Hackhofer

TU WIEN

compilers languages

DSL in the Insurance business

- Motivation
- Objectives
- TreeCalc language
- Implementation

TreeCalc

Hackhofer Software GmbH
1070 Wien
www.hackhofer.com

DI Stefan Neubauer, 30.03.2023

Movitation – Overview UNIQA Group

- 18 markets
- > 30 insurance companies
- 15.000 employees
- 15,5 mn customers
- 25 mn insurance policies
- € 6.300 mn premium (2021)
- € 314,7 mn net profit (2021)

https://www.uniqagroup.com/gruppe/versicherung/media/files/Uniqa_GB21_DE_2021-04-07_DE_Komplett.pdf
https://press-uniqagroup.com/news-uniqa-vorlaeufiges-ergebnis-2021-ueber-erwartungen-bereits-angehobene-jahresprognose-nochmals-uebertroffen-?id=150682&menueid=1684&l=deutsch

Motivation – business view

Computations repository

Line of business

IT System

Point of Sale - Online

Direct sales - Online

H B1 | C B1 | P B1 | L B1

H B2 | C B2 | P B2 | L B2

...

Motivation

- VP/MS ® from DXC Technology
  - Main calculation engine for the UNIQA group
    - Modelling part: Eclipse based; used by actuaries / business units
    - Execution part: Native library (Windows, Linux, z/OS, AS/400)
  - Challenges / Cons
    - Proprietary, enterprise license model
    - Hard times if instabilities/errors with new versions
- TreeCalc
  - Open Source → https://github.com/treecalc
  - Safer, better, faster ☺
  - Conversion from VP/MS models to TreeCalc implemented (not open source)
  - Playground for memoization

Example – size of calculation models

- Health insurance, AT
- For point of sales and contract management systems

Kenngröße	P.O.S.	GUI	HOST	Fachmodell	Person	Berufe	Summe
Attribute	238	40	119	111	17	-	525
Attributproperties	710	65	140	124	25	-	1.064
Funktionen	222	49	74	429	-	5	779
Produktbaum Knoten	50	148	44	7	-	-	249
Produktbaum Properties	163	329	51	336	-	-	879
Formeln	1.116	354	284	896	25	5	2.680
Formel Zeilen	8.603	1.144	1.617	11.292	25	14	**22.695**
Formel Zeichen	332.645	33.399	70.044	439.013	270	213	875.584
Formel Zeilen ohne Kommentare	6.220	812	1.407	7.475	25	14	15.953
Formel Zeichen ohne Kommentare	249.059	24.286	60.681	294.748	270	213	629.257
Tabellen	24	138	3	43	12	1	221
Tabellen: Anzahl Zeilen	1.411	3.219	24	3.449	182	1.517	9.802
Tabellen: Anzahl Zellen	3.131	12.370	57	19.137	435	4.551	**39.681**
Tabellen: Anzahl Zeichen	66.902	62.823	632	126.849	3.832	35.057	296.095

Objectives

- Language
  - Declarative, „safe“: no destructive assignment, no loops, ..
  - Text format (diff/merge with git etc. for free)
  - Calculations organized in trees
- Simple API
  - set value
  - compute result
  - get list of key/value pairs
  - input needed?
- Execution
  - Optimal integration for: Java, JavaScript
  - Fast, safe, correct, scaleable

