Cedalion: A Language for Language Oriented Programming

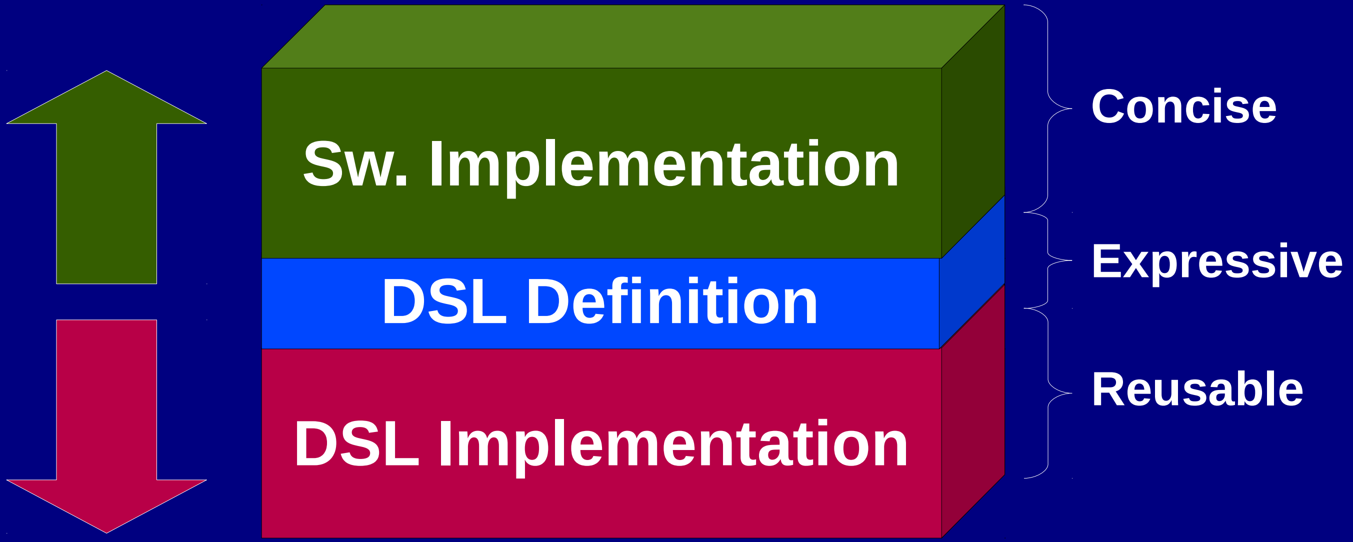
Boaz Rosenan
Dept. of Mathematics and Computer Science
The Open University of Israel

Joint Work With:
David H. Lorenz

Language Oriented Programming (LOP): Rethinking Software Development

- **Traditional Thinking**
  - Designing our ***software*** for a *programming language*.
- **New Thinking**
  - Design (domain specific) *programming languages* for our ***software***.

LOP: Middle Out


Sw. Implementation

DSL Definition

DSL Implementation

Concise

Expressive

Reusable

DSL State of the Art

- **External DSLs**
  - Implemented as compilers/interpreters.
- **Internal DSLs**
  - Implemented as libraries in a host language.
- **Language Workbenches**
  - IDEs for developing and using external DSLs.

What makes one approach better then the other?

DSL “Bill of Rights”

- *Freedom of Expression*
  - *Syntactic*
  - *Semantic*
- *Economic Freedom*
  - *Cost effective Implementation*
  - *Cost effective Usage*
- *Freedom of Assembly*
  - *DSL Interoperability*

Comparison of Approaches

	External DSLs
Freedom in Definition	☺
Cost effective Implementation	☹
Cost effective Usage	☹
DSL Interoperability	☹

Comparison of Approaches

	External DSLs	Internal DSLs
Freedom in Definition	☺	☹
Cost effective Implementation	☹	☺
Cost effective Usage	☹	☹
DSL Interoperability	☹	☺

Comparison of Approaches

	External DSLs	Internal DSLs	Language Workbenches
Freedom in Definition	☺	☹	☺
Cost effective Implementation	☹	☺	☹
Cost effective Usage	☹	☹	☺
DSL Interoperability	☹	☺	☺

