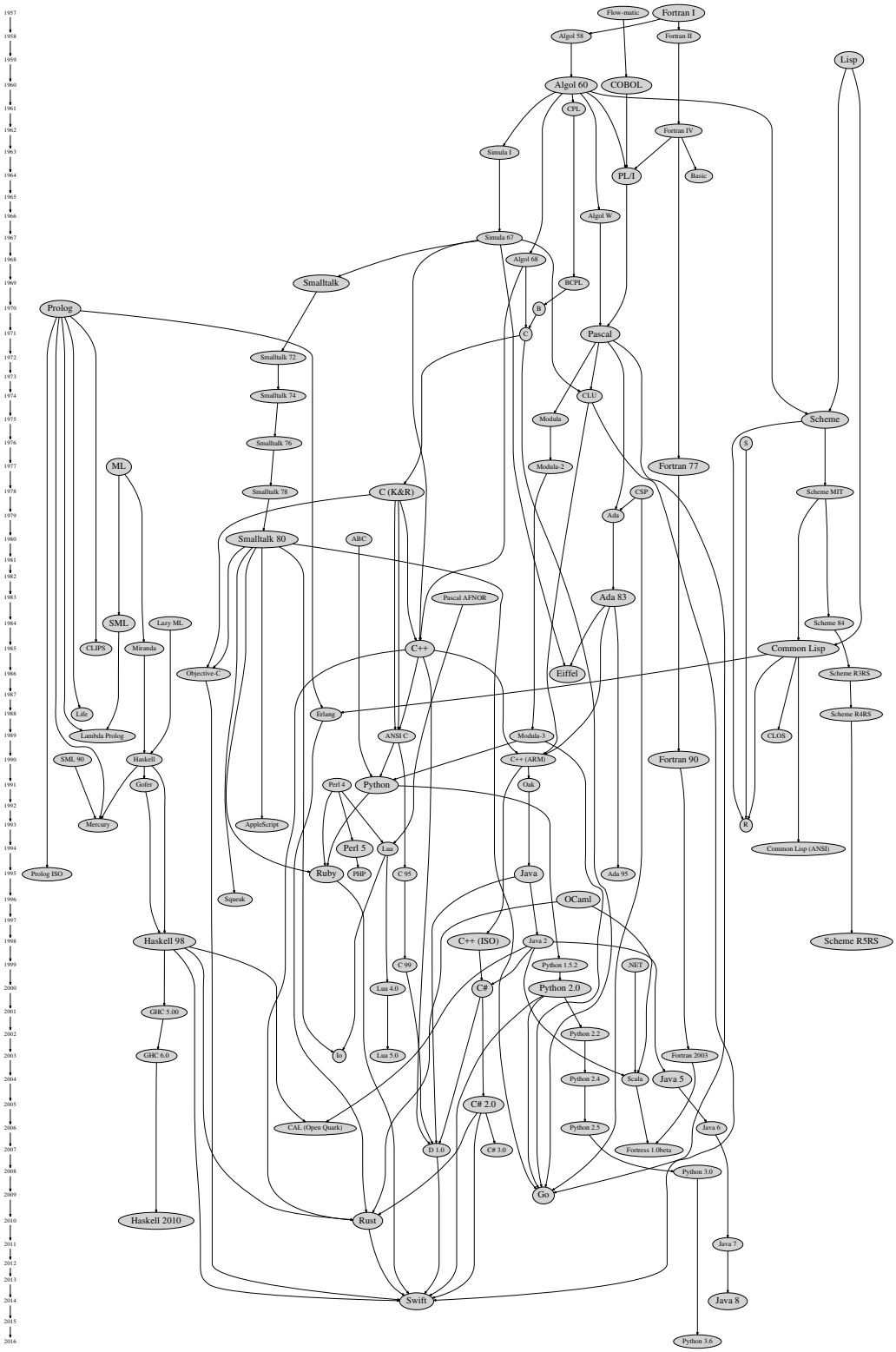
T_1–Parte A – Programação Lógica – 2017s2

Ciências da Computação – Universidade Federal de Santa Catarina



https://www.inf.ufsc.br/~alexandre.goncalves.silva/courses/17s2/ine5416/exercicios/t1A/diagrama-simplificado.pdf

- Considerando a hierarquia simplificada de linguagens [1] acima, escreva um **banco de dados** em Prolog, chamado de `linguagens.pl`, para a representação destes fatos, considerando a seguinte padronização:
  - Toda linguagem deve ter nome exatamente como está no diagrama, definida entre aspas simples. Por exemplo: `'Fortran I'`
  - Um fato sobre o ano (com 4 dígitos) de desenvolvimento da linguagem deve ser definido. O predicado deve ter a seguinte padronização: `linguagem(nome, ano).`
    Por exemplo: `linguagem('Fortran I', 1958).`
  - Um fato sobre a precedência imediata da linguagem deve ser definida. O predicado deve ter a seguinte padronização: `predecessora(nome, nome_pred).`
    Por exemplo: `predecessora('Fortran II', 'Fortran I').`