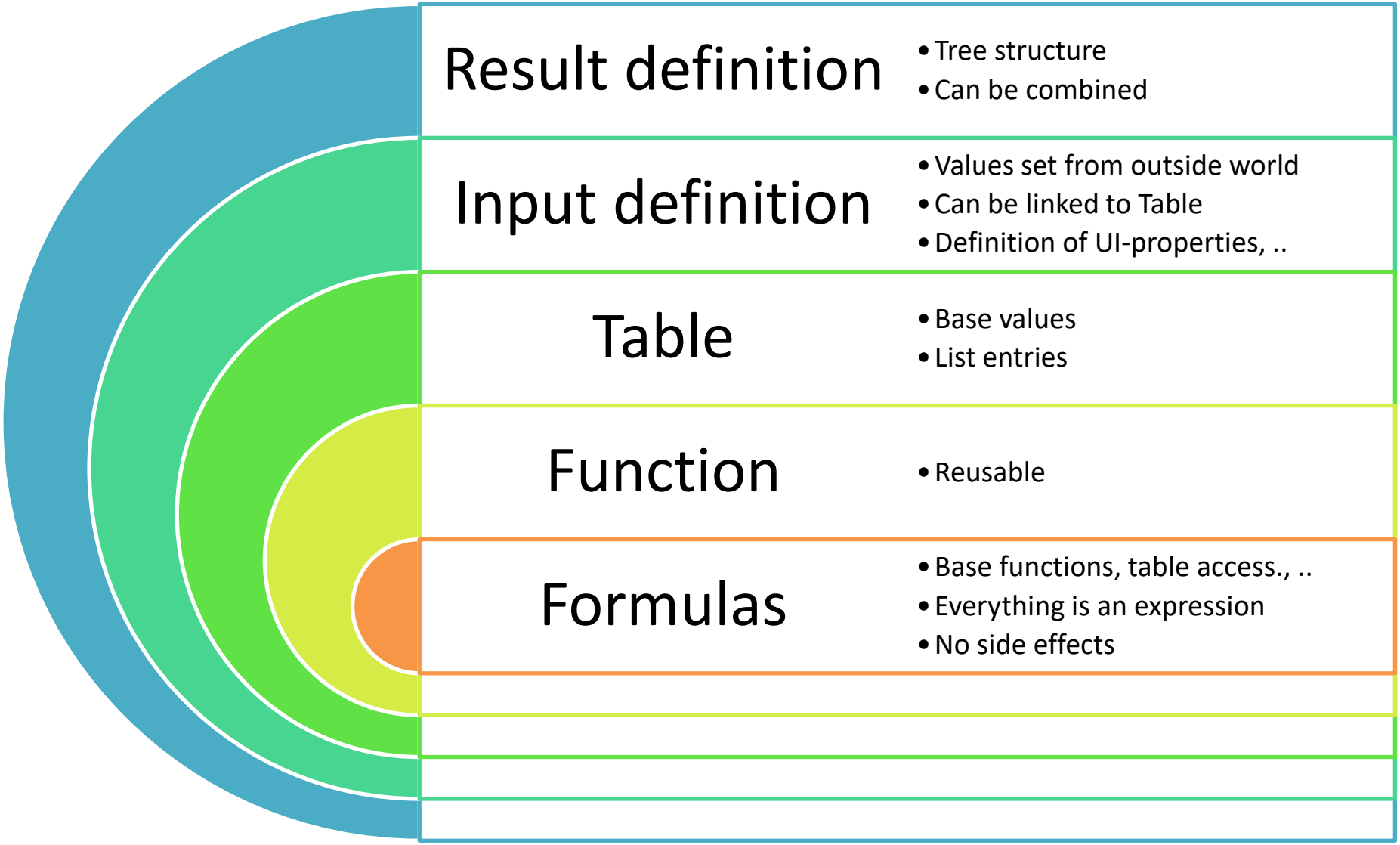
DSL implementation options

- Internal DSL / Embedded DSL
  - Hosted: Scala, Haskell, (Meta)OCaml, Lisp, Ruby, …
  - Fluent interface: Java, C#, ...
  - Main pro: „cheap“
  - Main cons: highly dependent on host language + implementation

External DSL

  - Custom syntax, custom parsing
  - Semantic model
  - Interpretation / Code generation
  - Main pro: most flexible
  - Main cons: high effort