Comparison of Approaches

	External DSLs	Internal DSLs	Language Workbenches
Freedom in Definition	☺	☹	☺
Cost effective Implementation	☹	☺	☹
Cost effective Usage	☹	☹	☺
DSL Interoperability	☹	☺	☺

Comparison of Approaches

	External DSLs	Internal DSLs	Language Workbenches	Cedalion
Freedom in Definition	☺	☹	☺	☺
Cost effective Implementation	☹	☺	☹	☺
Cost effective Usage	☹	☹	☺	☺
DSL Interoperability	☹	☺	☺	☺

Cedalion: A Language Oriented Programming Language

- A programming language designed for LOP
  - Designed as a host for internal DSLs.
- Extensible, compositional syntax
  - Through projectional editing.
- Extensible semantics
  - Through logic programming.

Cedalion Website:
http://cedalion.sf.net

Cedalion Language Overview

- **Syntax**
  - Structure (abstract syntax)
  - Default Projection
  - Projection Definition
- **Semantics**
  - Type System
  - Logic Programming
  - DSLs in Cedalion

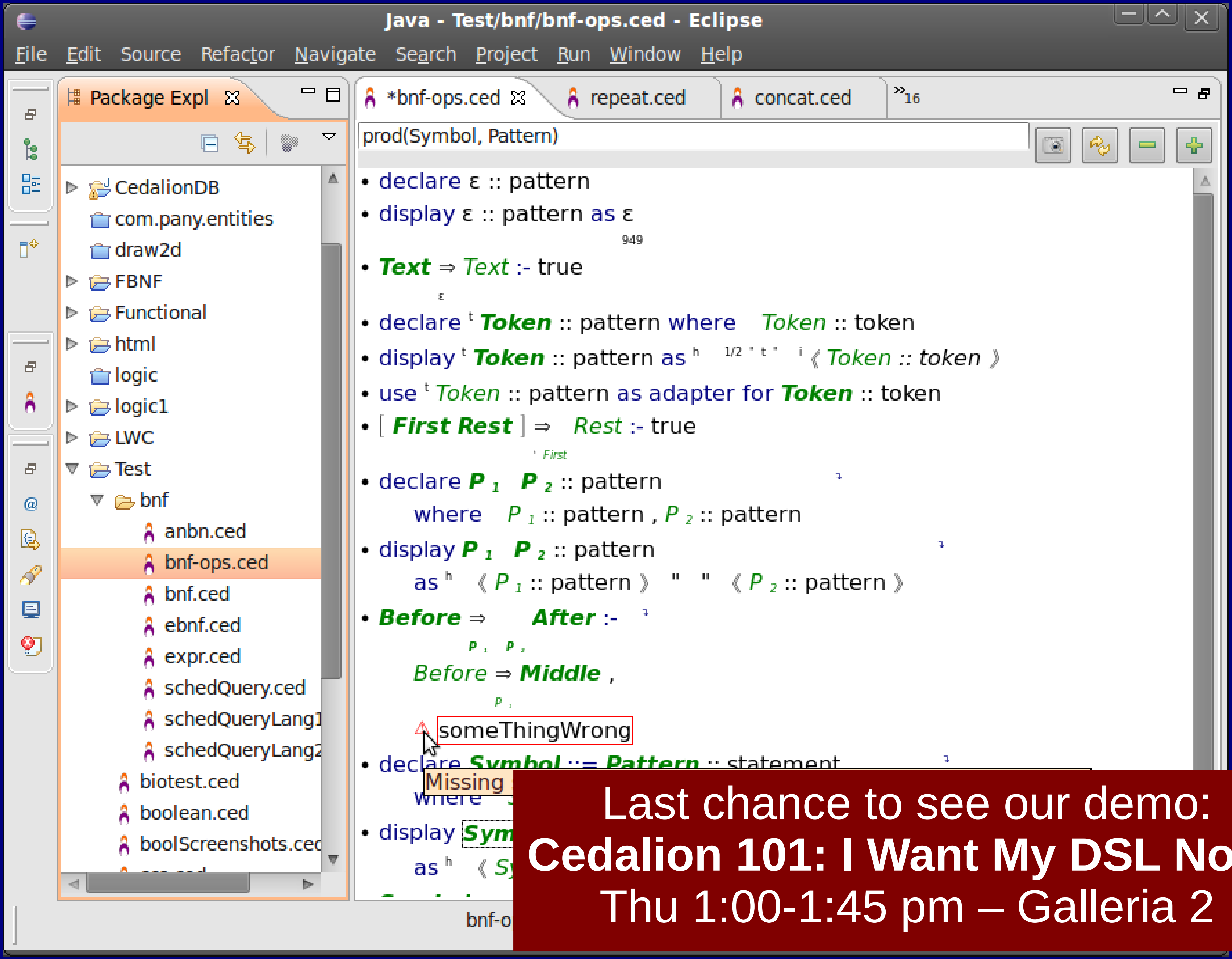
Abstract Syntax

- The AST of a Cedalion program is a term.
- A term can be:
  - A number.
  - A string.
  - A logic variable.
  - A compound term.
