



## #Dasher

Progress with #Dasher, a  
Continuous Gesture IDE

Luke Church  
luke@church.name

# Agenda



- Dasher and Source Code
- **Creating Source Code**
  - Lexical Issues
  - Parsing Issues
  - Semantic Issues
- Other Inference Sources
- Other Behaviours
- Applications and Extensions



# Terminology

- C# - .NET OOP language (MS)
- Language Theory
  - Lexical analysis - Chars -> Tokens
  - Parsing - Tokens -> Structure
  - DFA -> Deterministic Finite Automata, deterministic state machine

# Cognitive Dimensions



- Cognitive framework for interactions with notations systems (Thomas Green et al)
- Viscosity – Resistance to making a small change
- Hidden Dependencies
- Visibility
- ...



## Dasher and Source Code

# Why do software development by continuous gesture?



- Why?

- Disabilities (RSI!)
- Information driven exploration of decision space
- Interesting implications

- How?



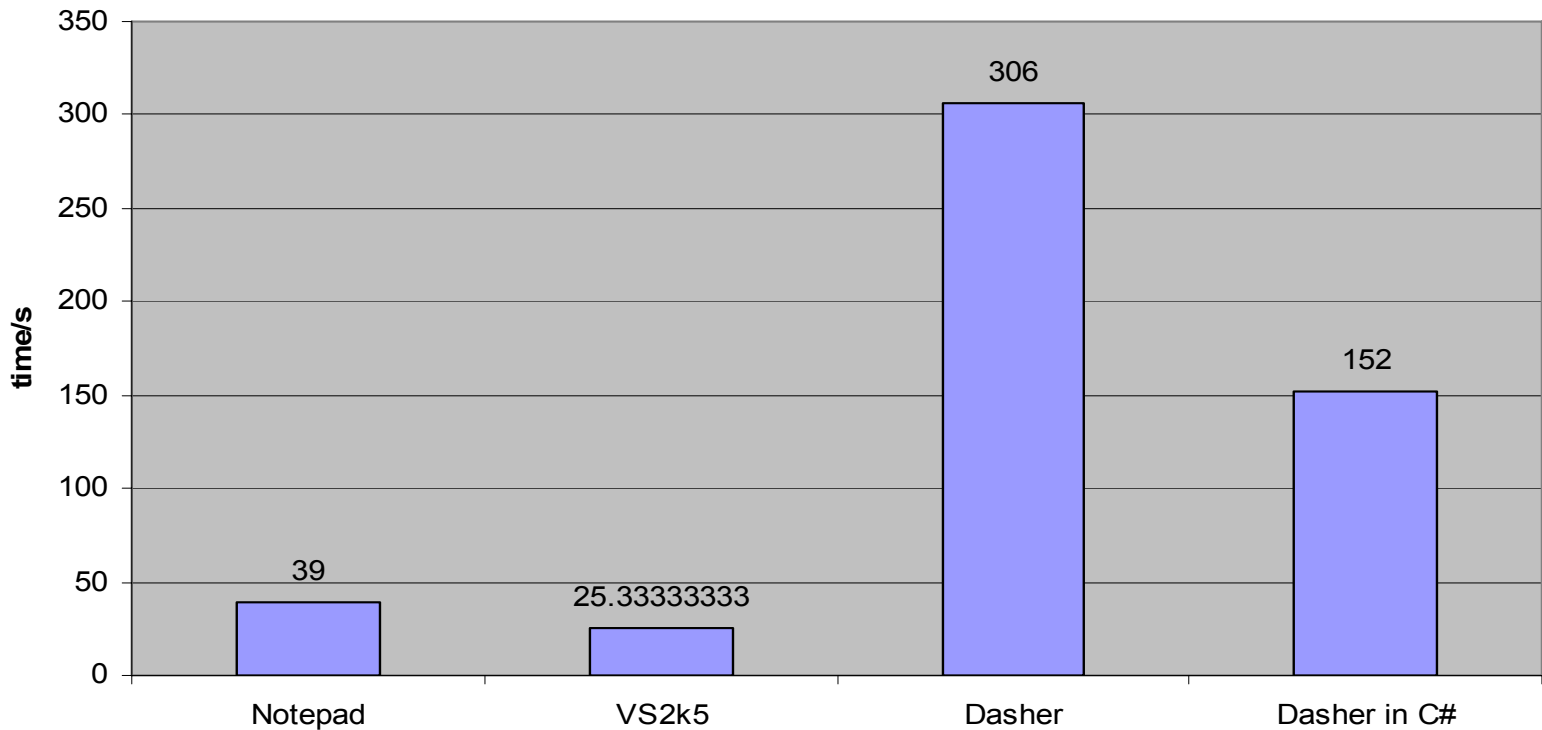
# C# with Dasher?