Language - Overview



Language - Properties

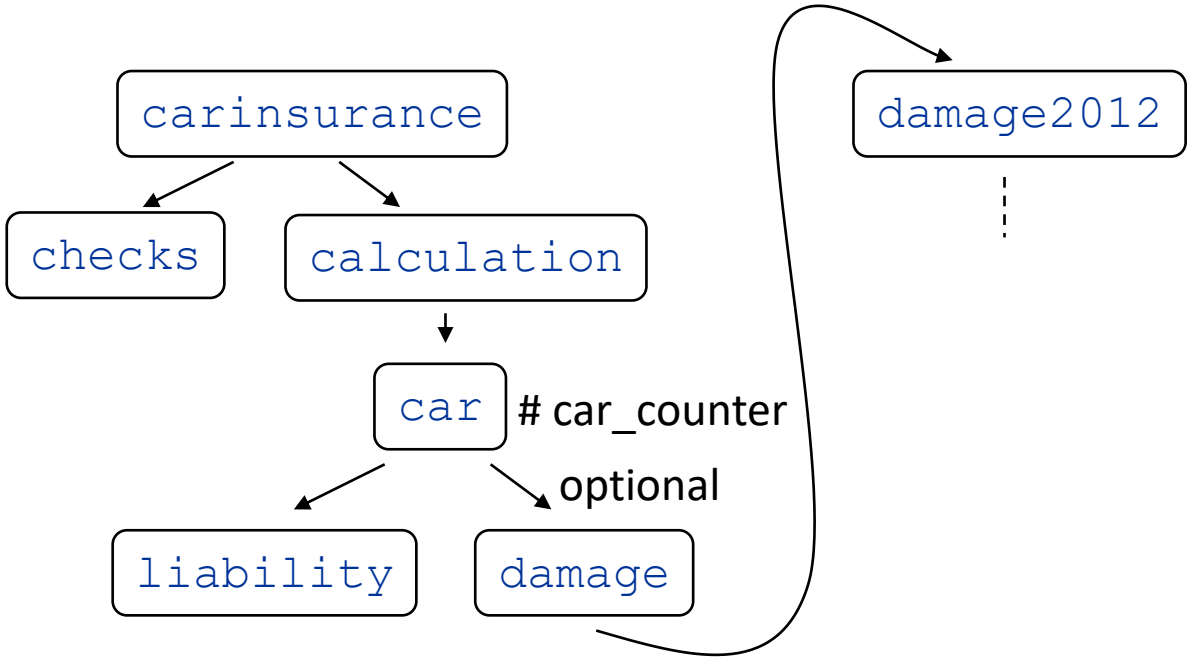
- Declarative / functional
  - No destructive assignment
  - Results only dependent on input values
  - Referential transparency
  - Recursion instead of loop
  - Everything is an expression
- Data types
  - Number (IEEE 754 binary double), String, Boolean, Date, List
  - Dynamic, weakly typed

Language – Example input defintion

```
INPUT I_DateOfBirth ;

INPUT I_Damage_Sum {
    visible = I_Damage_YN ;
    default = 10000;
    list = T_Damage_Sum ;
    select = key=1 || F_Age(I_DateOfBirth) > 30 ;
}
```

Language – Example tree

```
TREE carinsurance {
    NODE checks ;
    NODE calculation {
        NODE car TIMES I_CarCounter {
            NODE liability ;
            NODE damage IF I_Damage_YN {
                LINK damage2012;
            }
        }
    }
}
```



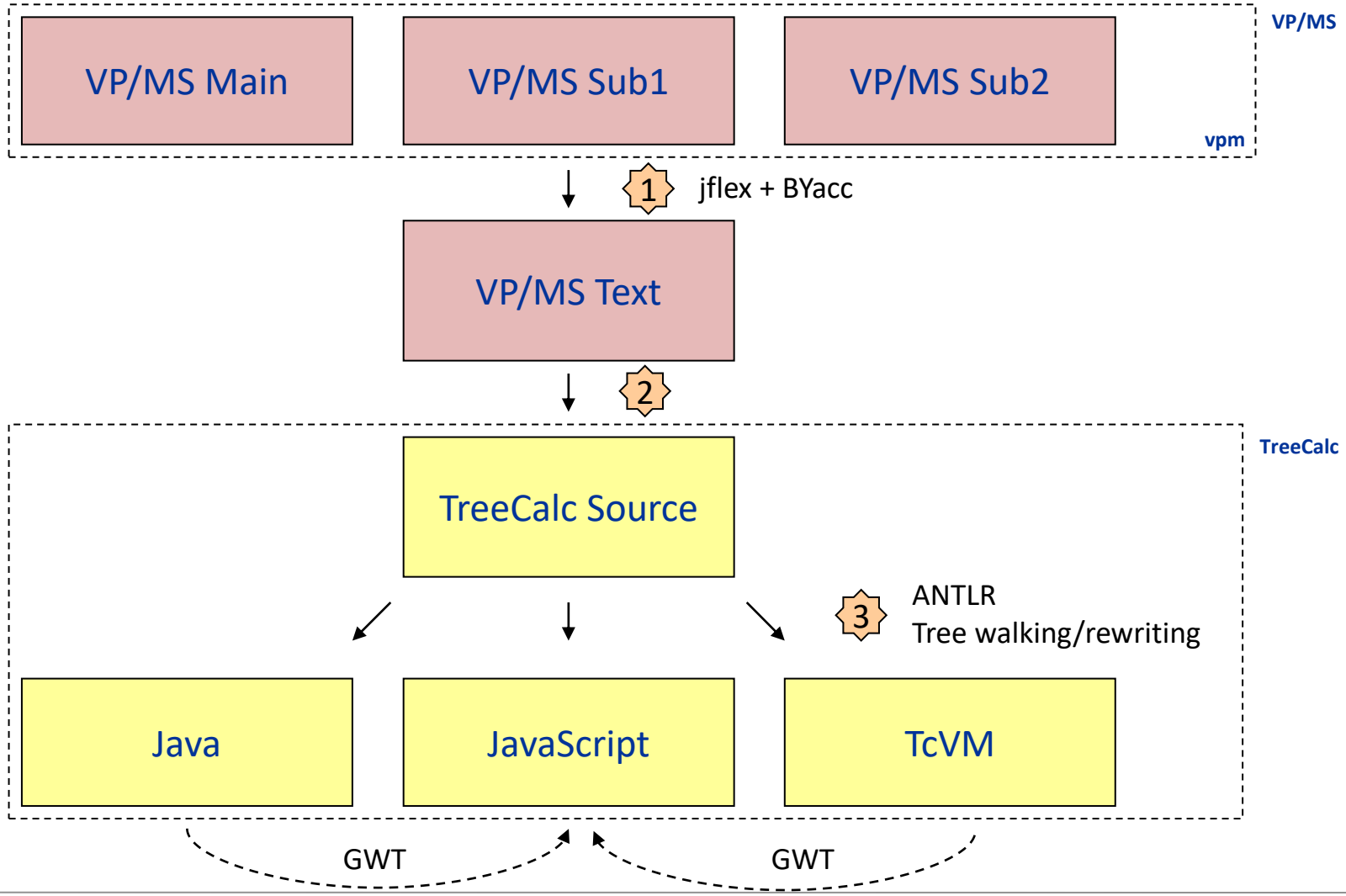
Language – Example result definition

```
CALC carinsurance.calculation {
     RX_Prem = R_Prem
                  *
               IF I_Discount_YN THEN
                     0.8
               ELSE
                     1
               ENDIF ;
}

CALC carinsurance.calculation.car.liability {
     R_Prem = I_kw * T_Area[I_Area].fact ;
     ...
}

CALC damage2012.calculation {
     R_Prem = ...
}
```

