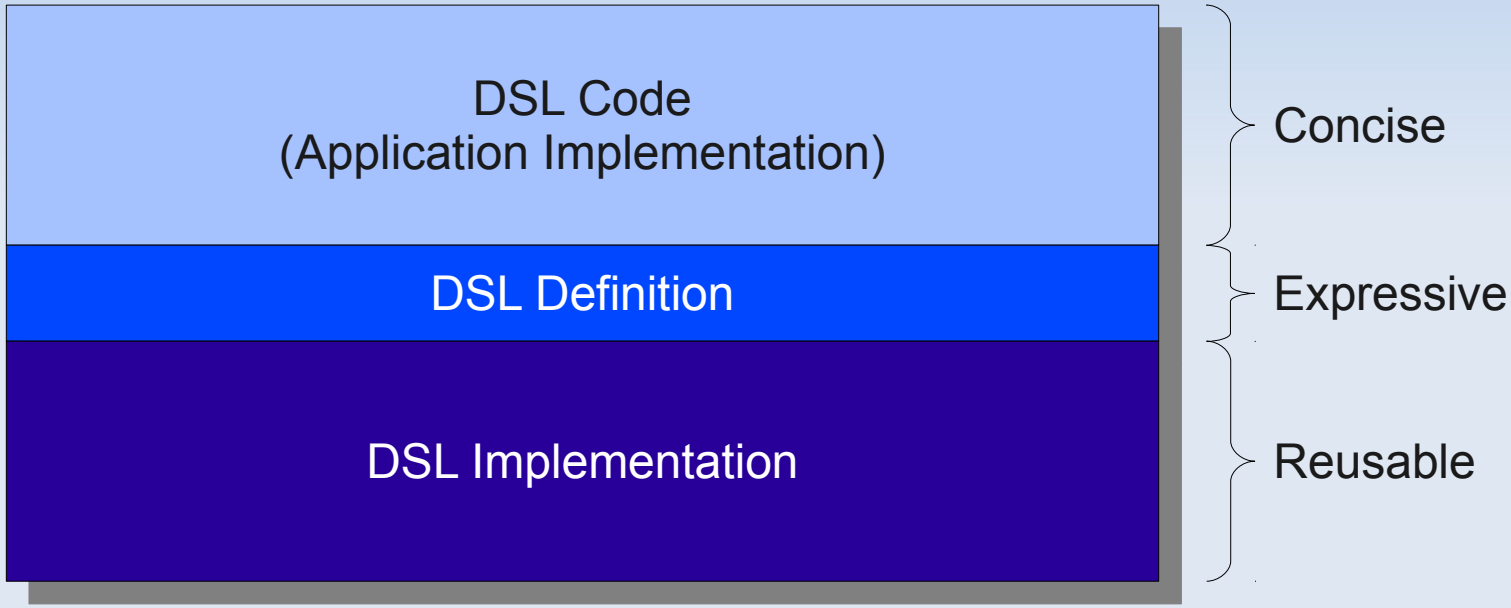
Cedalion: A Language Oriented Programming Language

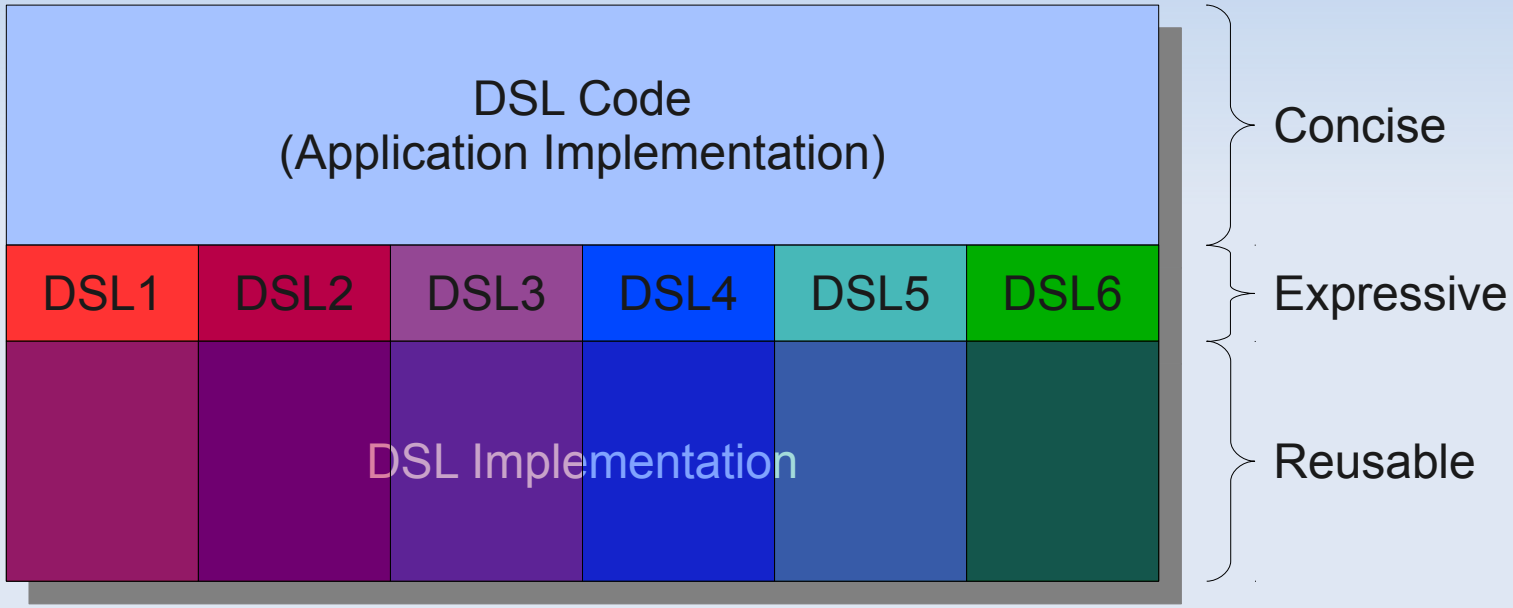
Boaz Rosenan
Dept. of Computer Science
The Open University of Israel