- Usability Problems
  - User interface
  - High viscosity
    - Navigation & non-linear entry support
    - Language model issues
- [DEMO]

# Performance



Average time to write Hello World in C#



**NB: This will not scale linearly!**



# #Dasher

- Dasher for source code
- Language aware UI & model
- We need
  - Code Prediction
    - Entry, [Decision space]
  - Navigation
    - Editing and exploration, [Code space]
  - Debugging
    - [Behaviour space]

# Creation Language Modelling



- **Usability driven**
  - PPM theoretically good, practically very bad
  - Language usage aware models
- **Information Perspective**
  - Source code is low entropy
  - Lexical, Syntactic and Semantic generation
- **Deep Information**
  - Subtle inference sources



# Creation Behaviour

- Information perspective:
  - Stream of decisions
- Token, not char, based
- What can come next?
  - `for (int x = 0; x < 10; ???)`
    - E.g. `x++`
  - `Console.???`
    - E.g. `Console.WriteLine(...)`
  - `public static void Main() { ???`
  - `string ???`
    - E.g. `string str = "I am a fish";`



# Inadequacy of PPM

- Shown to have usability issues
- PPM violates Language rules etc.
- C# has long range dependencies
- Common naming schemes are worst case for PPM

PPMD5 Window

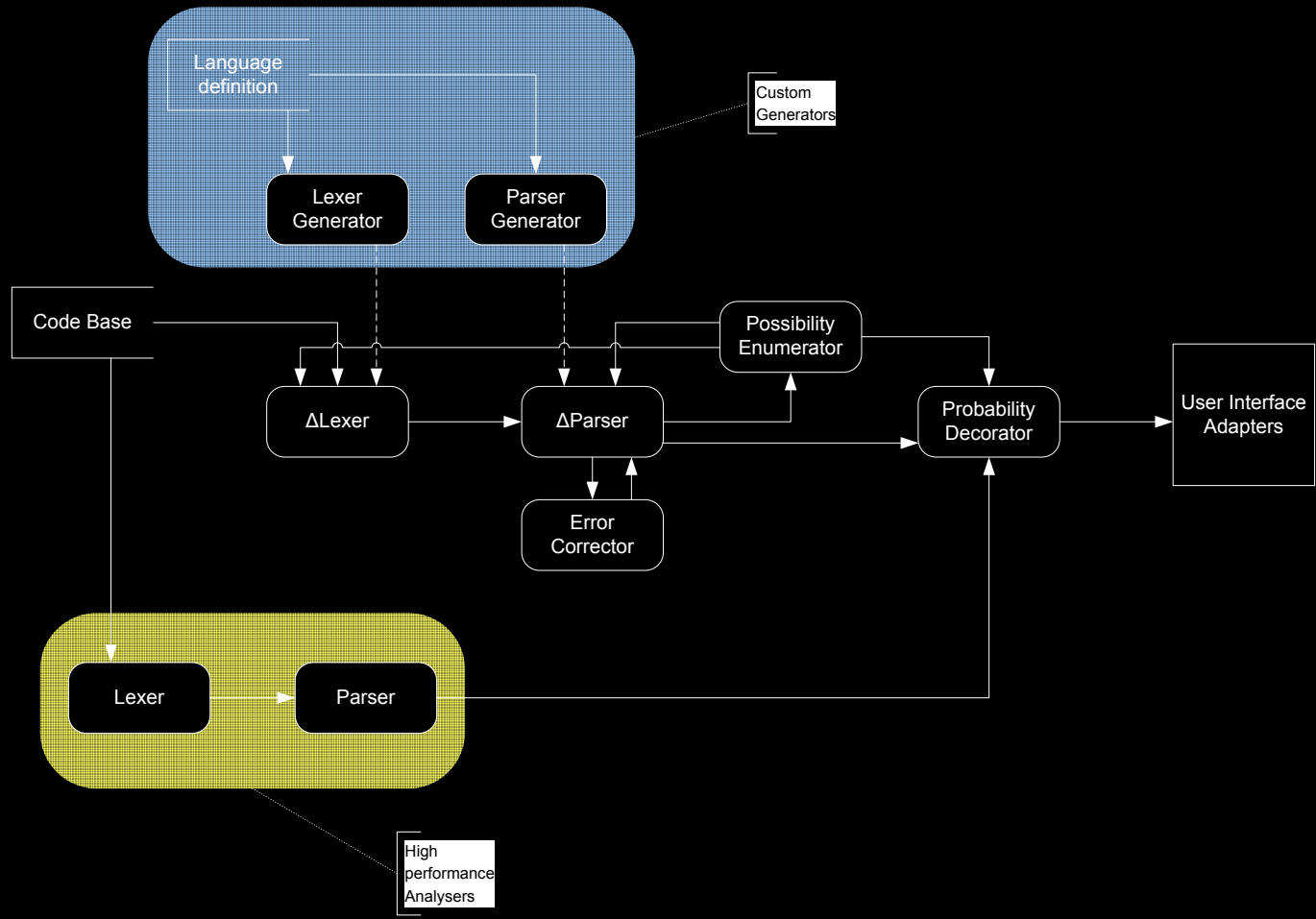
w	i	d	g	e	t	M	a	n	a	g	e	r	.
			l	o	g	M	a	n	a	g	e	r	.