Language – Example table definition

```
TABLE T_Mortality (age, qx, qy) {
   16, 0.0006380, 0.0003980 ;
   17, 0.0007200, 0.0004160 ;
   18, 0.0007760, 0.0004060 ;
   19, 0.0008060, 0.0003720 ;
   20, 0.0008400, 0.0003580 ;
    ...
}

TABLE T_Liability_Sum (key, text) {
   1, "€  6.000.000,-" ;
   2, "€ 12.000.000,-" ;
}
```

Implementation – Overview



Implementation – Tools and strategies

- ANTLR
  - Lexer + Parser + Tree construction
  - LL(*), semantic/syntactic predicates
  - Nice grammar, lots of tools, automatic error correction, …
  - Generated code a lot bigger than lex/yacc & co
- Implementation strategies
  - Homogeneous AST
  - External visitor for passes
    1. Scopes
    2. Symbol table
    3. Resolve names + rewrite AST
    4. Fill semantic model

```
void visitNodes(Ast node) {
    switch (node.getType()) {
    case TT_COMPUNIT: {
      List<Ast> c = node.getChildren();
      for (Ast child : c) {
        visitNodes(child);
      }
    break;
    }
    case KEYWORD_TREE: {
     ...
```

Implementation – Code generation

- out.print(...) ☺
- Simple because of semantic model + AST
  - Same model and helper classes
  - Separate writer classes for Java, JavaScript, TcVM
- Formulas
  - intermediate vars _1, _2, ...
  - AST node
    - → optional: out.print(...)
    - → expression string (short expression or varname)

Implementation – Generated Java source

```
F_LI_Lx(age; sex; risk):
  if(age <= 0;
     100000;
     F_LI_Lx(age-1; sex; risk) * (1 - F_LI_qx(age-1; sex; risk))
  )
```

```
static final V F_LI_LX(S _s, V age, V sex, V risk) {
    Object cacheKey = _s.getCacheKey(8156345, age, sex, risk);
    V ret = _s.readCache(cacheKey);
    if (ret!=null) { return ret; }
    V _1;
    V _2 = age.smleq(_i0);
    if (_2.booleanValue()) {
       _1 = _i100000;
    } else {
       V _3 = age.sub(_i1);
       V _4 = age.sub(_i1);
       V _5 = _i1.sub(F.F_LI_QX(_s, _4, sex, risk));
       V _6 = F.F_LI_LX(_s, _3, sex, risk).mult(_5);
       _1 = _6;
    }
    ret = _1;
    _s.writeCache(cacheKey, ret);
    return ret;
}
```

