CS 220: Survey of Programming Languages

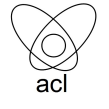
Lecture Slides

Declarative Programming Languages

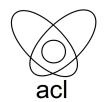
Jan Michael C. Yap

Algorithms and Complexity Laboratory
Department of Computer Science
University of the Philippines, Diliman
jcyap@dcs.upd.edu.ph

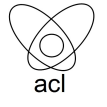
Session 7



Declarative Programming Languages



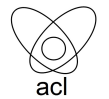
Declarative Programming Languages



Procedural vs. declarative programming

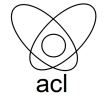
Procedural vs. declarative programming

- The first PL paradigms discussed provides a set of instructions for a system to perform towards the solution to a problem



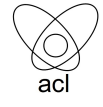
Procedural vs. declarative programming

- The first PL paradigms discussed provides a set of instructions for a system to perform towards the solution to a problem
  - As such, they are categorized as procedural languages



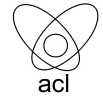
Procedural vs. declarative programming

- The first PL paradigms discussed provides a set of instructions for a system to perform towards the solution to a problem
  - As such, they are categorized as procedural languages
  - "[P]rograms are organized around control structures such as iteration and procedure invocation."



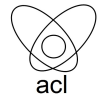
Procedural vs. declarative programming

- The first PL paradigms discussed provides a set of instructions for a system to perform towards the solution to a problem
  - As such, they are categorized as procedural languages
  - "[P]rograms are organized around control structures such as iteration and procedure invocation."
- An alternative way to program is to instead provide a set of goals and rules for the computers to follow towards the achievement of those goals



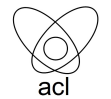
Procedural vs. declarative programming

- The first PL paradigms discussed provides a set of instructions for a system to perform towards the solution to a problem
  - As such, they are categorized as procedural languages
  - "[P]rograms are organized around control structures such as iteration and procedure invocation."
- An alternative way to program is to instead provide a set of goals and rules for the computers to follow towards the achievement of those goals
  - PLs that do this are called declarative languages



Procedural vs. declarative programming

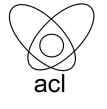
- The first PL paradigms discussed provides a set of instructions for a system to perform towards the solution to a problem
  - As such, they are categorized as procedural languages
  - "[P]rograms are organized around control structures such as iteration and procedure invocation."
- An alternative way to program is to instead provide a set of goals and rules for the computers to follow towards the achievement of those goals
  - PLs that do this are called declarative languages
  - Though rules were formulated by the programmer, the language's system "do[es] not explicitly invoke those rules in order to achieve the goals"



Characteristics of Declarative PL

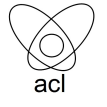
Characteristics of Declarative PL

- Programming involves the creation of propositions



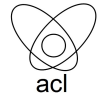
Characteristics of Declarative PL

- Programming involves the creation of propositions
  - Propositions can be atomic or compound



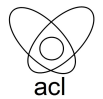
Characteristics of Declarative PL

- Programming involves the creation of propositions
  - Propositions can be atomic or compound
  - A grammatical standard usually imposed on propositions is that they should be Horn clauses



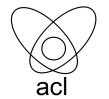
Characteristics of Declarative PL

- Programming involves the creation of propositions
  - Propositions can be atomic or compound
  - A grammatical standard usually imposed on propositions is that they should be Horn clauses
- Mechanisms of Declarative PLs are governed by formal (symbolic) logic



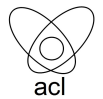
Characteristics of Declarative PL

- Programming involves the creation of propositions
  - Propositions can be atomic or compound
  - A grammatical standard usually imposed on propositions is that they should be Horn clauses
- Mechanisms of Declarative PLs are governed by formal (symbolic) logic
- Resolution allows inferred propositions to be computed from given propositions
  - Useful variables towards resolution are determined via unification

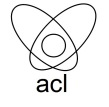


Characteristics of Declarative PL

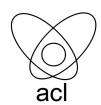
- Programming involves the creation of propositions
  - Propositions can be atomic or compound
  - A grammatical standard usually imposed on propositions is that they should be Horn clauses
- Mechanisms of Declarative PLs are governed by formal (symbolic) logic
- Resolution allows inferred propositions to be computed from given propositions
  - Useful variables towards resolution are determined via unification
  - Detecting inconsistencies in any propositions (refutation complete) is an important property of resolution



