	"John"
2	"Smith"

Projs

projId	empId
10	1
10	2
20	1

```
select Projs.projId, Emps.name
from Emps, Projs
where Emps.empId = Projs.empId;
```

```
Emps 1 name = "John";
Emps 2 name = "Smith";
Projs 10 (Emps 1) = true;
Projs 10 (Emps 2) = true;
Projs 20 (Emps 1) = true;

select mterm $p, $e name
where $p $e == true;
```

Froglingo – Time Complexity

Case 2: Queries on shredded hierarchies using SQL select

- Example: find the phone number of the postal office in Bridgewater township, Somerset County, New Jersey, U.S.A.
- Assume at each layer, org has m sub orgs, and there are n nested levels in a hierarchy (total $m^{n+1} - 1$ nodes).
- SQL (by knowing the depths) spends $O(n^2 \log(m))$
- Froglingo spends $O(n \log(m))$

org

orgId	name	parent
0	US	1000
1	NJ	0
2	CA	0
3	SO	1
5	BR	3
6	OR	2
7	NL	6

OrgPhone

townId	phone
6	123-233-9999
7	405-566-9878

SQL Query:

```
select orgPhone.phone from org, OrgPhone
where org.name = "BR" and org.orgId = OrgPhone.townId
      and org.parent in
select org.orgId from org
where org.name = "SO" and org.parent in
select org.orgId from org
where org.name="NJ" and org.parent in
select org.orgId from org
where org.name="US"
```

Froglingo database

US NJ SO BR phone = "123-233-9999"

US CA OR NL phone = "405-566-9878"

Query: US NJ SO BR phone;

Froglingo – Time Complexity

Case 3: Queries not SQL-expressible

- Example 1:If is there a path from A to Z in a directed graph? -- O(n log(n))
- Example 2: Find all the information about John: `select $info where $info {=- HR John` -- (O(n)).

Why Froglingo?

Higher Productivity

- Terms serve as global naming. It says that SQL doesn't support a mapping from relations to a single value.
- More expressive built-in operators than SQL, Datalog, path expressions in semi-structured data.
- Uniform language and storage for both data and application logic.
- Functional programming declarativeness– less bugs, and "what rather than how".
- 10 times less code than Java in practice.

Best Performance

- Froglingo re-arranges SQL's many-to-many relationships to avoid using SQL join. The resulting query expressions are cheaper.
- Froglingo doesn't shred hierarchical data that SQL does. The resulting query expressions are cheaper.
- Froglingo reaches the best time complexity for many queries not expressible by SQL, DataLog, and path expressions in Semi-structured data model.

Froglingo – Related Work

- Relational model misses semantics by “shredding” application data.
- Hierarchical (including XML) model is incapable for general application data.
- Network model lacks a consistent way of managing hierarchical data vs cyclic data, and a feasible algorithm for competent performance.
- Froglingo has its unique tree-like data structure which unites the best features of the other data models and beyond.

Froglingo – Related Work

- Persistent Java
  - Unifies db & programming by keeping running states persistent.
  - Not much on set oriented operations.
- Machiavelli
  - Unifies db & programming by typed lambda calculus
  - On the top of a list of tuples.
- .NET (LINQ, Yukon, Cω, C#/XML/SQL)
  - Build bridges for DB, Objects, & Documents.
  - Minimize communication cost.
- Froglingo
  - Unifies db & programming by tree-like structure/λ–calculus.
  - Data and Application Logic are unified logically and physically.

Froglingo – Future Work

- Optimize Froglingo System to test its competent performance.
- Develop a formal method to precisely analyze the expressiveness of Froglingo built-in operators and the time complexity.