Implementation - Java highlights

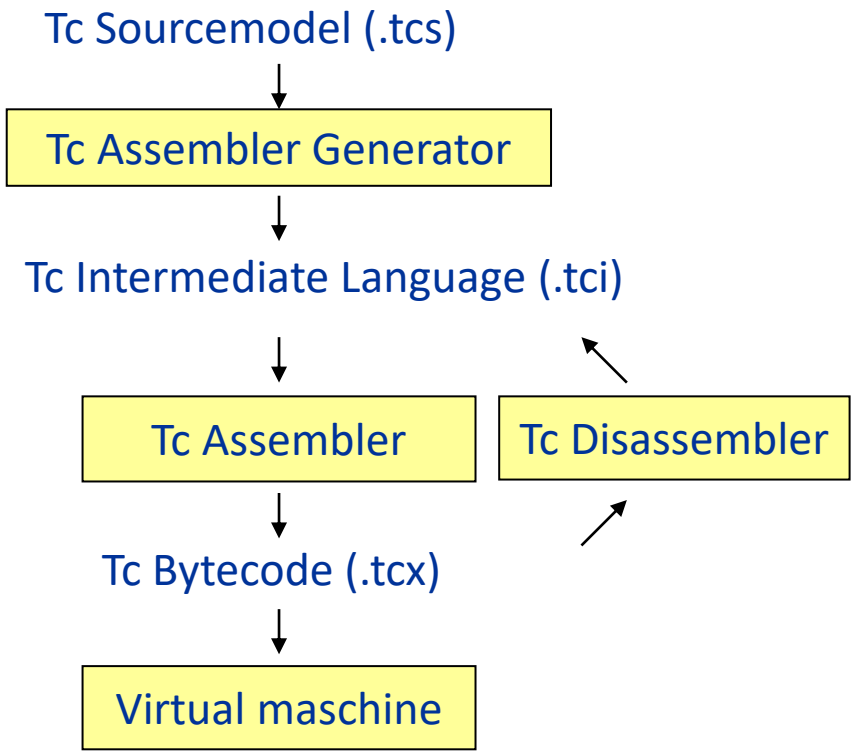
- Performance tweaks
  - Internal tables: sorted or direct access → O(1) or O(log n)
  - final static methods, constant pool, ids instead of names
  - Bit sets for node structure
  - switch(id) { ... } instead of reflection
    - for dynamic dispatch (calc, table, function, ..)
  - Caching
    - LRU instead of HashMap
    - random id; fast key object creation
    - no caching for simple formulas (heuristic)
- Handicaps
  - 16bit limits → split methods/classes
  - Big code blocks for dynamic dispatch

Implementation - JavaScript

- Some things easier
  - Dynamic dispatch
  - No 16bit limits
- Some things harder
  - Missing base libaries: HashMap, .. (got better)
  - Development environment
- Handicaps
  - Large models → big JavaScript source