Joint Work With:
David H. Lorenz

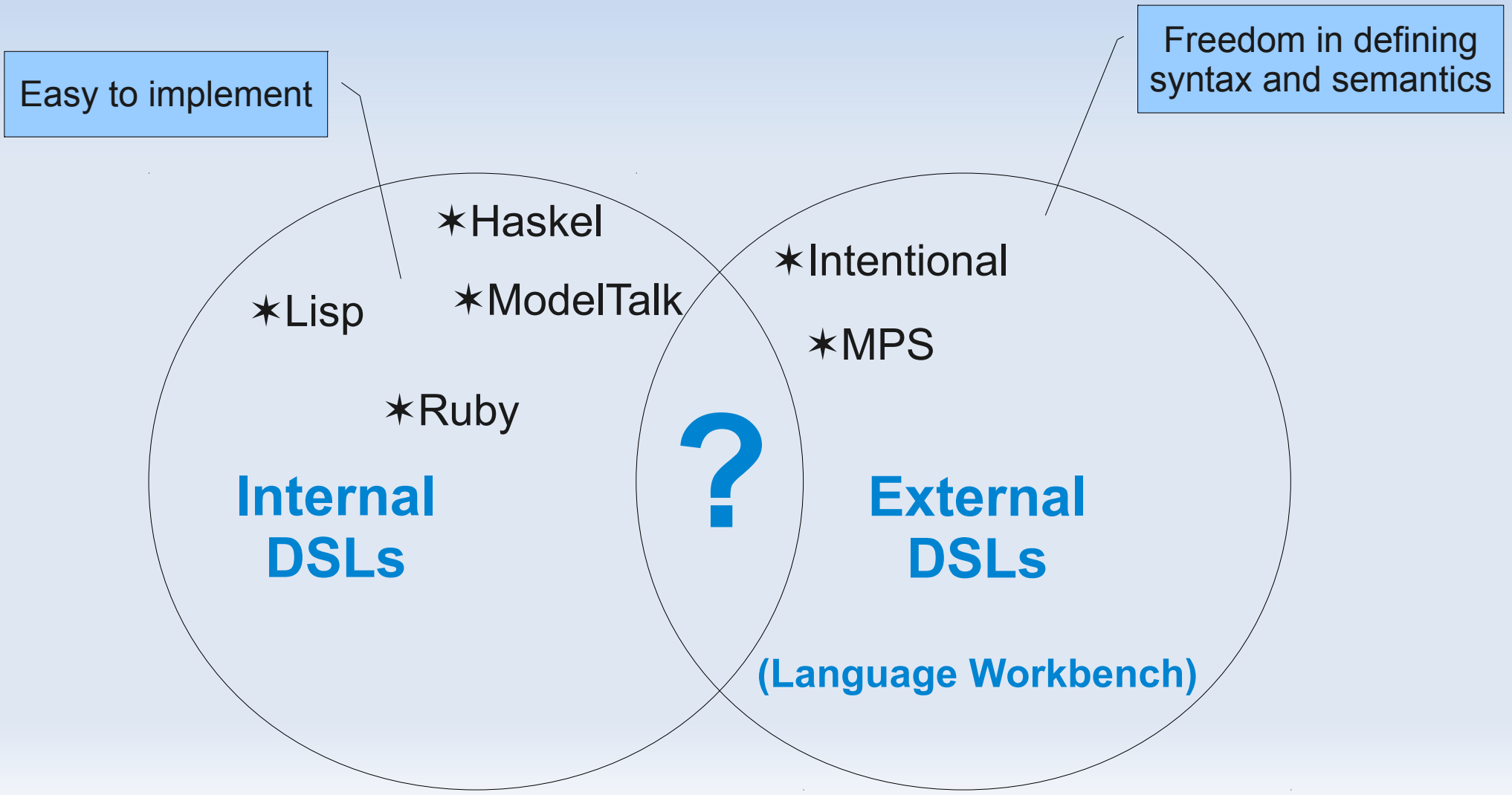
LOP: Middle-Out Development



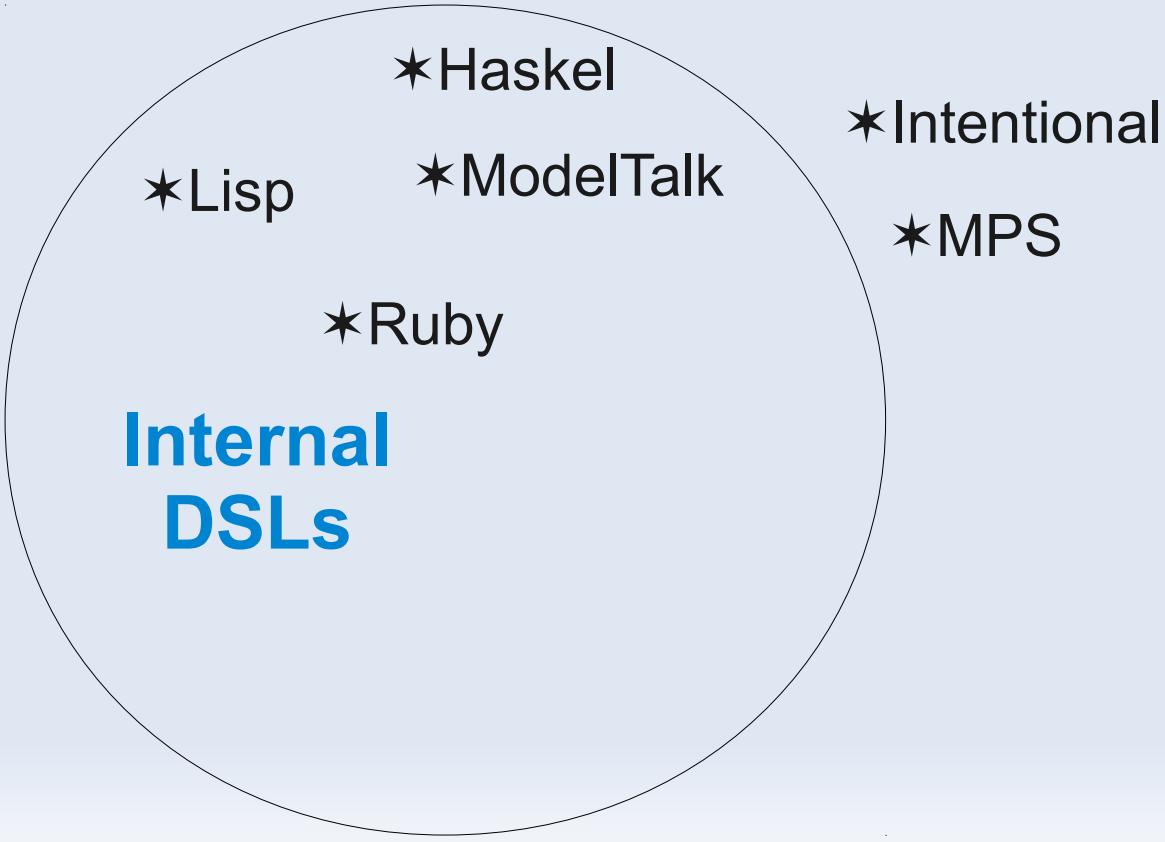
LOP: Middle-Out Development



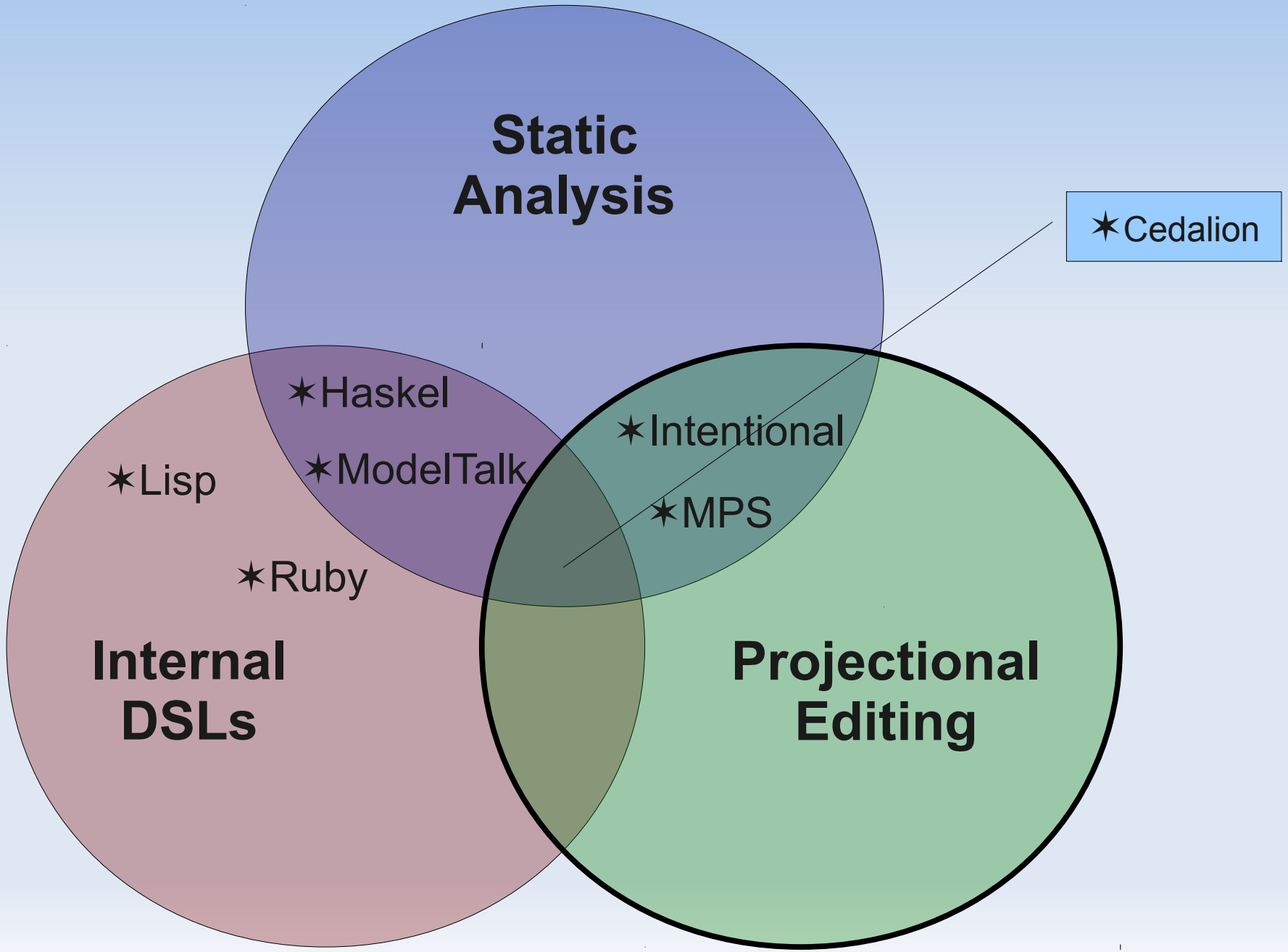
LOP Platforms



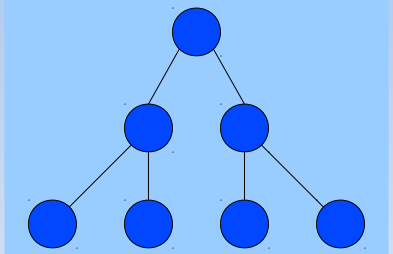
Language Design Perspective

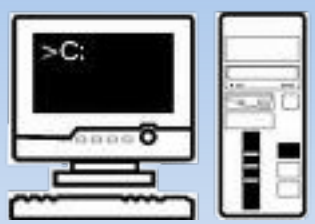


Language Design Perspective



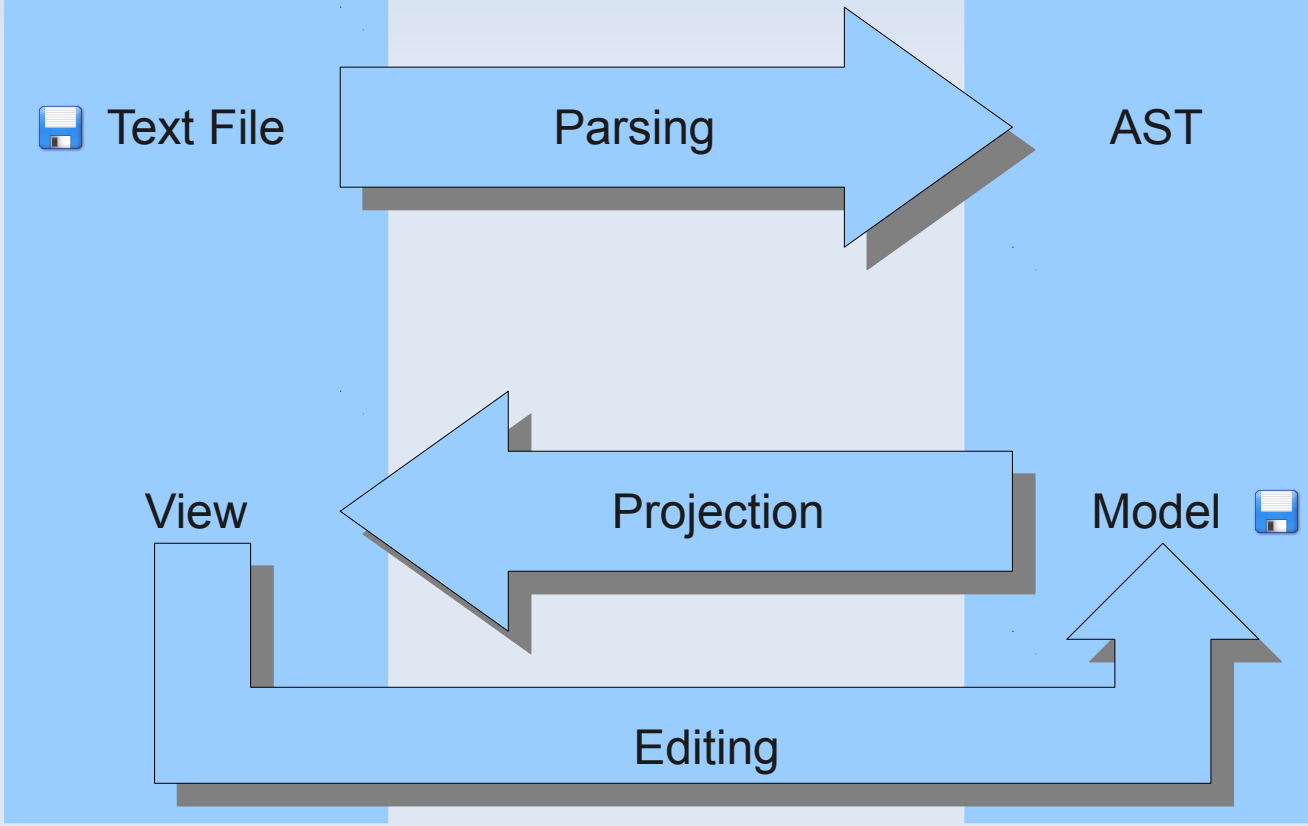