# Beyond PPM

- Simple token PPM is not enough
- We need:
  - P(Continuations | (Language && History))
  - Illegality tolerance
  - Fast (min 20fps)

# Architecture of #Dasher's Entry Mode



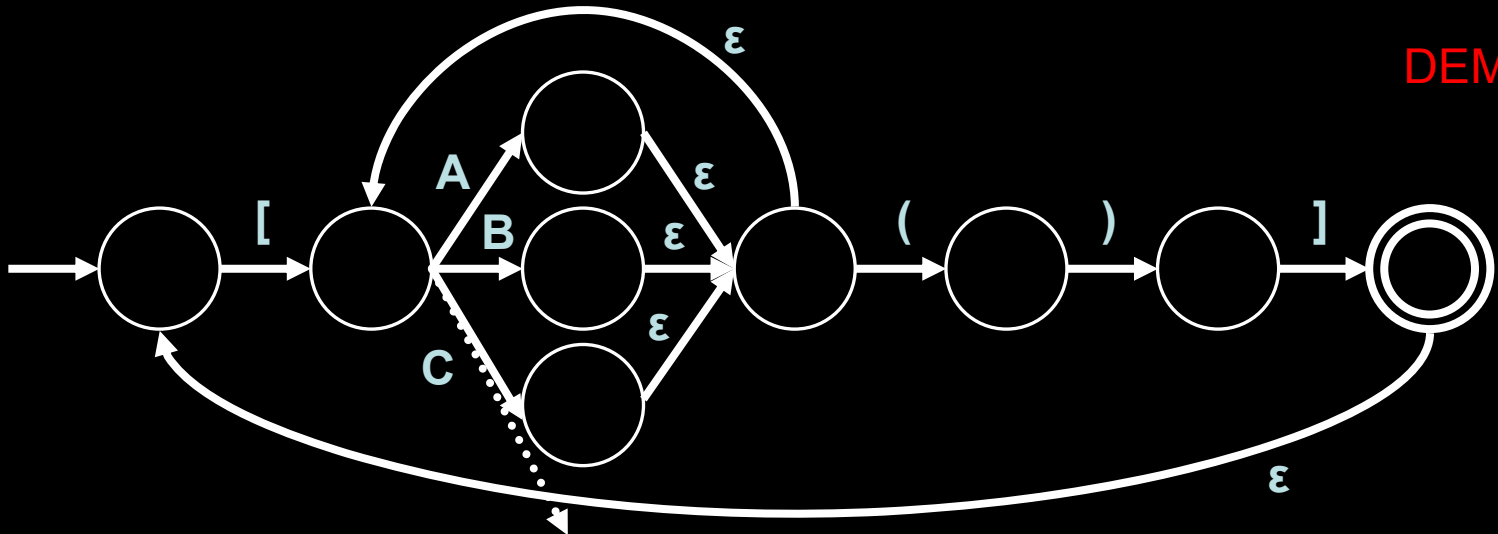


# Lexical Generation



# Lexical Generation

- Next symbol is an 'μ-attribute'
  - Regex in lang definition:
    - $(\backslash [A-Z]^+ \backslash (\backslash) \backslash)^+$
    - Regex  $\rightarrow$  NFA- $\epsilon$   $\rightarrow$  DFA  $\rightarrow$  Back-DFA
      - Arcs are legal options





