The Fitness Function For Programming Languages:

A Matter of Taste?

Gilad Bracha
SAP Labs

There are two kinds of languages - those that everyone complains about and those that aren't used

- Bjarne Stroustrup

Fitness: Success in the Market?

Turelio

Examples

- C
- C++
- Java
- C#
- Visual Basic
- Perl
- PHP
- Javascript

Counter-Examples

- Lisp
- APL
- Prolog
- Beta
- Smalltalk
- Self

Today's Market, or Tomorrow's?

Faustian Bargain: Success in this life, Oblivion in the hereafter

Academic Criteria

- Theory
- Implementation
- Empirical studies

What if Smalltalk was Invented Today?

Jonathan Edwards:

Reviewer 1 comments: You propose three new language features: encapsulation, polymorphism, and inheritance. Even though your paper was the maximum 12 pages, it discussed each of these concepts only informally, and did not do any rigorous evaluation.

What if Smalltalk was Invented Today?

Reviewer 2 comments: You claim that object orientation is in some sense more natural and intuitive than procedural programming, but offer only anecdotes and hand-picked examples as justification.

So is it just Taste?

There is nothing so practical as a good theory

- Philip Wadler

Or so rare

- Gilad Bracha

How do we Judge a Theory?

- Meta-theory?
- Implementation?
- Popularity?

How do we Judge a Theory?

- Consistency, Comprehensiveness
- Beauty and elegance
- Predictive value

Language can be based on Theory

- Relational algebra
- Functional programming
- Parser combinators

Parser Combinators

BNF

*id = letter (letter | digit) **

Parser Combinators

BNF

*id = letter (letter | digit) **

Newspeak

id = letter, (letter | digit) star.

Parser Combinators

BNF

*id = letter (letter | digit) **

Newspeak

id = letter, (letter | digit) star.

Javanese

id = letter().seq(letter().or(digit()).star());

How it Works

id = letter, (letter | digit) star.

How it Works

id = *letter, (letter | digit) star.*

How it Works

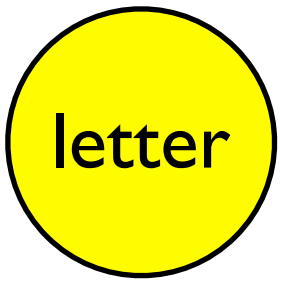
id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

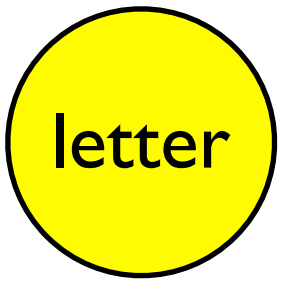
How it Works

id = letter, (letter | digit) star.



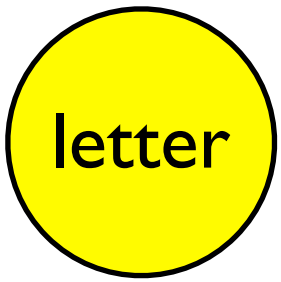
How it Works

id = letter, (letter | digit) star.



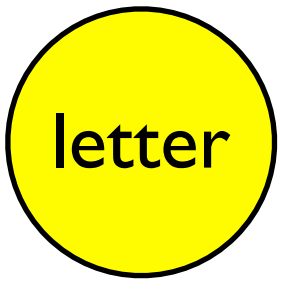
How it Works

id = letter, (letter | digit) star.



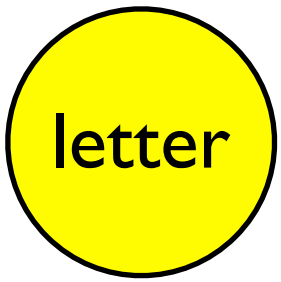
How it Works

id = letter, (letter | digit) star.



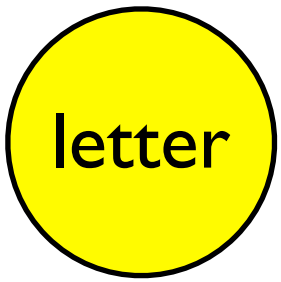
How it Works

id = letter, (letter | digit) star.