- Além disto, implemente um programa, chamado de `programa.pl`, com predicados e suas regras para responder às questões a seguir, inferirindo novos conhecimentos (não declarados nos fatos iniciais):

1. Linguagens `L` que possuem algum predecessor:

   `lingcompre(L) :-` _____

   Exemplo:

   ```
   ?- lingcompre('Fortran II').
   true.

   ?- lingcompre('Fortran I').
   false.
   ```

2. Linguagens `L` que possuem algum predecessor, e também são predecessoras de outras linguagens:

   `lingprecompre(L) :-` _____

   Exemplo:

   ```
   ?- lingprecompre('Fortran I').
   false.

   ?- lingprecompre('Fortran II').
   true.

   ?- lingprecompre('Basic').
   false.
   ```

3. Linguagens predecessoras `Lp` de linguagens desenvolvidas em um determinado ano `A`.

   `lingpreano(Lp, A) :-` _____

   Exemplo:

   ```
   ?- lingpreano(Lp, 1958).
   Lp = 'Fortran I'.
   ```

[1] Para uma versão completa, veja `http://rigaux.org/language-study/diagram.html`

- A relação de precedência entre linguagens pode ser acompanhada no diagrama inicial. De modo a facilitar a identificação, segue o ano de desenvolvimento de cada linguagem:
  - 1957: 'Flow-matic' , 'Fortran I'
  - 1958: 'Fortran II' , 'Algol 58'
  - 1959: 'Lisp'
  - 1960: 'COBOL' , 'Algol 60'
  - 1961: 'CPL'
  - 1962: 'Fortran IV'
  - 1963: 'Simula I'
  - 1964: 'PL/I' , 'Basic'
  - 1966: 'Algol W'
  - 1967: 'Simula 67'
  - 1968: 'Algol 68'
  - 1969: 'BCPL' , 'Smalltalk'
  - 1970: 'B' , 'Prolog'
  - 1971: 'Pascal' , 'C'
  - 1972: 'Smalltalk 72'
  - 1974: 'CLU' , 'Smalltalk 74'
  - 1975: 'Scheme' , 'Modula'
  - 1976: 'Smalltalk 76' , 'S'
  - 1977: 'Modula-2' , 'ML' , 'Fortran 77'
  - 1978: 'Smalltalk 78' , 'Scheme MIT' , 'CSP' , 'C (K&R)'
  - 1979: 'Ada'
  - 1980: 'ABC' , 'Smalltalk 80'
  - 1983: 'Ada 83' , 'Pascal AFNOR'
  - 1984: 'Lazy ML' , 'SML' , 'Scheme 84'
  - 1985: 'Common Lisp' , 'Miranda' , 'C++' , 'CLIPS'
  - 1986: 'Scheme R3RS' , 'Objective-C' , 'Eiffel'
  - 1988: 'Scheme R4RS' , 'Erlang' , 'Life'
  - 1989: 'CLOS' , 'Modula-3' , 'ANSI C' , 'Lambda Prolog'
  - 1990: 'C++ (ARM)' , 'Haskell' , 'SML 90' , 'Fortran 90'
  - 1991: 'Gofer' , 'Python' , 'Perl 4' , 'Oak'
  - 1993: 'Mercury' , 'AppleScript' , 'R'
  - 1994: 'Lua' , 'Common Lisp (ANSI)' , 'Perl 5'
  - 1995: 'C 95' , 'Prolog ISO' , 'Ada 95' , 'PHP' , 'Java' , 'Ruby'

- 1996: 'Squeak' , 'OCaml'
- 1998: 'Scheme R5RS' , 'C++ (ISO)' , 'Java 2' , 'Haskell 98'
- 1999: '.NET' , 'Python 1.5.2' , 'C 99'
- 2000: 'Lua 4.0' , 'Python 2.0' , 'C#'
- 2001: 'GHC 5.00'
- 2002: 'Python 2.2'
- 2003: 'Lua 5.0' , 'Io' , 'Fortran 2003' , 'GHC 6.0'
- 2004: 'Python 2.4' , 'Java 5' , 'Scala'
- 2005: 'C# 2.0'
- 2006: 'Python 2.5' , 'Java 6' , 'CAL (Open Quark)'
- 2007: 'D 1.0' , 'C# 3.0' , 'Fortress 1.0beta'
- 2008: 'Python 3.0'
- 2009: 'Go'
- 2010: 'Rust' , 'Haskell 2010'
- 2011: 'Java 7'
- 2014: 'Swift' , 'Java 8'
- 2016: 'Python 3.6'

- **Entrega do T_1–parte A:**
  - **Prazo:** dia 24ago2017 até 23h55
  - **Forma:** individual
  - **Submissão pelo VPL-Moodle:**
    1. Ao editar, salvar e executar o código-fonte, em `linguagens.pl` e `programa.pl` (no VPL), tem-se o registro de submissão
    2. A execução/avaliação pode ser feita, pelo navegador, quantas vezes forem necessárias
    3. Os exemplos de execução são produzidos pelo próprio VPL
    4. Os predicados dever ter os mesmos nomes indicados, em cada exercício, com a mesma ordem de argumentos (de modo a permitir a automatização da correção)