# Lexical Generation

- It works!
- But...
  - Internationalisation
    - E.g. Unicode and `.`
  - What chars are next, given either an ID or an attribute?
    - On the fly NFA->DFA is painful
    - -> Avoid using cheap back generation (DFA -> DFA, typically  $O(n)$ )
- I'm not worried 😊



Parsing



# Parsing LL(1)

## Factored Language def

Stm  $\rightarrow$  SingleStm( ; SingleStm)\*

SingleStm  $\rightarrow$  id := Exp  
 | printline ( ExpList )

Exp  $\rightarrow$  SingleExp ( Binop SingleExp)\*

SingleExp  $\rightarrow$  id | num

ExpList  $\rightarrow$  Exp( , Exp\*)

Binop  $\rightarrow$  + | - | \* | /

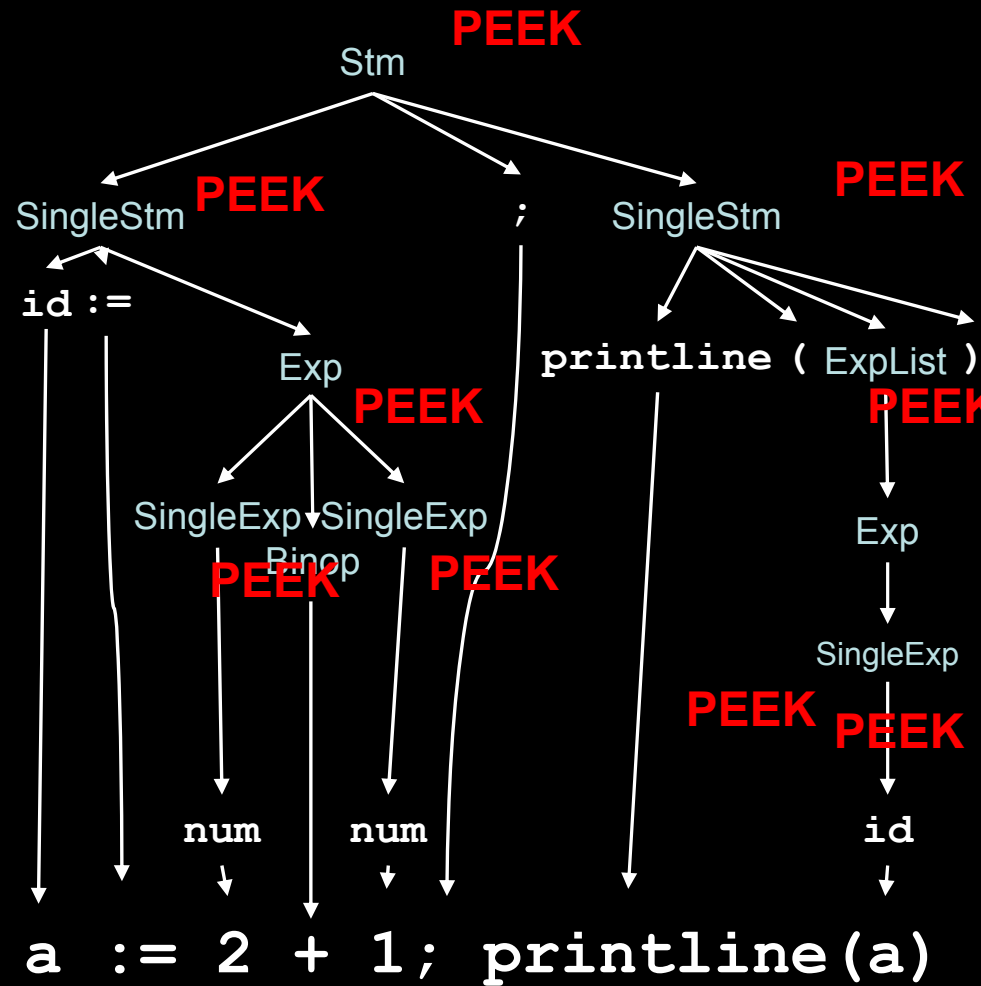
## StartSets

SingleStm  $\rightarrow$  id, printline

Exp  $\rightarrow$  id, num

ExpList  $\rightarrow$  id, num

Binop  $\rightarrow$  +, -, \*, /





# Partial Parsing

- But we're writing the source code...
  - What can come next?
- LL(k) languages only (for now)
- Recursive descent