Projectional Editing



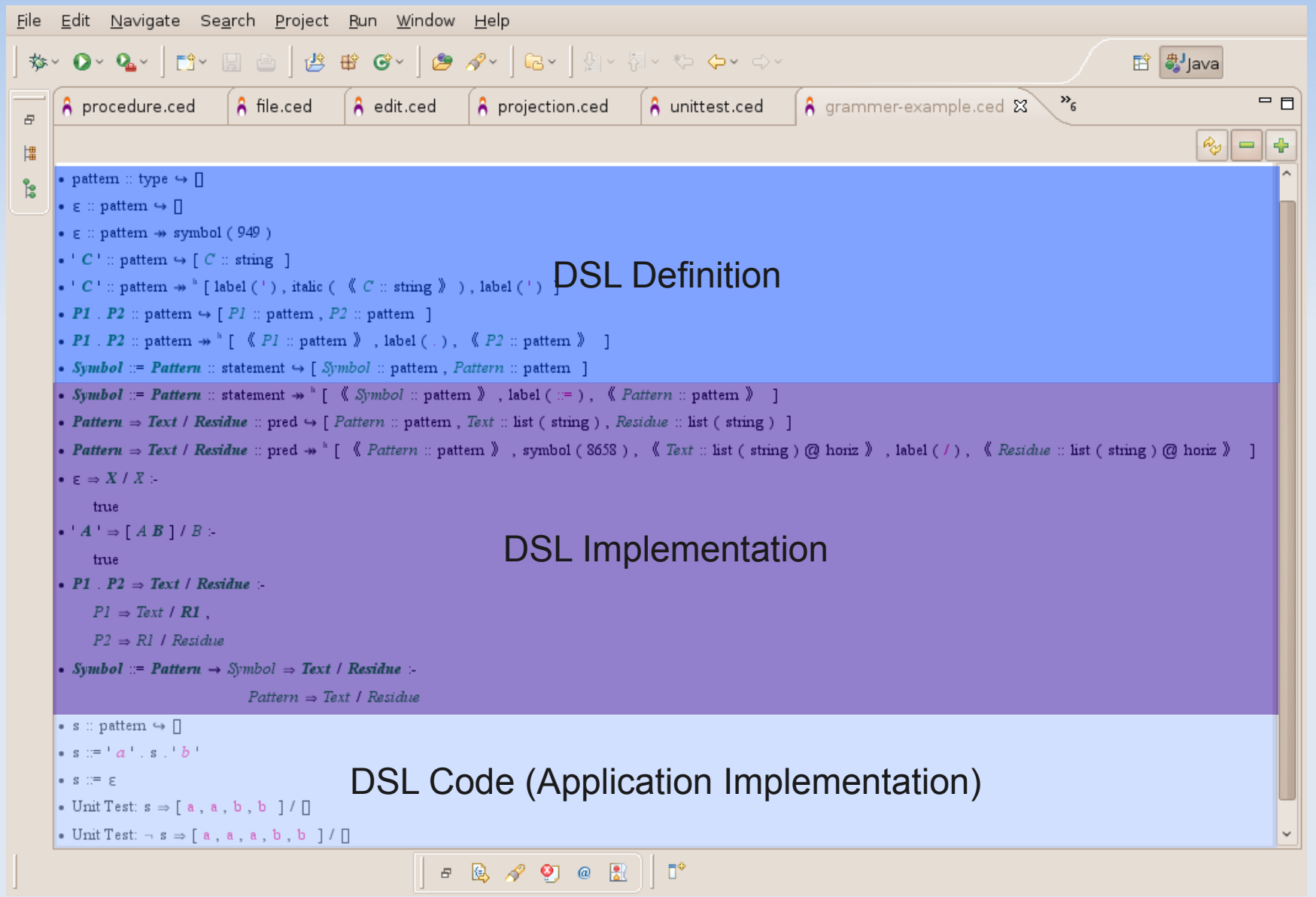


Traditional

Projectional Editing



A Peek at Cedalion



Outline

- Introduction
- **LOP Example**
- Designing an LOP Language
- Demo
- Conclusion

Example: LOP with Internal DSLs

- Task: Implement a parser for a given BNF grammar

  - $S ::= \mathbf{a}S\mathbf{b}$
  - $S ::= \varepsilon$

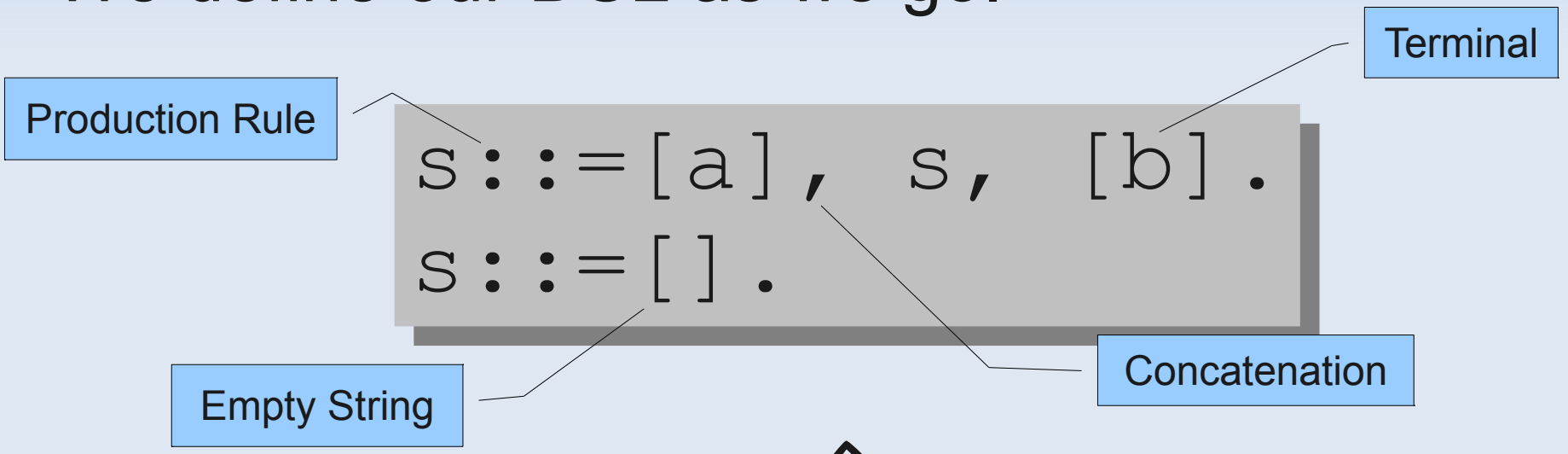