- A compound term has a name (ID), and zero or more arguments, which are terms.

Projectional Editing

- Cedalion uses projectional editing
  - Instead of parsing text to AST, AST is projected as text.
- Cedalion's syntax
  - Includes font style, color, layout and special symbols.
  - Supports ambiguities.

The Cedalion Workbench



```
prod(Symbol, Pattern)

• declare ε :: pattern
• display ε :: pattern as ε
                                949
• Text ⇒ Text :- true
        ε
• declare ᵗ Token :: pattern where   Token :: token
• display ᵗ Token :: pattern as ʰ   1/2 " t "   ⁱ 《 Token :: token 》
• use ᵗ Token :: pattern as adapter for Token :: token
• [ First Rest ] ⇒   Rest :- true
                  ' First
• declare P 1   P 2 :: pattern
     where   P 1 :: pattern , P 2 :: pattern
• display P 1   P 2 :: pattern
     as ʰ   《 P 1 :: pattern 》   "   "   《 P 2 :: pattern 》
• Before ⇒      After :-
            P 1   P 2
     Before ⇒ Middle ,
             P 1
     someThingWrong
• declare Symbol ::= Pattern :: statement
     Missing
     where
• display Sym
     as ʰ   《 Sy
```

Last chance to see our demo:

Cedalion 101: I Want My DSL Now!