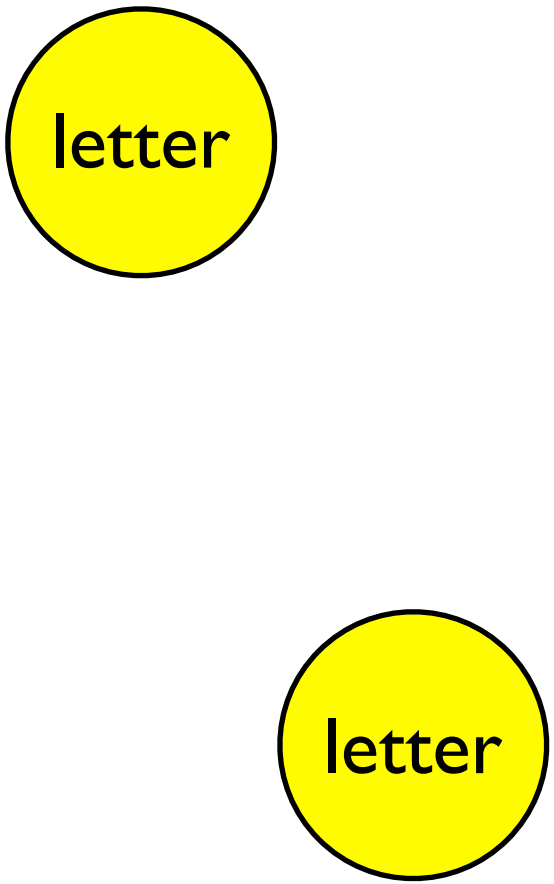
How it Works

id = letter, (letter | digit) star.



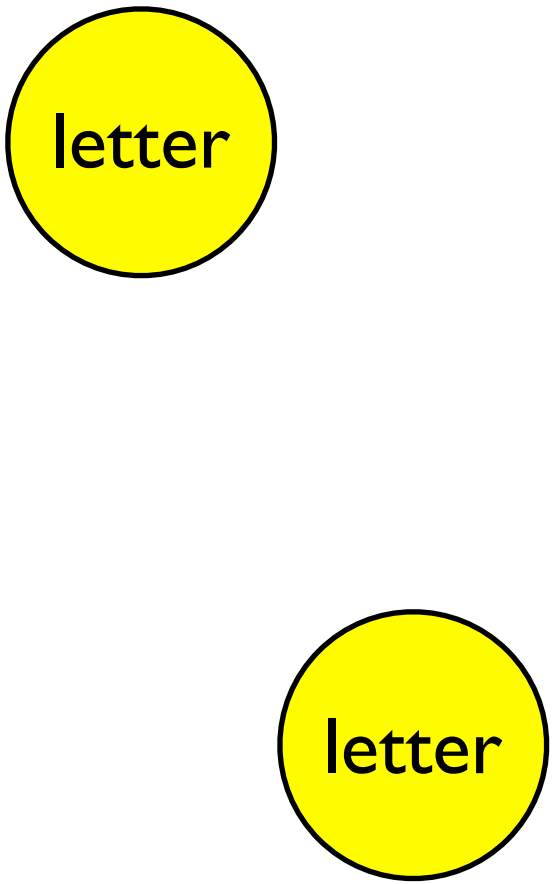
How it Works

id = letter, (letter | digit) star.



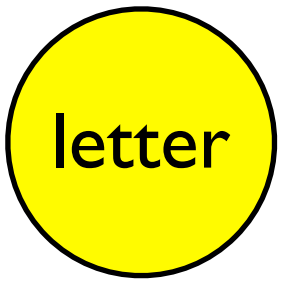
How it Works

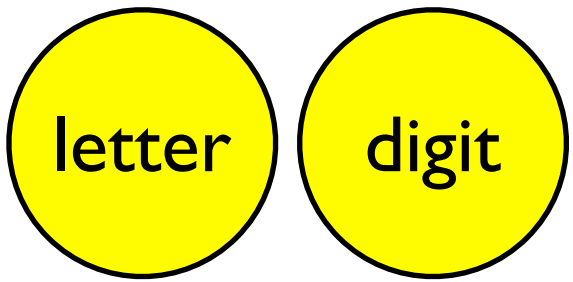
id = letter, (letter | digit) star.



How it Works

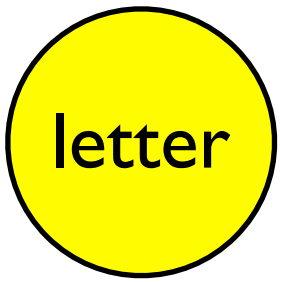
id = letter, (letter | digit) star.