- Steps:

  1. Translating the specification into a “parsable” form in our target language, while implicitly defining the DSL.
  2. Making the code executable by implementing the DSL constructs.

- We use Prolog as a host language.

Step 1: Formalizing the Spec

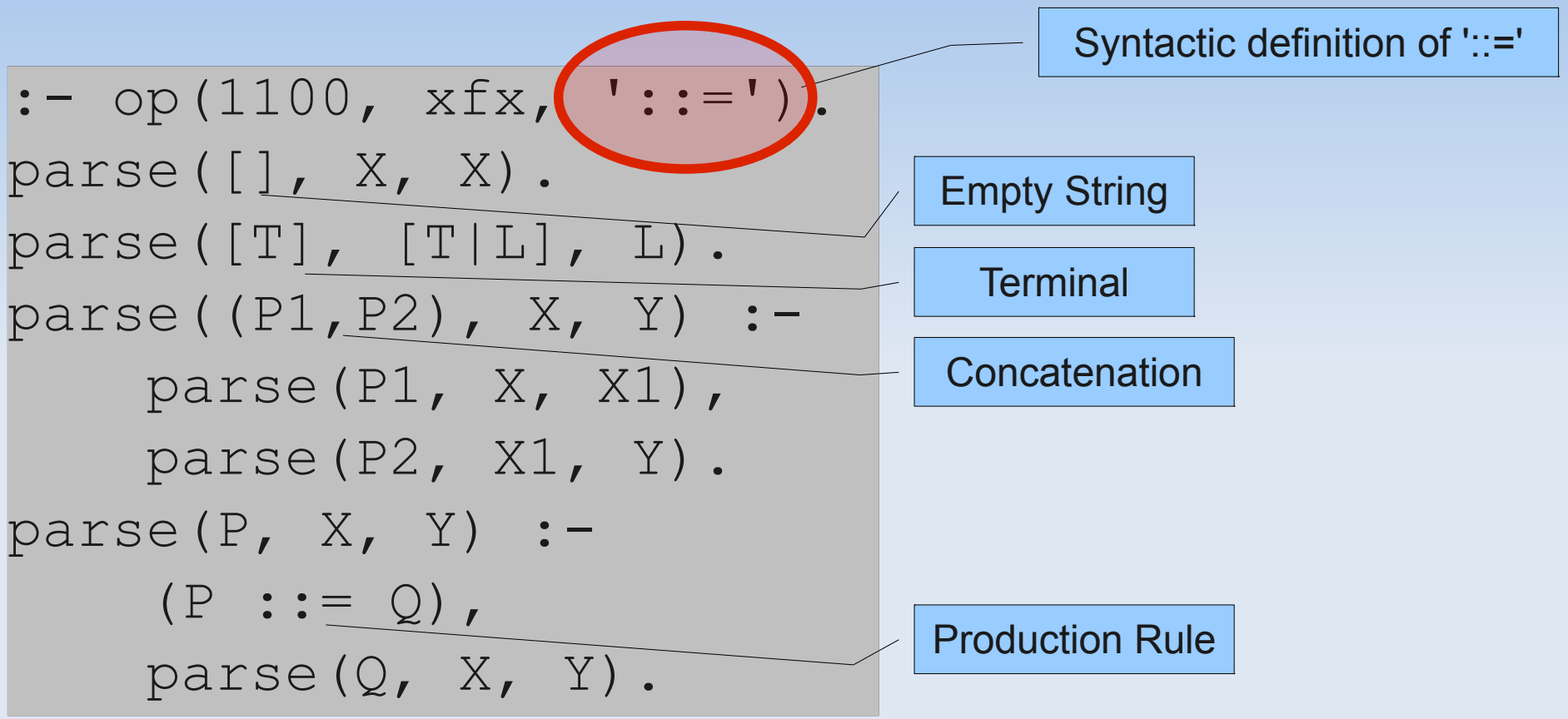
- We use Prolog's (meta) syntax to express the grammar (our specifications).
- We define our DSL as we go.