Declarative Programming Languages



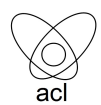
Background



RW Sebesta, Chapters 2.13 and 16.5

Background

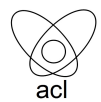
- Stands for Programming Logic



RW Sebesta, Chapters 2.13 and 16.5

Background

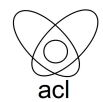
- Stands for Programming Logic
- Designed by Alain Colmerauer and Phillippe Roussel in the Artificial Intelligence Group at the University of Aix-Marseille, and Robert Kowalski of the Department of Artificial Intelligence at the University of Edinburgh



RW Sebesta, Chapters 2.13 and 16.5

Background

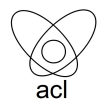
- Stands for Programming Logic
- Designed by Alain Colmerauer and Phillippe Roussel in the Artificial Intelligence Group at the University of Aix-Marseille, and Robert Kowalski of the Department of Artificial Intelligence at the University of Edinburgh
  - Prolog was meant as an automated theorem prover



RW Sebesta, Chapters 2.13 and 16.5

Background

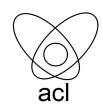
- Stands for Programming Logic
- Designed by Alain Colmerauer and Phillippe Roussel in the Artificial Intelligence Group at the University of Aix-Marseille, and Robert Kowalski of the Department of Artificial Intelligence at the University of Edinburgh
  - Prolog was meant as an automated theorem prover
  - Design was influenced by natural language processing



RW Sebesta, Chapters 2.13 and 16.5

Background

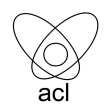
- Stands for Programming Logic
- Designed by Alain Colmerauer and Phillippe Roussel in the Artificial Intelligence Group at the University of Aix-Marseille, and Robert Kowalski of the Department of Artificial Intelligence at the University of Edinburgh
  - Prolog was meant as an automated theorem prover
  - Design was influenced by natural language processing
- "The primary components of Prolog are a method for specifying predicate calculus propositions and an implementation of a restricted form of resolution."



RW Sebesta, Chapters 2.13 and 16.5

Background

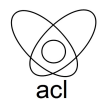
- Stands for Programming Logic
- Designed by Alain Colmerauer and Phillippe Roussel in the Artificial Intelligence Group at the University of Aix-Marseille, and Robert Kowalski of the Department of Artificial Intelligence at the University of Edinburgh
  - Prolog was meant as an automated theorem prover
  - Design was influenced by natural language processing
- "The primary components of Prolog are a method for specifying predicate calculus propositions and an implementation of a restricted form of resolution."
- "The first Prolog interpreter was developed at Marseille in 1972".



RW Sebesta, Chapters 2.13 and 16.5

Evaluation

- Data types[2,3]



[2] http://www.cs.bham.ac.uk/~pjh/prolog_module/md1/md1_data_types.html

[3] http://pages.cpsc.ucalgary.ca/~denzinge/courses/449-winter2004/slides/17-TypesControl-handout.pdf

Evaluation

- Data types[2,3]
  - Integer, floating point real numbers, character and string as supported primitive data types, but the actual (and sole) data type supported is term

[2] http://www.cs.bham.ac.uk/~pjh/prolog_module/md1/md1_data_types.html
[3] http://pages.cpsc.ucalgary.ca/~denzinge/courses/449-winter2004/slides/17-TypesControl-handout.pdf

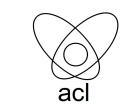
Evaluation

- Data types[2,3]
  - Integer, floating point real numbers, character and string as supported primitive data types, but the actual (and sole) data type supported is term
  - Lists, and in some variants bags and lists, as derived types

[2] http://www.cs.bham.ac.uk/~pjh/prolog_module/md1/md1_data_types.html
[3] http://pages.cpsc.ucalgary.ca/~denzinge/courses/449-winter2004/slides/17-TypesControl-handout.pdf

Evaluation

- Syntax design:



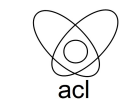
[4] https://github.com/simonkrenger/ch.bfh.bti7064.w2013.PrologParser/blob/master/doc/bnf-grammar.txt

[5] http://cseweb.ucsd.edu/classes/fa09/cse130/misc/prolog/prolog_tutorial.pdf

[6] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html

Evaluation

- Syntax design:
  - Production rules: 19 [4,5]
  - Keywords: No keywords for "pure" Prolog; a variant (SWI Prolog) has 12[6]



[4] https://github.com/simonkrenger/ch.bfh.bti7064.w2013.PrologParser/blob/master/doc/bnf-grammar.txt