Thu 1:00-1:45 pm – Galleria 2

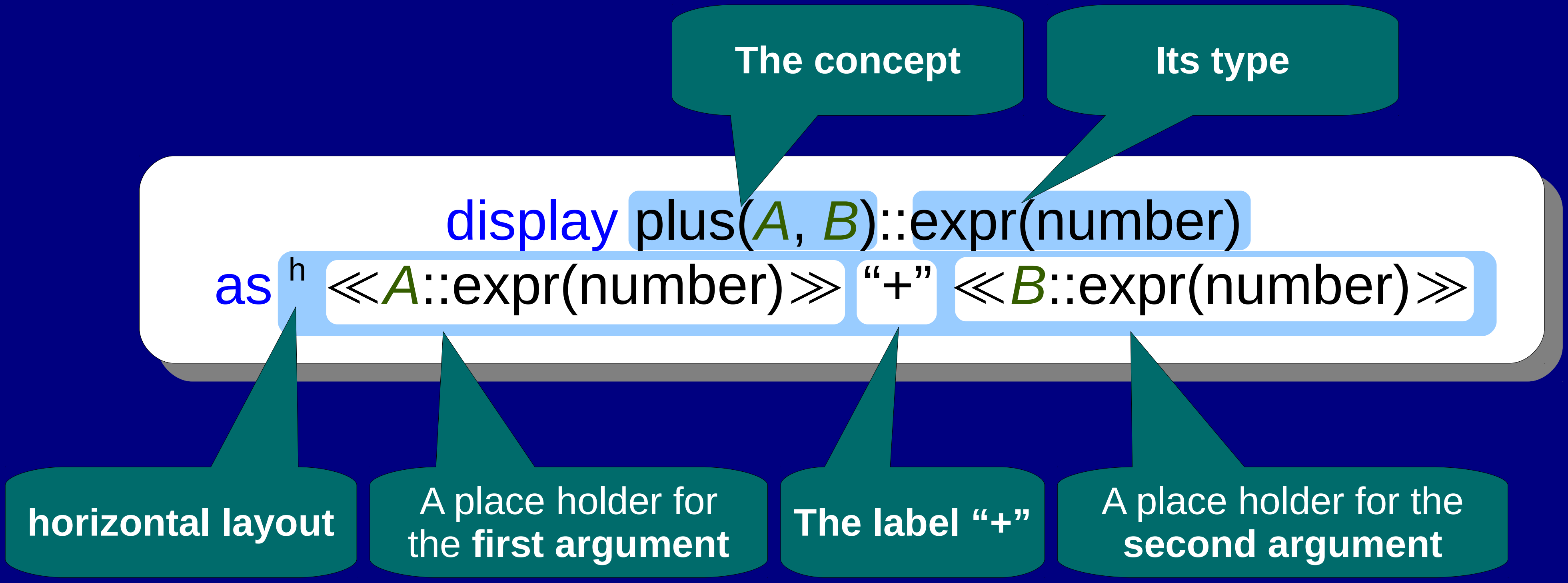
Projecting Terms

Cedalion provides rules for projecting terms.

- Strings: Depicted in magenta.
- Numbers: Depicted as decimals.
- Logic variables: Depicted in *green italics*.
- Compoun terms: Defined by the user...

Projection Definition

- The projection of compound terms can be customized using projection definitions.
- Such definitions tell Cedalion how to visualize some kind of compound term (concept).



The concept

Its type

display plus(*A*, *B*)::expr(number)

as ^{h} ≪*A*::expr(number)≫ “+” ≪*B*::expr(number)≫

horizontal layout

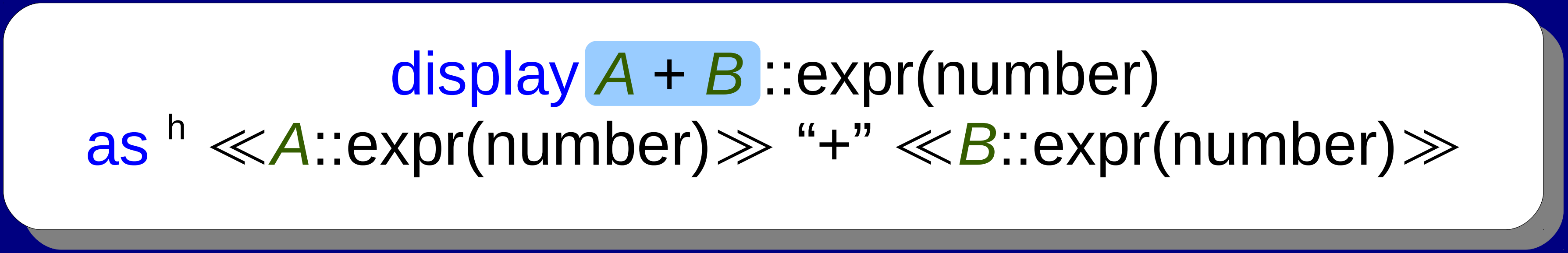
A place holder for the **first argument**

The label “+”

A place holder for the **second argument**

Projection Definition

- The projection of compound terms can be customized using projection definitions.
- Such definitions tell Cedalion how to visualize some kind of compound term (concept).