Implementation - TcVM

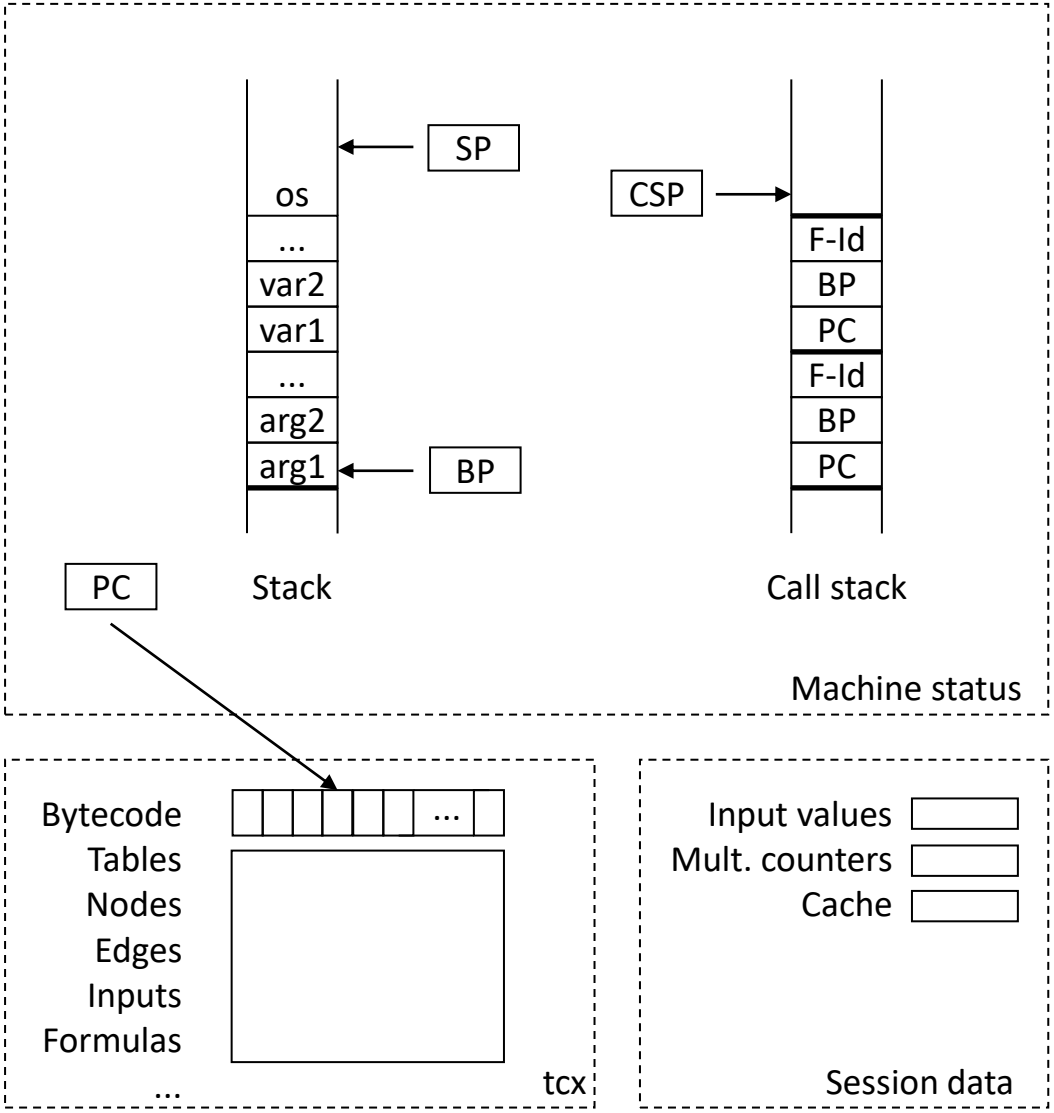
- Why
  - Optimized for size
  - Easy to play around with
- Implementation
  - TcVM implementated in Java
  - For some models too slow



Implementation – TcVM example

```
.func func=1 name=F_FACT args=1 simple=false formula=1
.formula formula=1 ; line 6830
   //start of if statement, line 6830
   : load 0 ; N
   : pushconst 0
   : cmpsmleq
   : iffalse L0
   : pushconst 1
   : goto L1
 L0:
   : load 0 ; N
   : dup
   : pushconst 1
   : sub
   : callfunc 1 ; F_FACT
   : mult
 L1:
   //end of if statement
   : return
```

TcVM - internals



TcVM - highlights

- Numbers
  - Functionality: 100 %
  - Performance: 10 times slower
  - Size: tcx size around 15 % of generated Java class file size
- Code generation
  - Text representation (.tci) proved to be useful
  - Generation simpler than Java/JavaScript source code generation
- Implementation
  - Tree access with TcVM „macro programs“, not neatest solution
  - Base classes reused from Java implementation

Wish list

- Extensions
  - VP/MS to TreeCalc converter Open Source (clarify legal situation first)
  - Partial generation (e.g. just for UI-control)
  - Modelling environment (Xtext, …)
- Organization/Community
  - Build up / extend user group and contributors
  - Landing page, Wiki, Tutorial, Bug tracker, …
  - Books, Training and Certifications ;-)
- Improvements
  - Tooling: build chain, maven plugin, … (learn from Clojure, Frege, ..)
  - Add modules concept
  - Test cases + examples
  - Upgrade from ANTLR 3 to ANTLR 4
  - Improve JavaScript port
  - Implement with Truffle / GraalVM
  - Performance optimizations: Type inference, dynamic memoizaiton, ..
  - Implement features from VP/MS® Runtime XE

Interested? Contact sneubauer@gmx.at

References

- Domain Specific Languages, Martin Fowler, Addison-Wesley, 2010
- Language Implementation Patterns, Terence Parr, Pragmatic Bookshelf, 2009
- The Definitive ANTLR Reference, Terence Parr, Pragmatic Bookshelf, 2007
- http://www.antlr.org/
- https://github.com/treecalc