⇑

- $S ::= \mathbf{a}S\mathbf{b}$
- $S ::= \varepsilon$

Step 2: Making It Executable

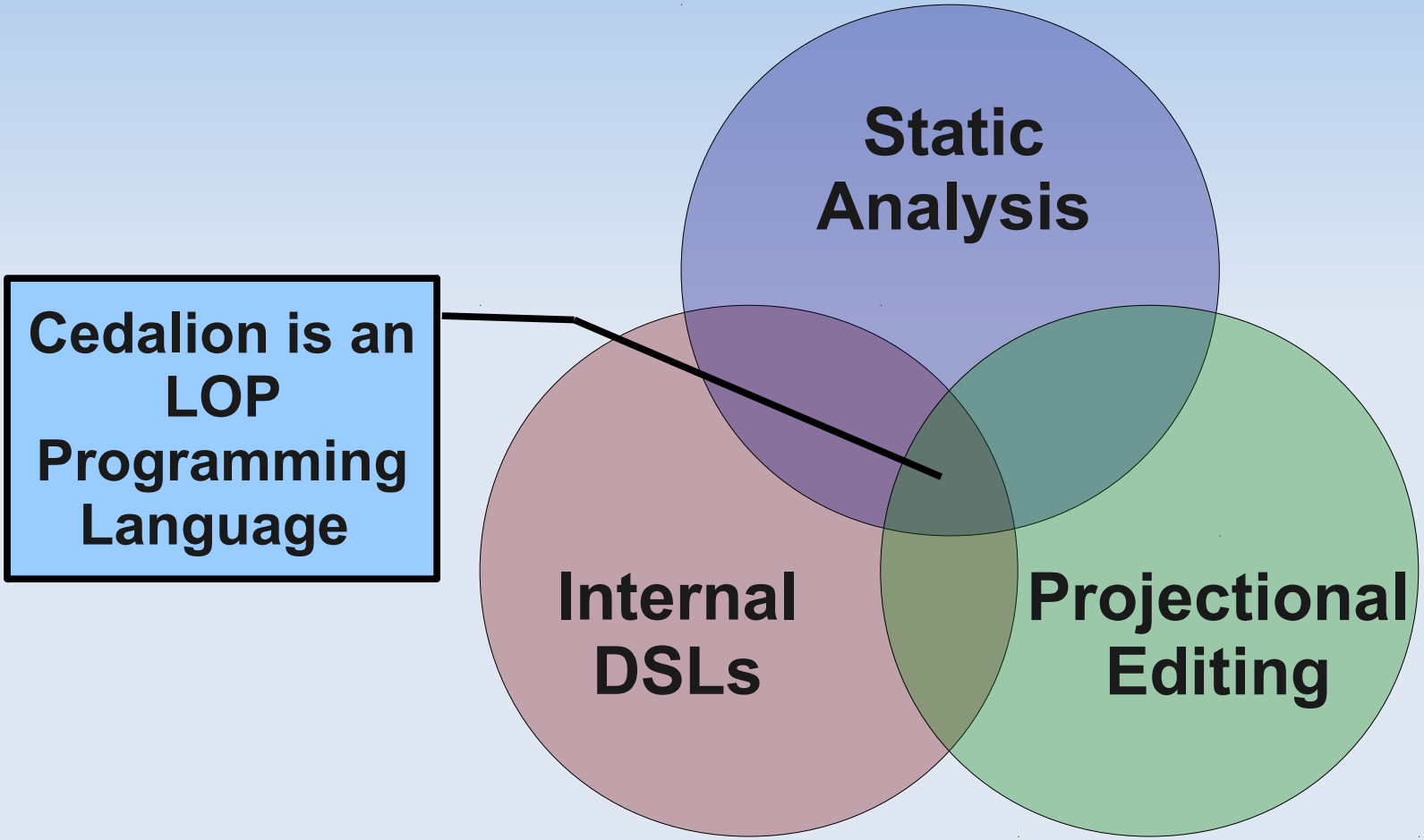


parse(P, T, R): succeeds if a prefix of *T* is derivable from *P*, leaving residue *R*.

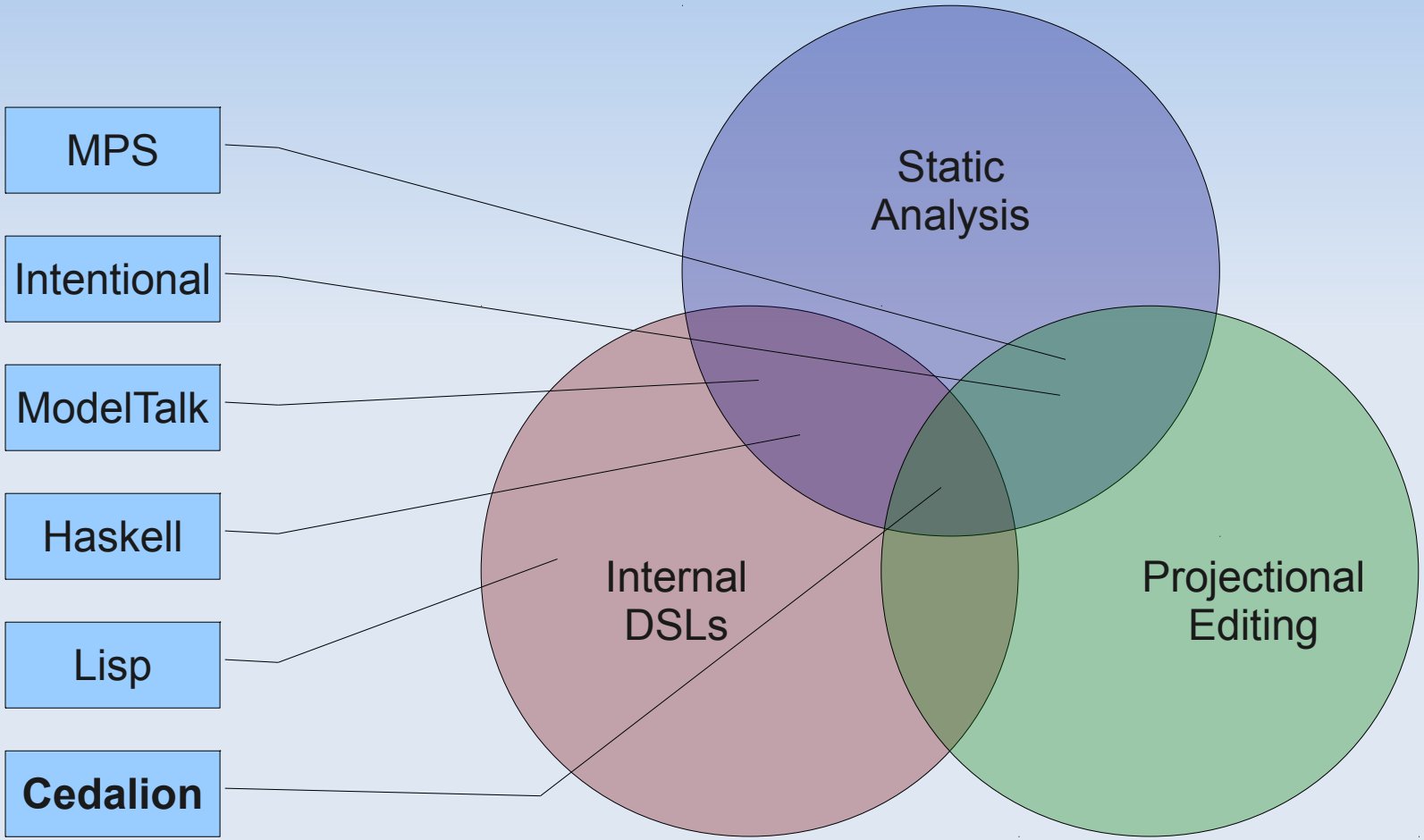
Limitations of Internal DSLs in Prolog

- Syntactic Limitations
  - When fusing several DSLs, operator collisions may occur.
- Semantic Limitations
  - No conformance checking.
  - Mistakenly writing s::={a},s,{b}
    instead of s::=[a],s,[b]
    will result in a bug, not an error.