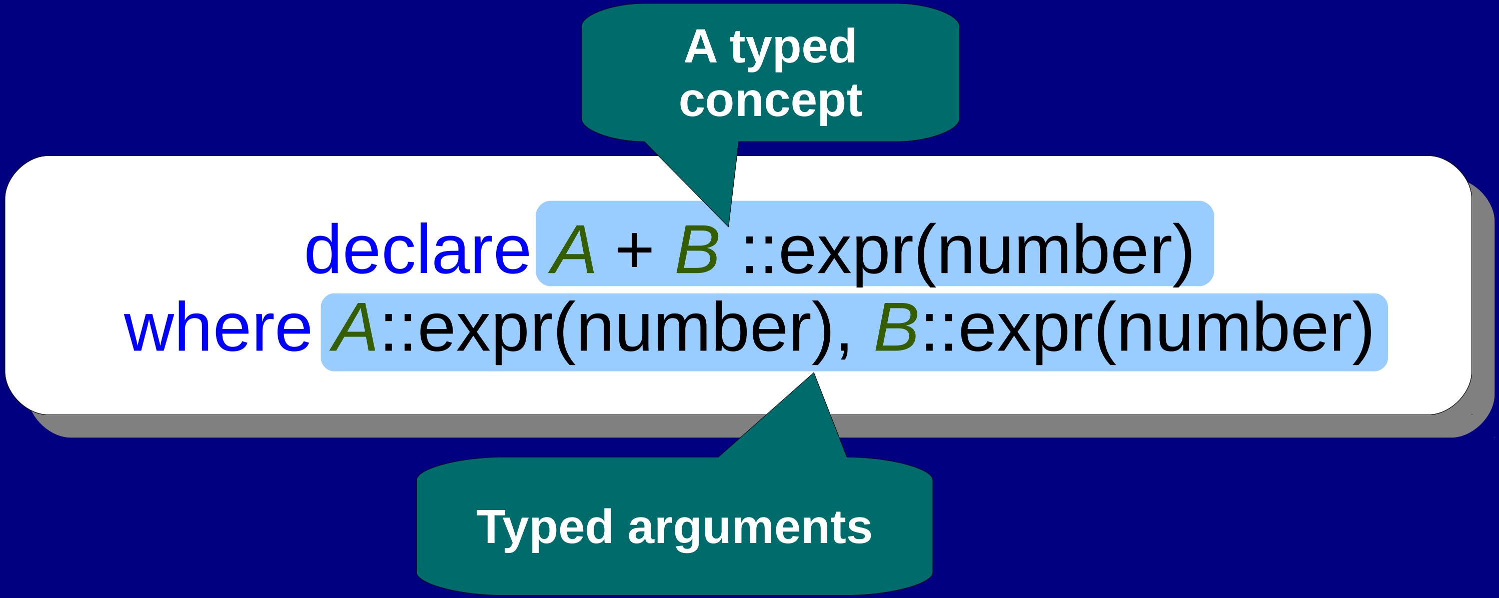
display A + B ::expr(number)
as h ≪A::expr(number)≫ “+” ≪B::expr(number)≫

Cedalion's Semantics

- **Static Semantics:**
  - Checkers define domain specific validity rules.
  - Cedalion's type system is also implemented as a set of checkers.
- **Dynamic Semantics:**
  - Logic programming.

Cedalion's Type System

- Concepts must be declared with a type signature.
- Type inference is used to infer the types of variables.



A typed concept

```
declare A + B ::expr(number)
where A::expr(number), B::expr(number)
```