# Partial Parsing LL(1)



## Factored Language def

Stm  $\rightarrow$  SingleStm( ; SingleStm)\*

SingleStm  $\rightarrow$  id := Exp  
 | printline ( ExpList )

Exp  $\rightarrow$  SingleExp ( Binop SingleExp)\*

SingleExp  $\rightarrow$  id | num

ExpList  $\rightarrow$  Exp( , Exp\*)

Binop  $\rightarrow$  + | - | \* | /

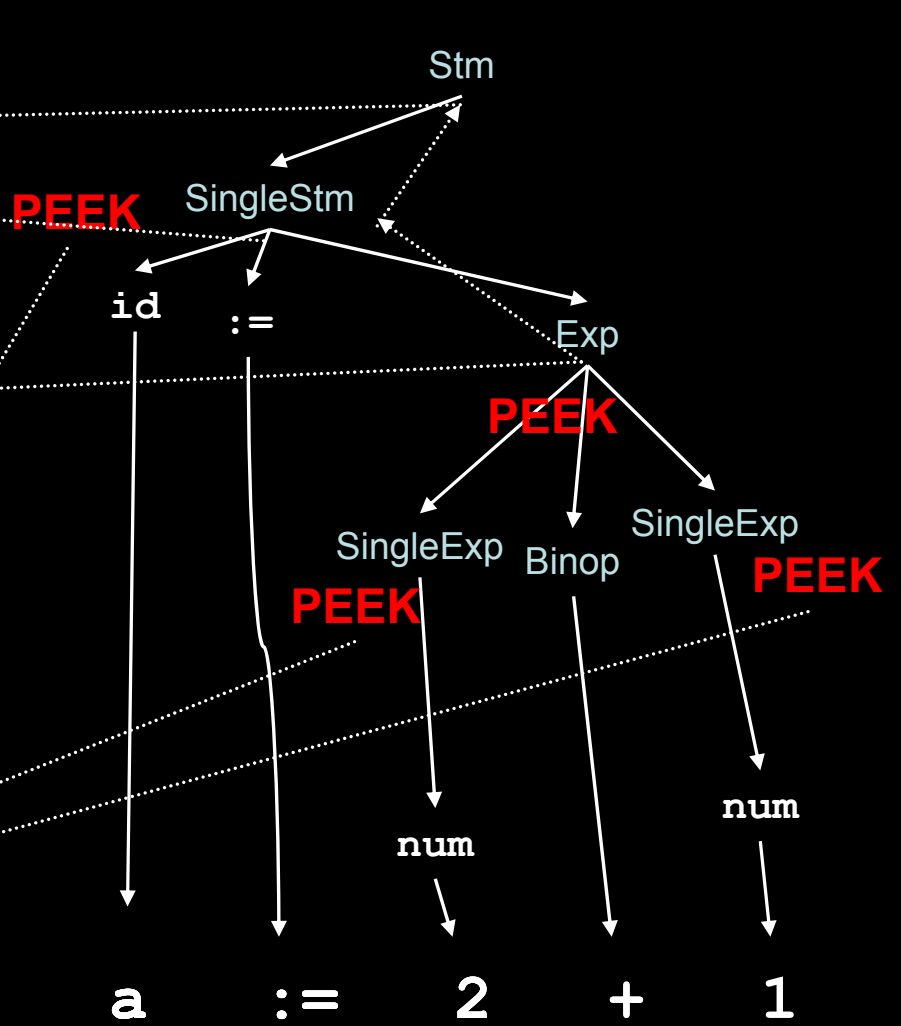
## StartSets

SingleStm  $\rightarrow$  id, printline

SingleExp  $\rightarrow$  id, num

ExpList  $\rightarrow$  id, num

Binop  $\rightarrow$  +, -, \*, /

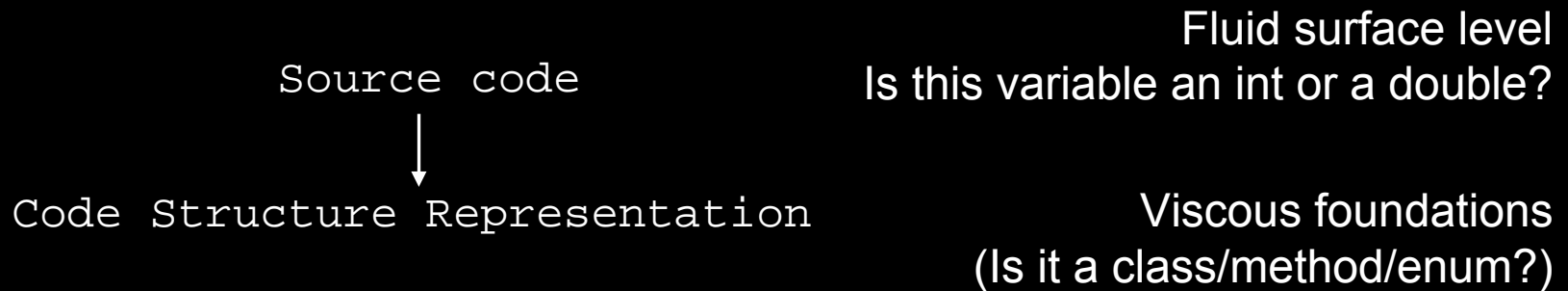