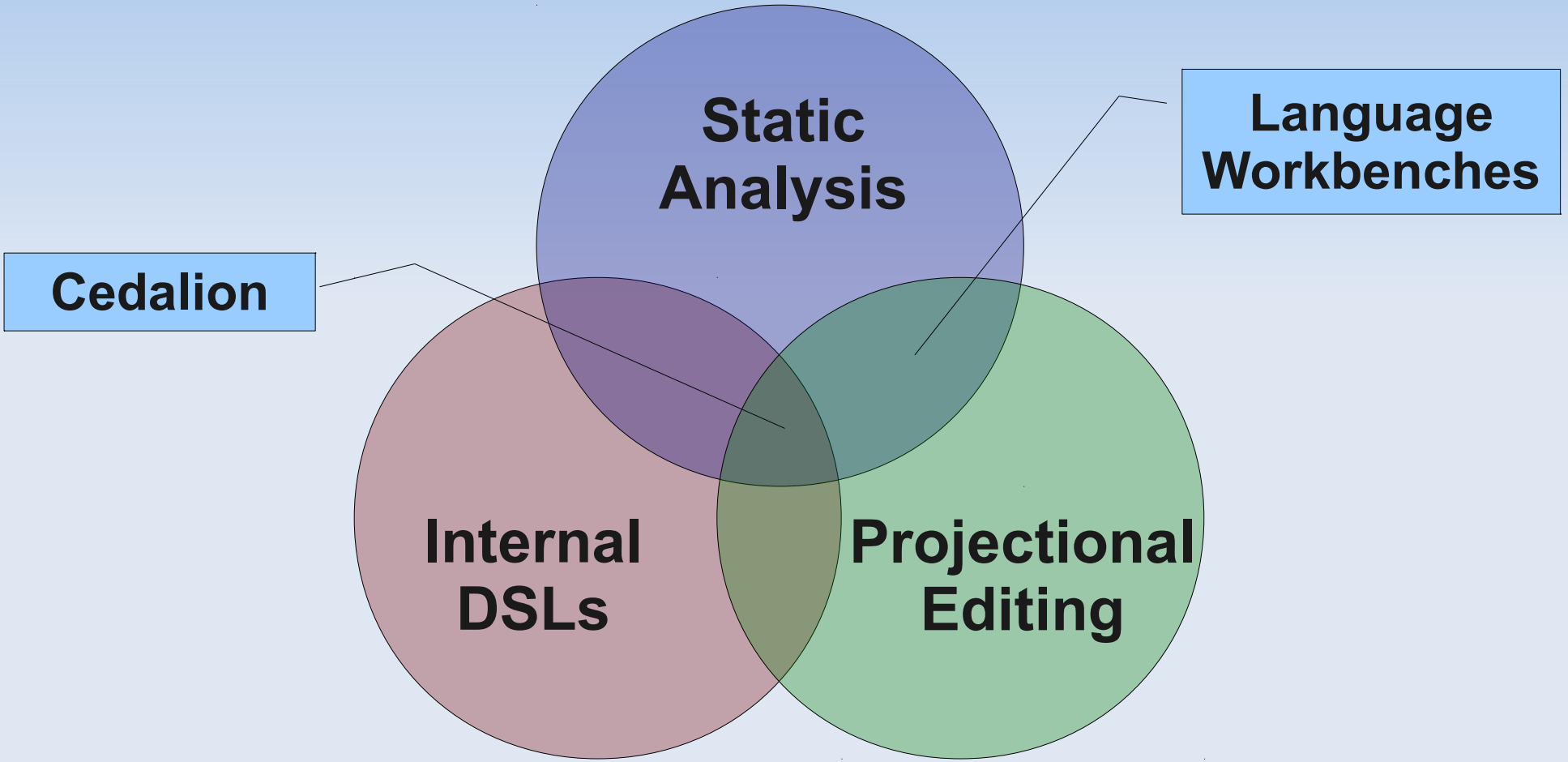
What is Cedalion



The Best of All Worlds

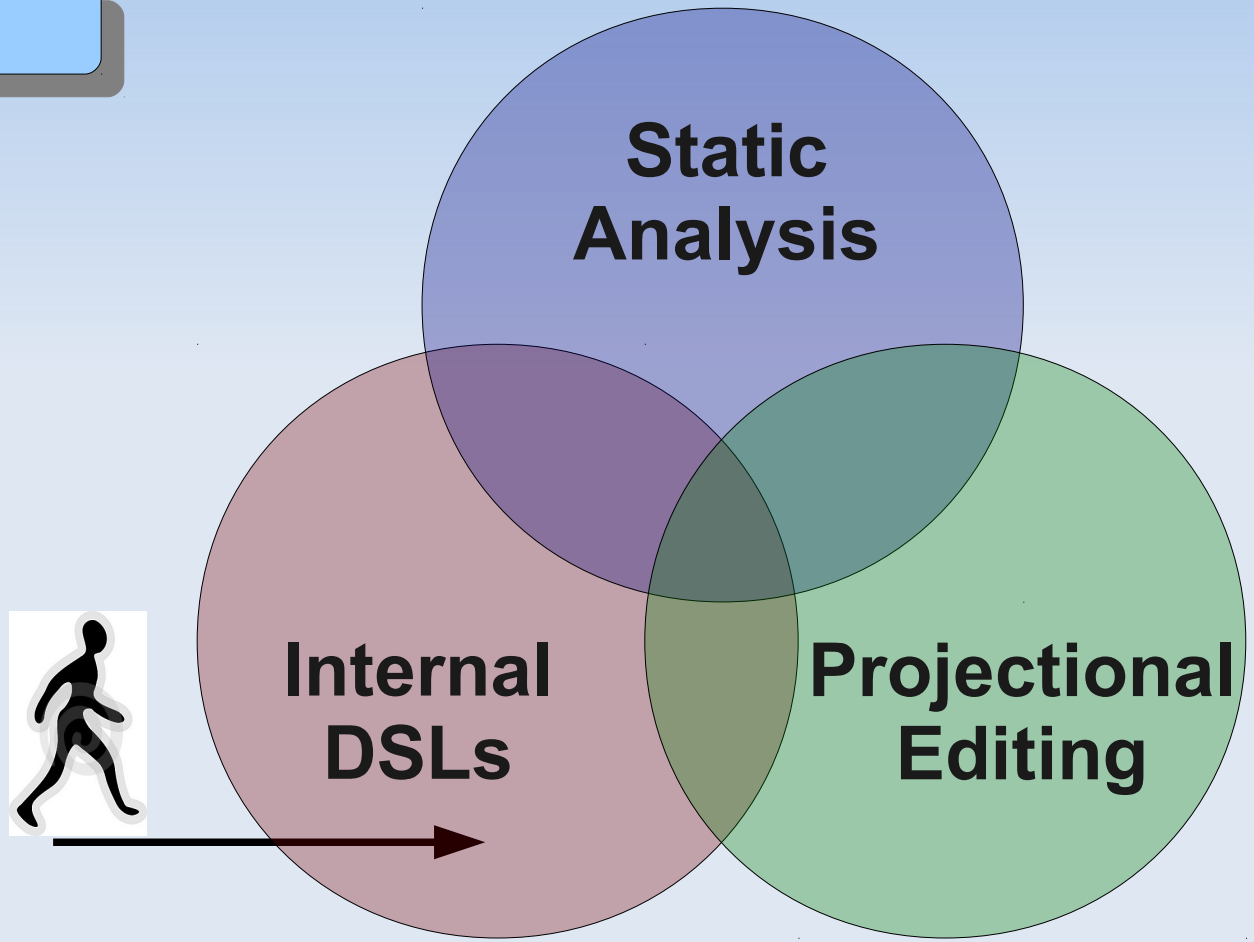


Our Approach



Designing an LOP Language

Extensible

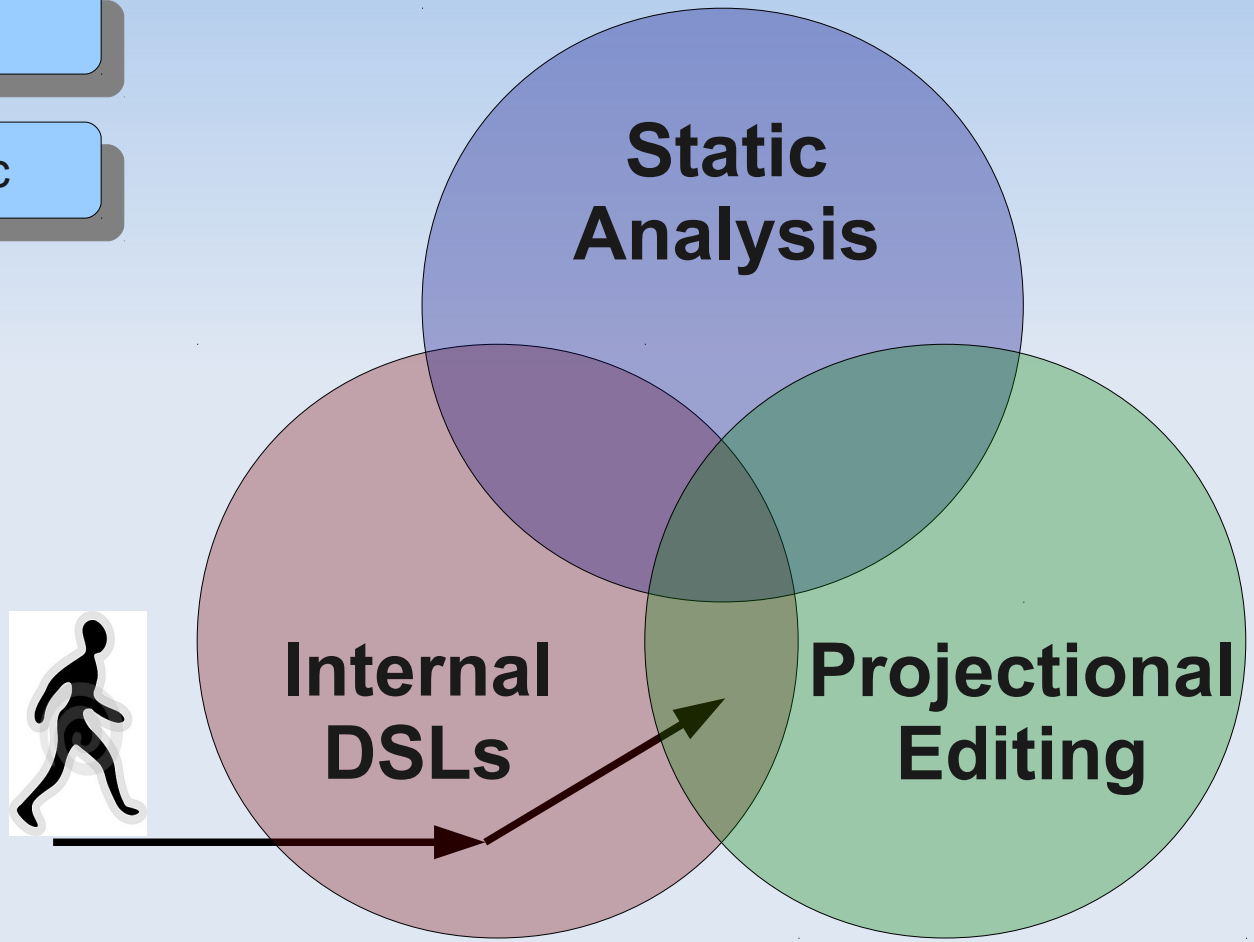


- ~~C/C++~~
- ~~C#~~
- Haskell