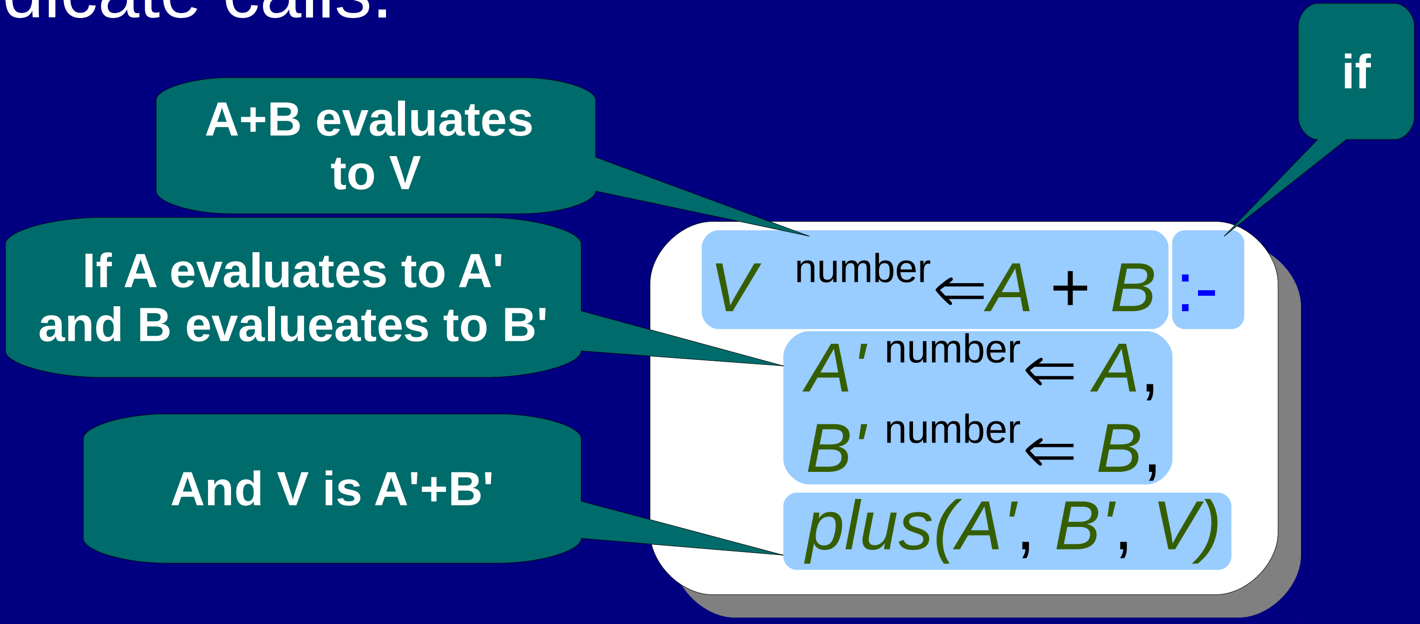
Typed arguments

Cedalion's Dynamic Semantics

- Cedalion is a logic programming language.
- A Cedalion program consists of a set of statements, which can be:
  - Deduction rules
  - Rewrite rules
  - Statements that evaluate to deduction rules through rewrite rules.
- A Cedalion program is evaluated by querying predicates. Predicates are defined using deduction rules.

Deduction Rules

- Deduction rules come from Prolog.
- They have the form: *Head* :- *Body*.
  - *Head* is a compound term of the predicate we define.
  - *Body* is a goal, consisting of a conjunction of predicate calls.



$V \overset{\text{number}}{\Leftarrow} A + B$:-

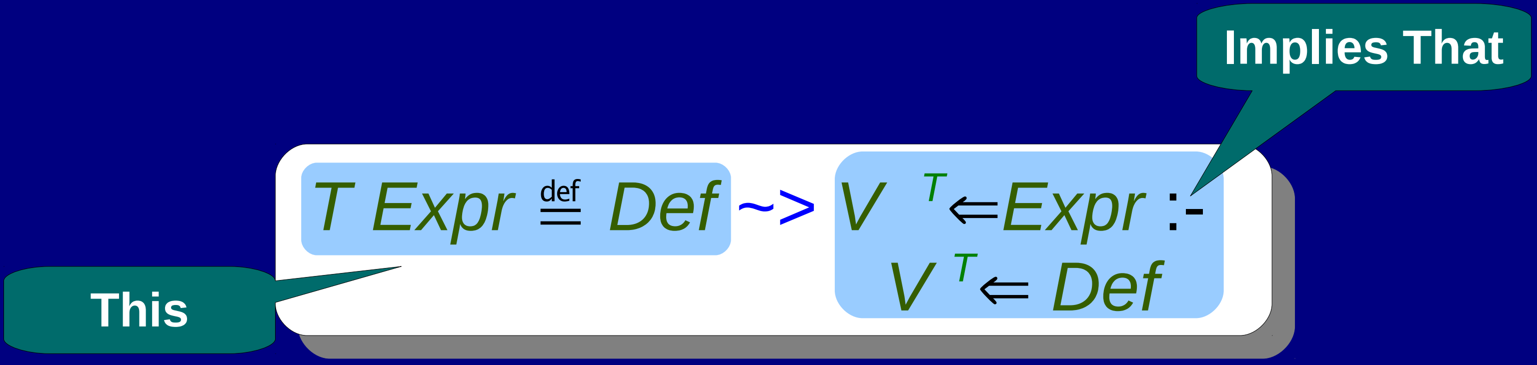
$A' \overset{\text{number}}{\Leftarrow} A$,

$B' \overset{\text{number}}{\Leftarrow} B$,

plus(A', B', V)

Rewrite Rules

- Rewrite rules transform user-defined statements to deduction rules.
- They take the form: S_1 ~> S_2 where:
  - S_1 is a pattern matching the defined statement.
  - S_2 is matching the statement S_1 is equivalent to.
- A Cedalion statement has a meaning if there is a sequence of rewrite rules translating it to at least one deduction rule.