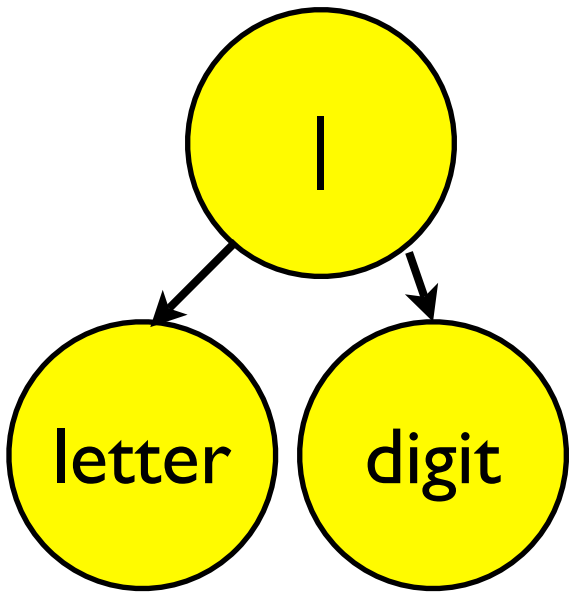




How it Works

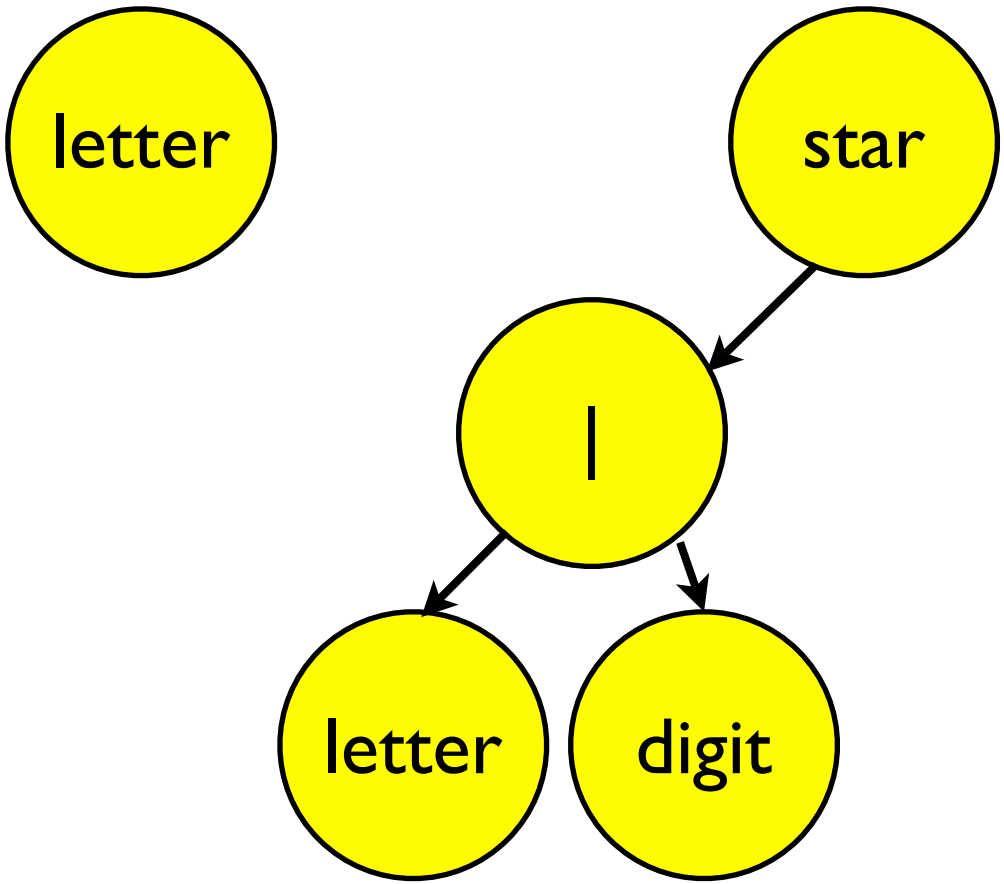
id = letter, (letter | digit) star.