- ~~Java~~
- Lisp
- Nemerle
- Prolog
- Ruby
- Smalltalk

Designing an LOP Language

Extensible

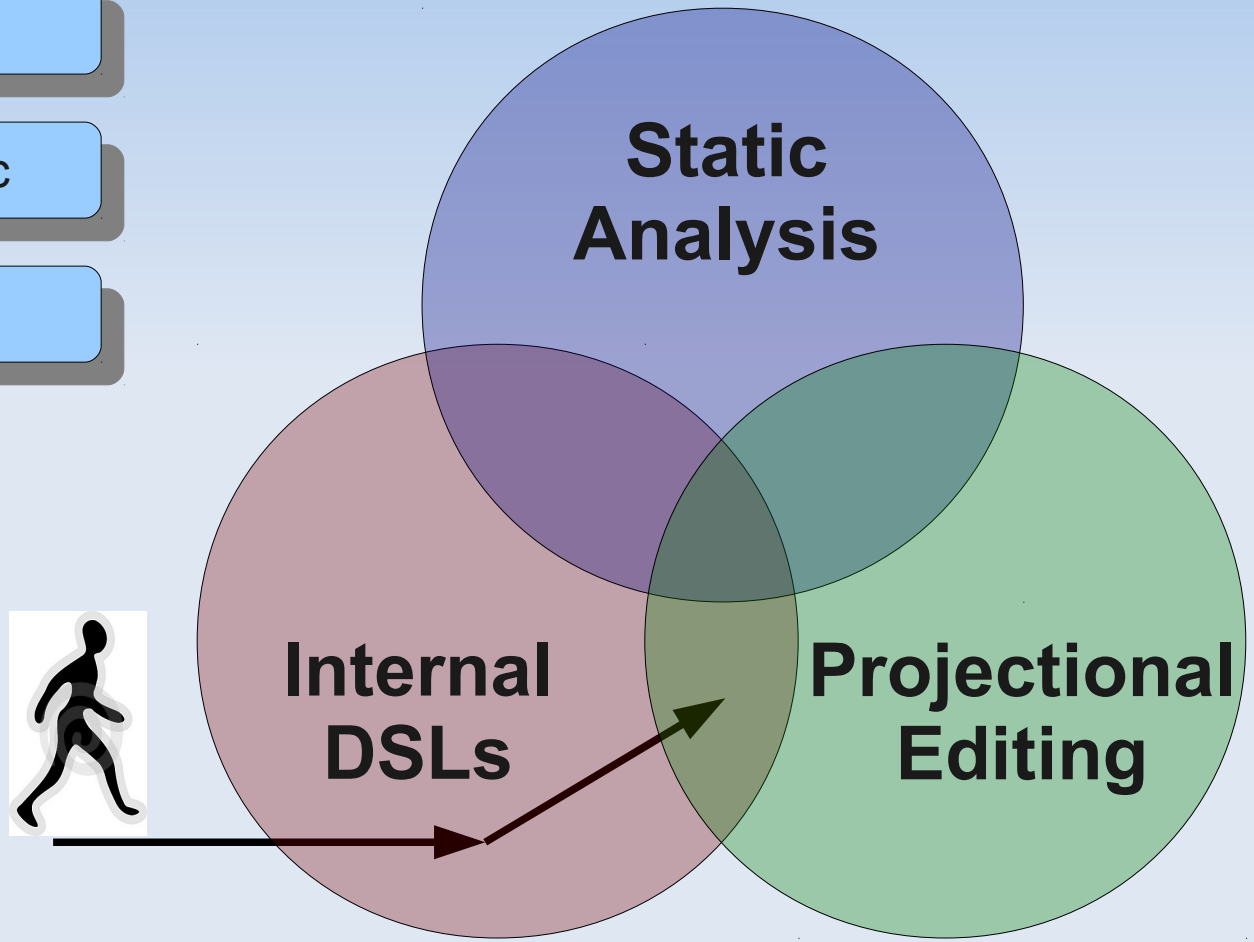
Homoiconic



- ~~C/C++~~
- ~~C#~~
- ~~Haskell~~
- ~~Java~~
- Lisp
- Nemerle
- Prolog
- ~~Ruby~~
- ~~Smalltalk~~