[5] http://cseweb.ucsd.edu/classes/fa09/cse130/misc/prolog/prolog_tutorial.pdf

[6] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html

Evaluation

- Syntax design:
  - Production rules: 19 [4,5]
  - Keywords: No keywords for "pure" Prolog; a variant (SWI Prolog) has 12[6]
  - Very bare to allow for creation of facts and rules as freely as possible

[4] https://github.com/simonkrenger/ch.bfh.bti7064.w2013.PrologParser/blob/master/doc/bnf-grammar.txt

[5] http://cseweb.ucsd.edu/classes/fa09/cse130/misc/prolog/prolog_tutorial.pdf

[6] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html

Evaluation

- Syntax design:
  - Production rules: 19 [4,5]
  - Keywords: No keywords for "pure" Prolog; a variant (SWI Prolog) has 12[6]
  - Very bare to allow for creation of facts and rules as freely as possible
  - Everything in Prolog are either facts, rules, or assertions

[4] https://github.com/simonkrenger/ch.bfh.bti7064.w2013.PrologParser/blob/master/doc/bnf-grammar.txt

[5] http://cseweb.ucsd.edu/classes/fa09/cse130/misc/prolog/prolog_tutorial.pdf

[6] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html

Evaluation

- Syntax design:
  - Production rules: 19 [4,5]
  - Keywords: No keywords for "pure" Prolog; a variant (SWI Prolog) has 12[6]
  - Very bare to allow for creation of facts and rules as freely as possible
  - Everything in Prolog are either facts, rules, or assertions
  - Since the underlying mechanics of achieving the goals are not explicitly controlled by the programmer, several overlaps in functionalities are expected

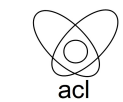
[4] https://github.com/simonkrenger/ch.bfh.bti7064.w2013.PrologParser/blob/master/doc/bnf-grammar.txt

[5] http://cseweb.ucsd.edu/classes/fa09/cse130/misc/prolog/prolog_tutorial.pdf

[6] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html

Evaluation

- Abstraction:



[7] http://faculty.simpson.edu/lydia.sinapova/www/cmsc310/Prolog/PrologLessons/LS01Rules.htm

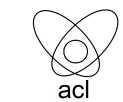
[8] http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse52

Evaluation

- Abstraction:
  - Variables are only "visible" to the clause they are used in[7]

[7] http://faculty.simpson.edu/lydia.sinapova/www/cmsc310/Prolog/PrologLessons/LS01Rules.htm

[8] http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse52



Evaluation

- Abstraction:
  - Variables are only "visible" to the clause they are used in[7]
  - Rules and predicates can be declared in separate files and have them "accessible" to each other via `module` declarations[8]

[7] http://faculty.simpson.edu/lydia.sinapova/www/cmsc310/Prolog/PrologLessons/LS01Rules.htm

[8] http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse52

Evaluation

- Abstraction:
  - Variables are only "visible" to the clause they are used in[7]
  - Rules and predicates can be declared in separate files and have them "accessible" to each other via `module` declarations[8]
    - The `use_module` predicate allows for importing modules

[7] http://faculty.simpson.edu/lydia.sinapova/www/cmsc310/Prolog/PrologLessons/LS01Rules.htm

[8] http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse52

Evaluation

- Abstraction:
  - Variables are only "visible" to the clause they are used in[7]
  - Rules and predicates can be declared in separate files and have them "accessible" to each other via `module` declarations[8]
    - The `use_module` predicate allows for importing modules
- Expressivity: Little to no add-on functionalities, but then again not really an issue in declarative programming

[7] http://faculty.simpson.edu/lydia.sinapova/www/cmsc310/Prolog/PrologLessons/L801Rules.htm

[8] http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse52

Evaluation

- Exception handling: SWI-Prolog has `try` and `catch` predicates to perform exception handling[9]

[9] http://www.swi-prolog.org/pldoc/man?section=exception
[10] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html
[11] https://www.info.ucl.ac.be/~pvr/Peter.thesis/Peter.thesis.html
[12] https://www.info.ucl.ac.be/~pvr/Taylor.thesis.pdf

Evaluation

- Exception handling: SWI-Prolog has `try` and `catch` predicates to perform exception handling[9]
- Restricted aliasing: No (explicit) aliasing being done

[9] http://www.swi-prolog.org/pldoc/man?section=exception
[10] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html
[11] https://www.info.ucl.ac.be/~pvr/Peter.thesis/Peter.thesis.html
[12] https://www.info.ucl.ac.be/~pvr/Taylor.thesis.pdf

Evaluation

- Exception handling: SWI-Prolog has `try` and `catch` predicates to perform exception handling[9]
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency

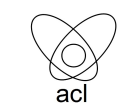
[9] http://www.swi-prolog.org/pldoc/man?section=exception
[10] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html
[11] https://www.info.ucl.ac.be/~pvr/Peter.thesis/Peter.thesis.html
[12] https://www.info.ucl.ac.be/~pvr/Taylor.thesis.pdf

Evaluation

- Exception handling: SWI-Prolog has `try` and `catch` predicates to perform exception handling[9]
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency
  - Original Prolog had no (need for) type checking

[9] http://www.swi-prolog.org/pldoc/man?section=exception
[10] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html
[11] https://www.info.ucl.ac.be/~pvr/Peter.thesis/Peter.thesis.html
[12] https://www.info.ucl.ac.be/~pvr/Taylor.thesis.pdf



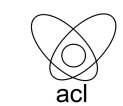


Evaluation

- Exception handling: SWI-Prolog has `try` and `catch` predicates to perform exception handling[9]
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency
  - Original Prolog had no (need for) type checking
  - Can be compiled or interpreted (some variants at least)[10]

[9] http://www.swi-prolog.org/pldoc/man?section=exception
[10] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html
[11] https://www.info.ucl.ac.be/~pvr/Peter.thesis/Peter.thesis.html
[12] https://www.info.ucl.ac.be/~pvr/Taylor.thesis.pdf



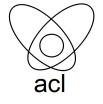


Evaluation

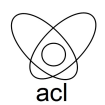
- Exception handling: SWI-Prolog has `try` and `catch` predicates to perform exception handling[9]
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency
  - Original Prolog had no (need for) type checking
  - Can be compiled or interpreted (some variants at least)[10]
  - Formal proofs exist proving that Prolog can be as fast as C[11,12]

[9] http://www.swi-prolog.org/pldoc/man?section=exception
[10] http://www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/ext-prolog.html
[11] https://www.info.ucl.ac.be/~pvr/Peter.thesis/Peter.thesis.html
[12] https://www.info.ucl.ac.be/~pvr/Taylor.thesis.pdf

Declarative Programming Languages



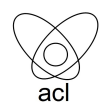
Background



http://self.gutenberg.org/articles/G%C3%B6del_(programming_language)

Background

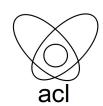
- Gödel was developed by Patricia Hill and John Lloyd in 1992



http://self.gutenberg.org/articles/G%C3%B6del_(programming_language)

Background

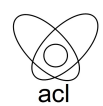
- Gödel was developed by Patricia Hill and John Lloyd in 1992
- Named after Kurt Gödel, but also is an acronym which stands for "God's Own Declarative Language"



http://self.gutenberg.org/articles/G%C3%B6del_(programming_language)

Background

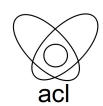
- Gödel was developed by Patricia Hill and John Lloyd in 1992
- Named after Kurt Gödel, but also is an acronym which stands for "God's Own Declarative Language"
- "It is based on, and supposed to be a successor of Prolog..."



http://self.gutenberg.org/articles/G%C3%B6del_(programming_language)

Background

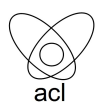
- Gödel was developed by Patricia Hill and John Lloyd in 1992
- Named after Kurt Gödel, but also is an acronym which stands for "God's Own Declarative Language"
- "It is based on, and supposed to be a successor of Prolog..."
- "The control component of the program is concerned with the construction and pruning of search trees. Typically, the computation rule (which selects a subformula in a goal for an extension step) is partially specified by control declarations and the pruning of a search tree is specified by pruning operator."



http://self.gutenberg.org/articles/G%C3%B6del_(programming_language)

Evaluation

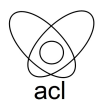
- Data types[14]



[14] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

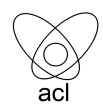
- Data types[14]
  - Integer, rational numbers, character and string as supported primitive data types



[14] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

- Data types[14]
  - Integer, rational numbers, character and string as supported primitive data types
  - Lists, sets, and some other (unspecified) abstract data types bags and lists, as derived types

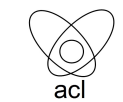


[14] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

- Syntax design:



[15] http://www.comp.leeds.ac.uk/hill/GOEDEL/examples.html
[16] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[17] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf

Evaluation

- Syntax design:
  - No grammar found, but looking at sample programs[15] and the basic features available[16], Prolog is relatively simpler than Gödel

[15] http://www.comp.leeds.ac.uk/hill/GOEDEL/examples.html
[16] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[17] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf

Evaluation

- Syntax design:
  - No grammar found, but looking at sample programs[15] and the basic features available[16], Prolog is relatively simpler than Gödel
  - Program symbols are either BASE, CONSTRUCTOR, CONSTANT, FUNCTION, PROPOSITION, or PREDICATE, and syntax for declaring such are consistently applied[17]

[15] http://www.comp.leeds.ac.uk/hill/GOEDEL/examples.html
[16] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[17] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf

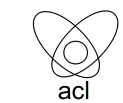
Evaluation

- Syntax design:
  - No grammar found, but looking at sample programs[15] and the basic features available[16], Prolog is relatively simpler than Gödel
  - Program symbols are either BASE, CONSTRUCTOR, CONSTANT, FUNCTION, PROPOSITION, or PREDICATE, and syntax for declaring such are consistently applied[17]
  - "Modularized" syntax (compared with Prolog) in that certain code sections must be declared to create the components (e.g.imported modules, predicates) needed for a program

[15] http://www.comp.leeds.ac.uk/hill/GOEDEL/examples.html
[16] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[17] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf

Evaluation

- Abstraction:[18]



[18] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf
[19] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

- Abstraction:[18]
  - "A Gödel source program is contained in one or more modules. A module declares all the symbols of the language of that module. A module may import other modules in order to avail itself of their symbols."

[18] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf
[19] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

- Abstraction:[18]
  - "A Gödel source program is contained in one or more modules. A module declares all the symbols of the language of that module. A module may import other modules in order to avail itself of their symbols."
  - "Gödel is polymorphic. A single FUNCTION or PREDICATE may be declared to accept different types as its arguments. "

[18] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf
[19] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

- Abstraction:[18]
  - "A Gödel source program is contained in one or more modules. A module declares all the symbols of the language of that module. A module may import other modules in order to avail itself of their symbols."
  - "Gödel is polymorphic. A single FUNCTION or PREDICATE may be declared to accept different types as its arguments. "
- Expressivity:[19]

[18] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf
[19] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

- Abstraction:[18]
  - "A Gödel source program is contained in one or more modules. A module declares all the symbols of the language of that module. A module may import other modules in order to avail itself of their symbols."
  - "Gödel is polymorphic. A single FUNCTION or PREDICATE may be declared to accept different types as its arguments. "
- Expressivity:[19]
  - "The Gödel language include modules that process numbers, lists, strings and sets, modules that provide input/output, and modules that provide meta-programming facilities."

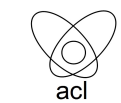
[18] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf
[19] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

- Abstraction:[18]
  - "A Gödel source program is contained in one or more modules. A module declares all the symbols of the language of that module. A module may import other modules in order to avail itself of their symbols."
  - "Gödel is polymorphic. A single FUNCTION or PREDICATE may be declared to accept different types as its arguments. "
- Expressivity:[19]
  - "The Gödel language include modules that process numbers, lists, strings and sets, modules that provide input/output, and modules that provide meta-programming facilities."
  - "Gödel also makes significant use of abstract data types, which are implemented by means of the module and type systems."

[18] http://web.cecs.pdx.edu/~antoy/homepage/theses/D_Shapiro_thesis.pdf
[19] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

Evaluation

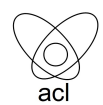


[20] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

[21] http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

Evaluation

- Exception handling: No (explicit) exception handling mechanisms



[20] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[21] http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

Evaluation

- Exception handling: No (explicit) exception handling mechanisms
- Restricted aliasing: No (explicit) aliasing being done

[20] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[21] http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

Evaluation

- Exception handling: No (explicit) exception handling mechanisms
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency[20,21]

[20] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language

[21] http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

Evaluation

- Exception handling: No (explicit) exception handling mechanisms
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency[20,21]
  - Employs both static and dynamic type-checking

[20] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[21] http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

Evaluation

- Exception handling: No (explicit) exception handling mechanisms
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency[20,21]
  - Employs both static and dynamic type-checking
  - Built on top of Prolog

[20] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[21] http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

Evaluation

- Exception handling: No (explicit) exception handling mechanisms
- Restricted aliasing: No (explicit) aliasing being done
- Efficiency[20,21]
  - Employs both static and dynamic type-checking
  - Built on top of Prolog
  - Pruning mechanism employed is based on the commit operation of concurrent programming language

[20] http://self.gutenberg.org/articles/G%C3%B6del_%28programming_language
[21] http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

END OF SESSION 7