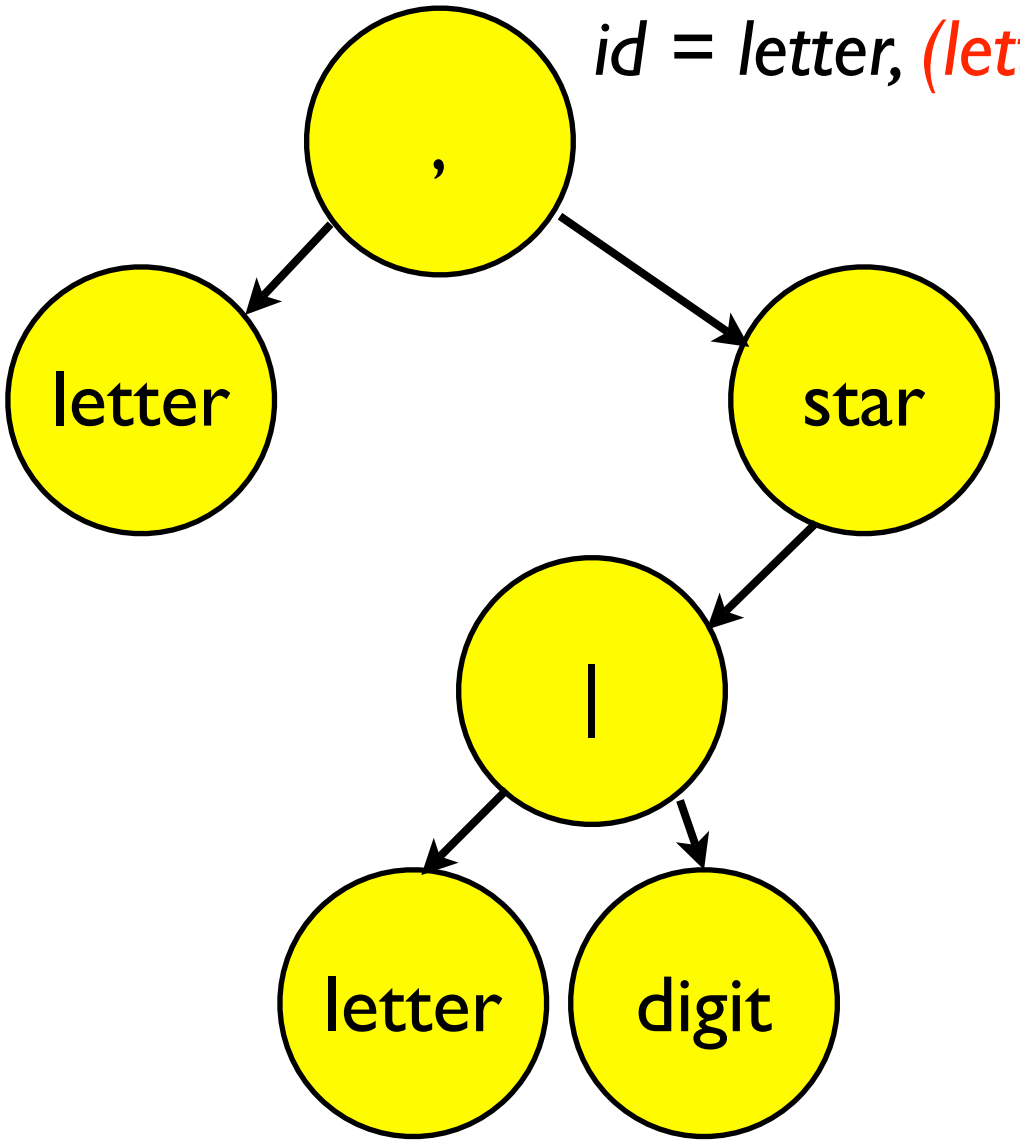
How it Works

id = letter, (letter | digit) star.



How it Works

id = letter, (letter | digit) star.



Why is this Pretty?

id = letter, (letter | digit) star.

Why is this Ugly?

id = letter().seq(letter().or(digit()).star());

Why is this Ugly?

id = letter().seq(letter().or(digit()).star());

vs.

*id = letter (letter | digit) **

vs.

id = letter, (letter | digit) star.

Why is this Ugly?

id = letter().seq(letter().or(digit()).star());

vs.

*id = letter (letter | digit) **

vs.

id = letter, (letter | digit) star.

Why is it Ugly?

A programming language is low level when its programs require attention to the irrelevant

- Alan Perlis

Compositionality

- Uniform space of values
- Operators that map this space into **itself**
- Small core is a basis for infinite space

Pattern Matching

Joint work with Felix Geller and Robert Hirschfeld at HPI, University of Potsdam

Pattern Literals

<1>
<‘a’>
<_>
<num: n>
<multiply: left by: right>

Pattern Combinators

p1 | p2
p1 & p2
p1 >> p2
p => actionBlock
p not

Pattern Combinators in Action

```
fib: n = (
   n case: <1> | <2> => [^n-1]
     otherwise:[^(fib: n-2) + (fib: n-1)]

)
```

Pattern Matching

```
class Term = ()()
class Num of: n = Term ( | val = n. | )
( match: pat = ( ^pat num: val. ))
class Var named: n = Term ( | name = n. | )
( match: pat = ( ^pat var: name. ) )
class Product of: n by: m = Term ( | left = n. right = m. | )
( match: pat = ( ^pat multiply: left by: right. ) )
```

Pattern Matching

```
simplify: expr = (
    ^expr case: <multiply: ?x by: <num: 1>> => [x]
                otherwise: [expr].
)
```

Higher Order Patterns in Action

```
simplify: expr = (
    ^expr case: <multiply: ?x by: <num: 1>> => [x]
                otherwise: [expr].
)
```

Language can be based on Theory

But, more importantly

Language may *be* the Theory

- APL: Vectors
- Beta: Patterns
- Smalltalk, Self : Objects

Programs are Models; Languages are Theories for building Programs

Judge Languages as Theories

- Consistency
- Comprehensiveness- does it model what I want? How easily and how accurately
- Beautiful/Elegant (compositional)
- Predictive value
  - Can easily can I tell
    - What a program does
    - How hard it is to build a program

Good Aesthetics makes Good Software