Options: lexChars(id) | lexChars(num) DEMO



# Illegality Usability

- Can't require legality
  - => Atomic operations
    - Code surface work => \*Bad\* for C#
- Low viscosity at structural level causes instability
  - => Add viscosity at structural level!



# Illegality Management



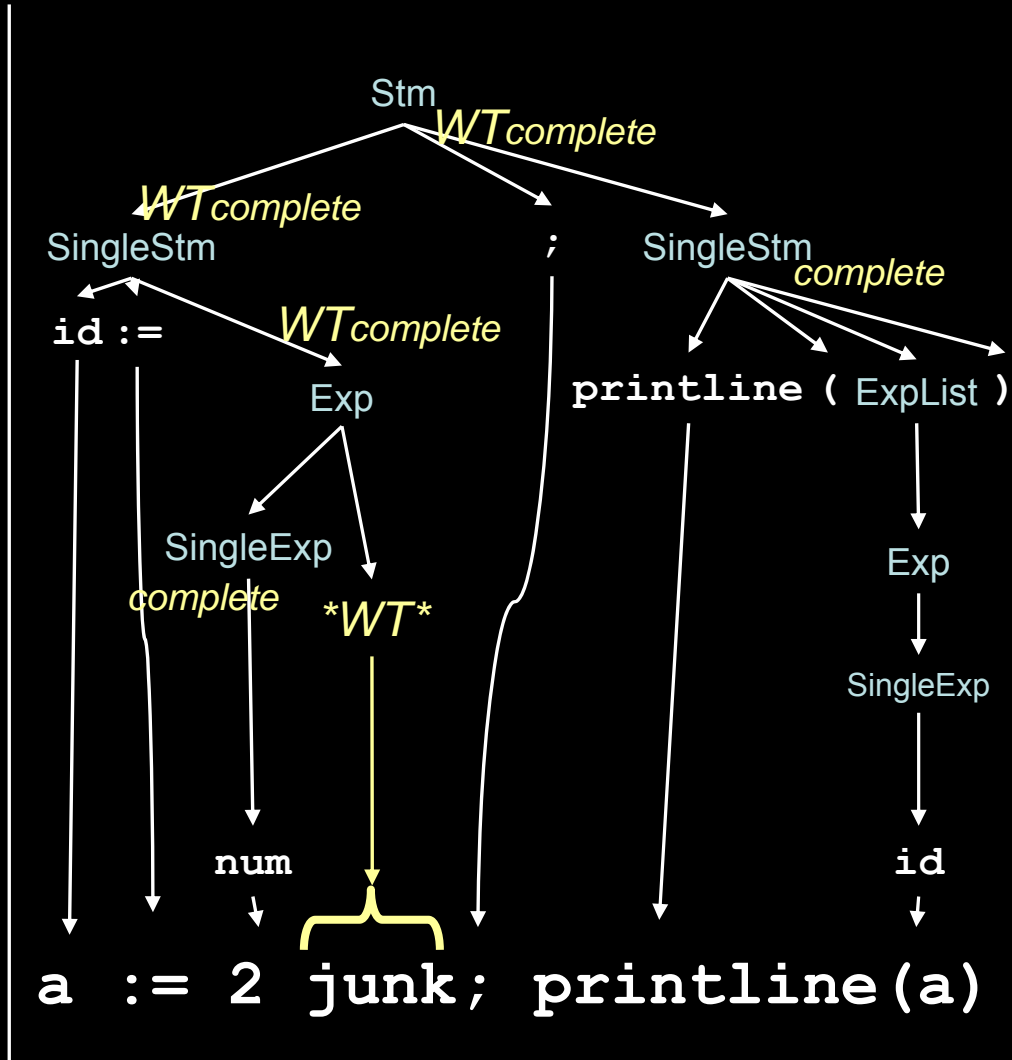
- Best Endeavours only
- Wild Tokens
  - Change minimisation
- Structural Viscosity
- Semantic Issues (types and scoping)
- But: **Will it be usable?**
  - Viscosity balance?
  - Cognitive cost of dependencies?