Designing an LOP Language

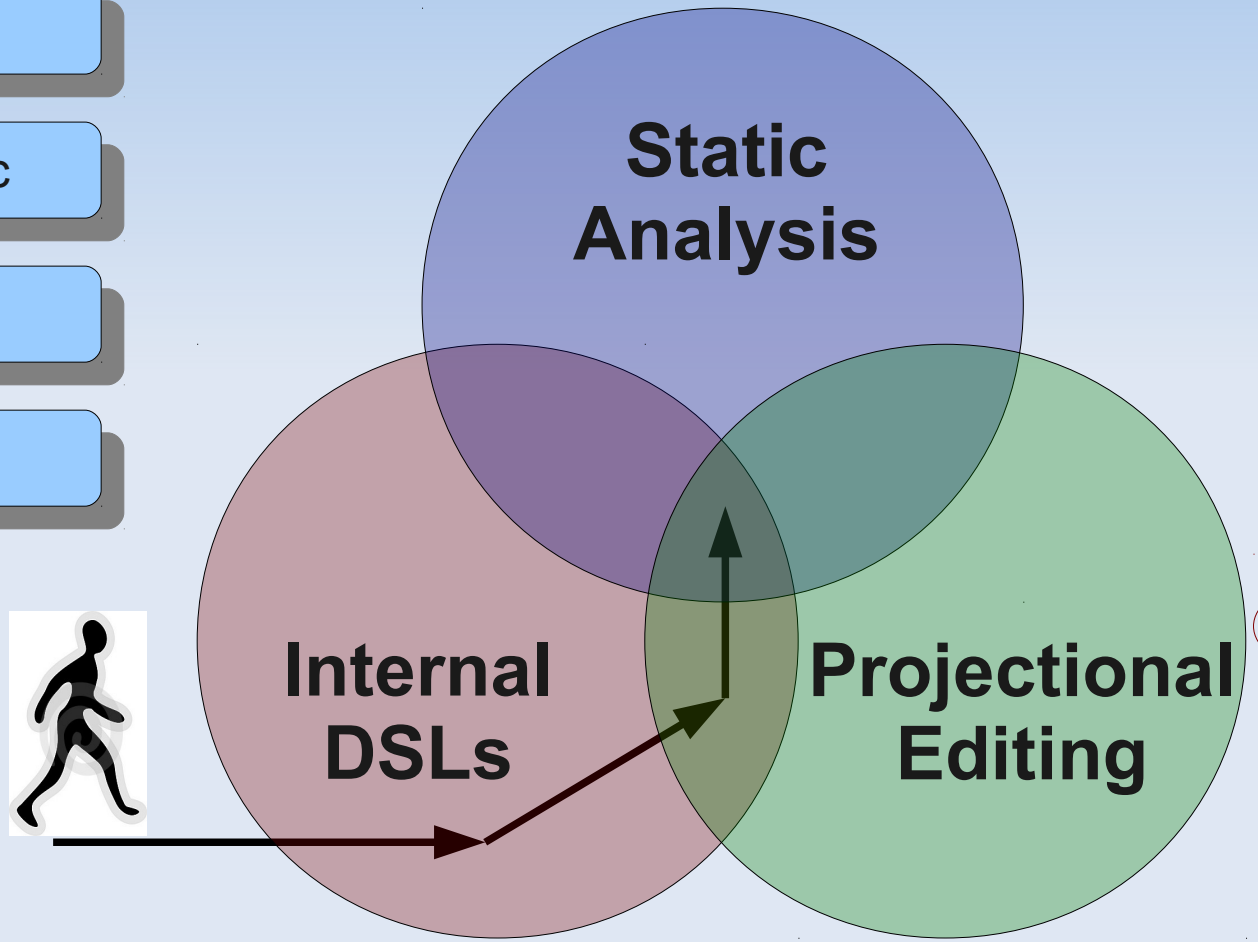
- Extensible
- Homoiconic
- Dynamic



- ~~C/C++~~
- ~~C#~~
- ~~Haskell~~
- ~~Java~~
- Lisp
- ~~Nemerle~~
- Prolog
- ~~Ruby~~
- ~~Smalltalk~~

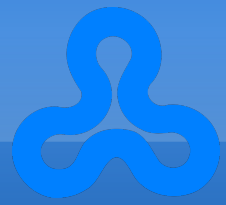
Designing an LOP Language

- Extensible
- Homoiconic
- Dynamic
- Validators



- ~~C/C++~~
- ~~C#~~
- ~~Haskell~~
- ~~Java~~
- Lisp
- ~~Nemerle~~
- Prolog
- ~~Ruby~~
- ~~Smalltalk~~

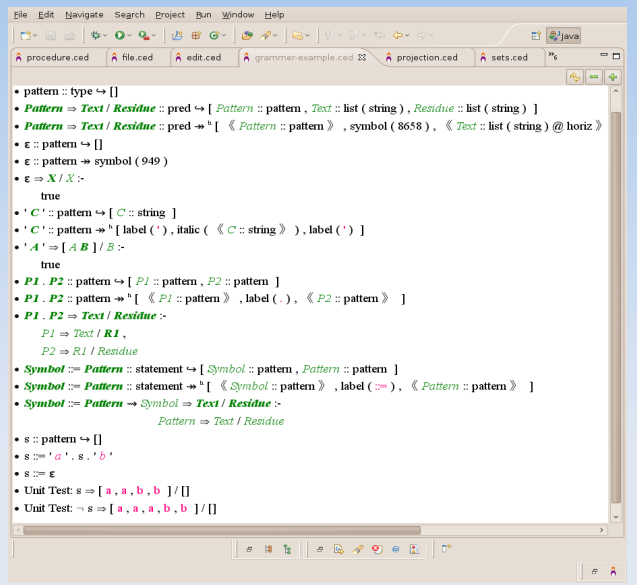
Cedalion



- An LOP language
  - Extensible, homoiconic, and dynamic
  - Statically typed, using Hindley-Milner type inference, implemented using validators.
  - Uses projectional editing (cannot be edited as text)
- Based on logic programming
  - As its “Domain 0”
- Comes with a Language Workbench
  - Provides pojectional editing for Cedalion

The Cedalion Workbench

- Based on Eclipse (3.5).
- Queries the Cedalion program for information on how to project language constructs.



Outline

- Introduction
- LOP Example
- Designing an LOP Language
- **Demo**
- Conclusion

Related Work

- Meta Programming System
  - A language workbench for external DSLs.
  - Developed by S. Dmitriev at Jetbrains, released 2009.

- Intentional Domain Workbench
  - A language workbench for external DSLs.
  - Developed by C. Simonyi at Intentional Software, not released yet.

- ModelTalk/Ink
  - A language workbench for XML-based internal DSLs. Used internally at Pontis.
  - [A. Hen-Tov, D. H. Lorenz and L. Shachter, 2009].

Conclusion

- Cedalion is a proof of concept
  - Provides the ease of defining internal DSLs.
  - Provides full control in defining syntax and semantics.
- Future Work
  - Making Cedalion ready for “prime time”.
  - Case studies: Solve real-life problems with Cedalion.
  - Evaluating other base semantics.

Thank You

- Cedalion
  - http://aop.cslab.openu.ac.il/research/cedalion/
- Boaz Rosenan
  - boaz@nansore.net
  - http://wiki-openu.openu.ac.il/research/index.php/Boaz_Rosenan
- David H. Lorenz
  - lorenz@openu.ac.il
  - http://www.openu.ac.il/home/lorenz/