$T\ Expr \stackrel{def}{=} Def$ ~> $V\ {}^{T}\!\Leftarrow Expr$:- $V\ {}^{T}\!\Leftarrow Def$

Implies That

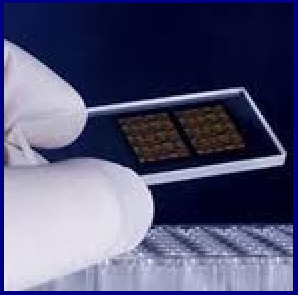
This

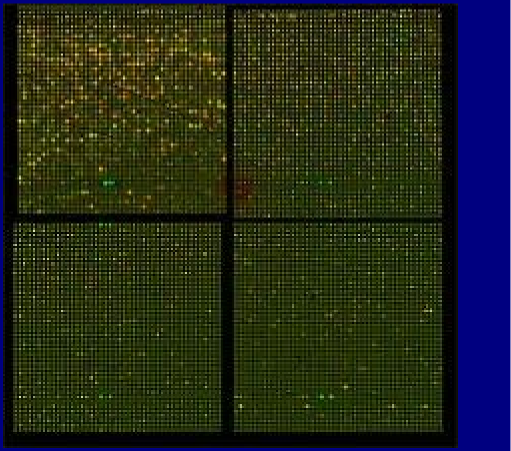
Defining a DSL in Cedalion

- Abstract syntax
  - Concept declarations.
- Concrete syntax
  - Projection definitions.
- Semantics
  - Rewrite rules, deduction rules and other statements.

Case Study: DNA Microarray Design

- Biologists use customized DNA microarrays in research.
- Each microarray has contains $O(10^5)$ unique sequences.
- We provided a DSL for microarray design.
- This DSL was used by our colleagues at the Technion IIT to design a real life DNA microarray.





This case study is joint work with
Itai Beno and **Tali E. Haran**,
Department of Biology,
Technion – Isreal Institute of Technology

Case Study Results

- The DSL was intuitive enough to allow our colleagues (biologists) to **understand** and to **modify** designs.
- Cost Effective:
  - DSL implementation: 1 day.
  - Initial design: 1 hour.
  - Each modification: 1-2 minutes.
  - Generating the microarray: ~6 minutes.
- DSL for expressing DNA microarray designs, interoperable with other DSLs.

Additional Examples and Case Studies

- Train Schedule Example (full source-code), in our paper.
- Functional Programming.
- Process Calculus (CCS) + Modal Logic (HML).
- Language Workbench Competition of 2011 (LWC11) submission.
- A calculator product-line, comparison with MPS [Lorenz and Rosenan, 2011].

All source-code can be found on the
Cedalion source-code repository, at
http://cedalion.sf.net

Related Work

- **Language Oriented Programming**
  - [**Ward, 1994**] Language-oriented programming. Software-Concepts and Tools, 15(4):147–161, 1994
  - [**Fowler, 2005**] Language workbenches: The killer-app for domain specific languages. 2005.
- **Language Workbenches**
  - [**Dmitriev, 2004**] Language oriented programming: The next programming paradigm. JetBrains onBoard, 1(2), 2004.
  - [**Simonyi, Christerson, and Clifford, 2006**] Intentional software. ACM SIGPLAN Notices, 41(10):451–464, 2006.
- **Internal DSLs**
  - [**Hudak, 1996**] Building domain-specific embedded languages. ACM Computing Surveys (CSUR), 28(4es), 1996.

Conclusion

- **Cedalion presents a novel approach to LOP**
  - DSL user
    - You can insist on using your preferred notation.
  - DSL designer
    - DSLs can be cost effective.
  - DSL tool developer
    - Internal DSLs form a better stepping stone than external DSLs.
- **Future Work**
  - Formal semantics for Cedalion.
  - The next big step: Cedalion for web applications.

Thank You!

Boaz Rosenan
David H. Lorenz
Dept. of Computer Science
The Open University of Israel

brosenan@cslab.openu.ac.il
http://cedalion.sf.net