# Partial Parsing LL(1)



*\*WT\** : Wildtoken  
*WTcomplete* : Wildtoken Complete  
*complete*: Complete

- Parsers are too complex to build by hand
- Temporal dependency makes codebase even more complex



# Partial Parsing LL(k)



- **k** lookahead, need **k** tokens to work out what's next
  - Form up to **k** parallel trees
  - Offer all to user
  - Prune possibilities

Using Directive  
Namespace Declaration  
Class Declaration  
Enum Declaration  
Struct Declaration  
Interface Declaration  
Delegate Declaration

~~Constant~~  
Field  
Method  
Property  
~~Event~~  
~~Indexer~~  
~~Operator~~  
~~Constructor~~  
~~Destructor~~  
~~Static Constructor~~  
~~Type~~

```
class Camel {          static          int          Kick          (
```



## Semantic Analysis and Other Inference Sources



# Semantic Analysis

- Type inference
- Type manipulations
  - Methods, Properties, Casts etc.
- E.g. `Math.Min(int, int)` overload
  - But: `"Fish".Length` is an `int`
  - $\Rightarrow$  `Math.Min(2, "` is a legal partial-statement
- Powerful in a good OO design but not for base types

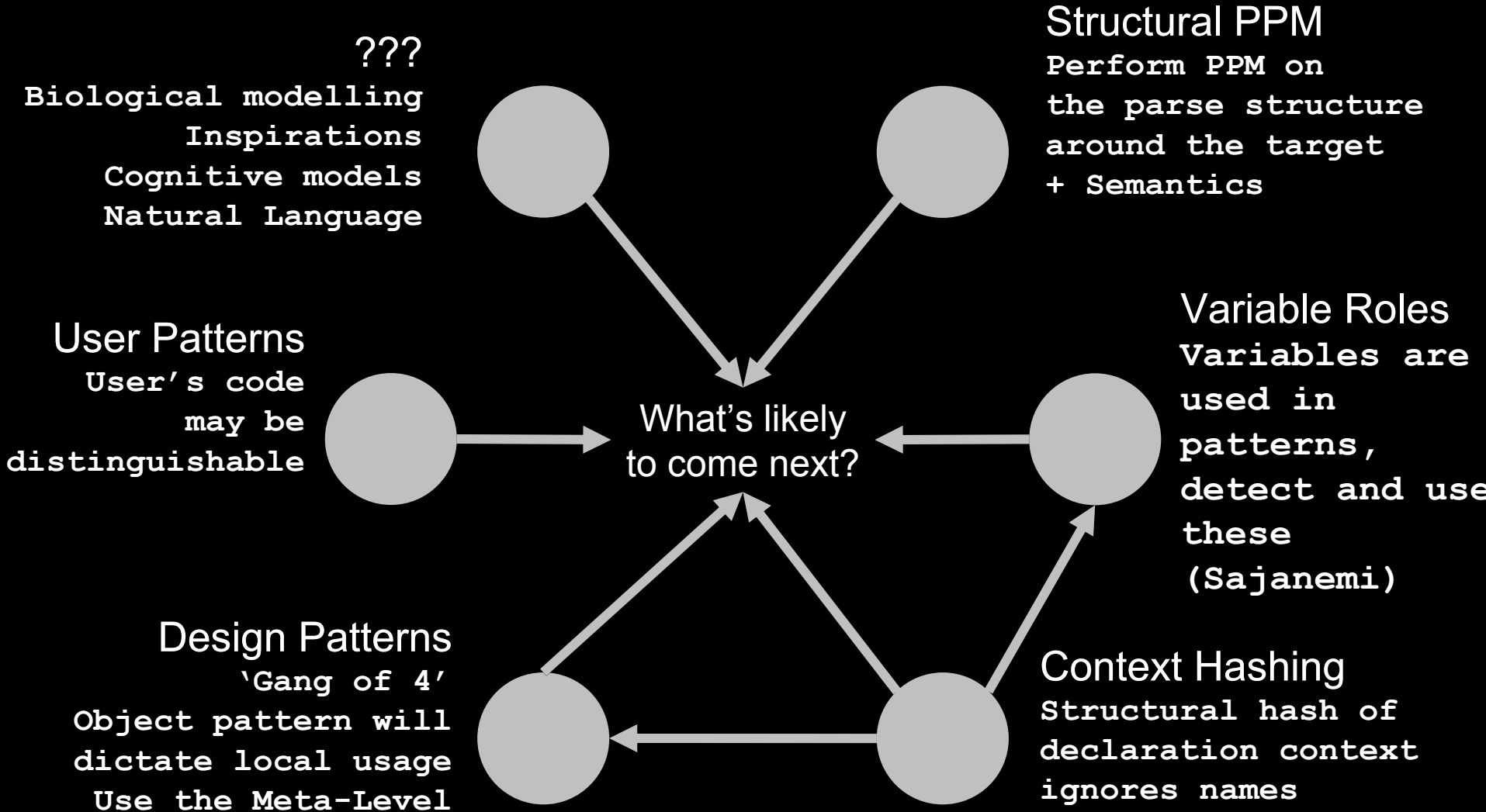
---

`Math.Cos` requires a double  
`x : Object  $\forall$  x`  
`Object.ToString().Length`  $\rightarrow$  `int`  
`int` implicit upcast  $\rightarrow$  `double`  
 Any object valid

`Evo.Breed` requires an organism  
`Population[i] :`  
     `i is int`  $\rightarrow$  `organism`  
 No other paths from items in scope  
 $\rightarrow$  `Cast`, or `Population`



# Inference Sources





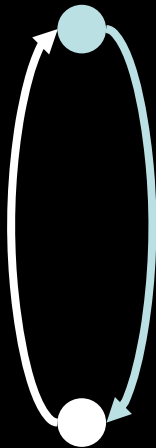
Other Behaviours



# Navigation Behaviour

- Screen recording, but only 1 subject
- **Navigation is a primary activity**
- Where is the user going to navigate to next?
- **Local vs Long Range vs Exploratory**
- Navigation Patterns:

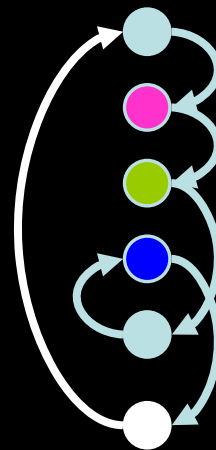
Navigate and Return



Cascade



Complex



Exploratory



# Navigation

- What information does the user need to get there?
  - Primary Notation
    - Code Modularity
  - Secondary Notation [Cognitive Dimension]
    - Indentation
  - Manually optimised model
- [Nav Concept Demo]
- Performance: 1.5-2x improvement

# Debugging



- Key activity in modern development
- Massive juxtaposition
- Probabilistically enhanced state viewing system
- Painting with probabilities



**Applications and Extensions**

# Different Languages



- Target C# 2.0
- Adding a new language involves:
  1. Lex + yacc definition
  2. Frame tracking and semantics code
  3. Navigation and structural information
  4. Semantic code-space descriptions (goto)
  5. Project management code (assemblies, refs etc.)
  6. Training sets
- Intend to support: C#, Java, ML,  $\mu$ IL (For GA work), C++
- It's not as easy as with Dasher ☹️

# Applications



- Conventional IDE enhancements
  - Better code prediction
  - Better navigation and exploration
  - Information driven debugging
  - Better static code analysis through pattern recognition
- Unconventional IDEs
  - #Dasher
  - 'Dashing with scissors' (Chuck, Princeton)
  - Speech driven development



# ++Applications;

- **Enhanced Genetic Algorithms Coding**
  - **Bitstream -> Prob Model -> GA Code**
    - Legality enforcement on encoding model
  - **GA Code language optimisation**
    - Stop frame-shift mutations
    - Code surface comparisons to determine language properties
    - 'Evolutionary cognition': Evolution as a notation system



# Summary

- #Dasher is making progress
- Lexical analysis is solved
- Parsing requires engineering work to test error tolerance
- Inference and Navigation are waiting on parsing system
- Debugging is still an idea...
- UI is a matter of engineering
- Applications seem interesting



# Questions?

[www.SharpDasher.net](http://www.SharpDasher.net)



# AI Techniques

- Techniques from the AI world
  - Adaptive look-ahead with pruning
    - Do the most probable thing next
    - Parallelise
- Never change what's already on the screen
  - Tree freezing
    - Tuned for usability



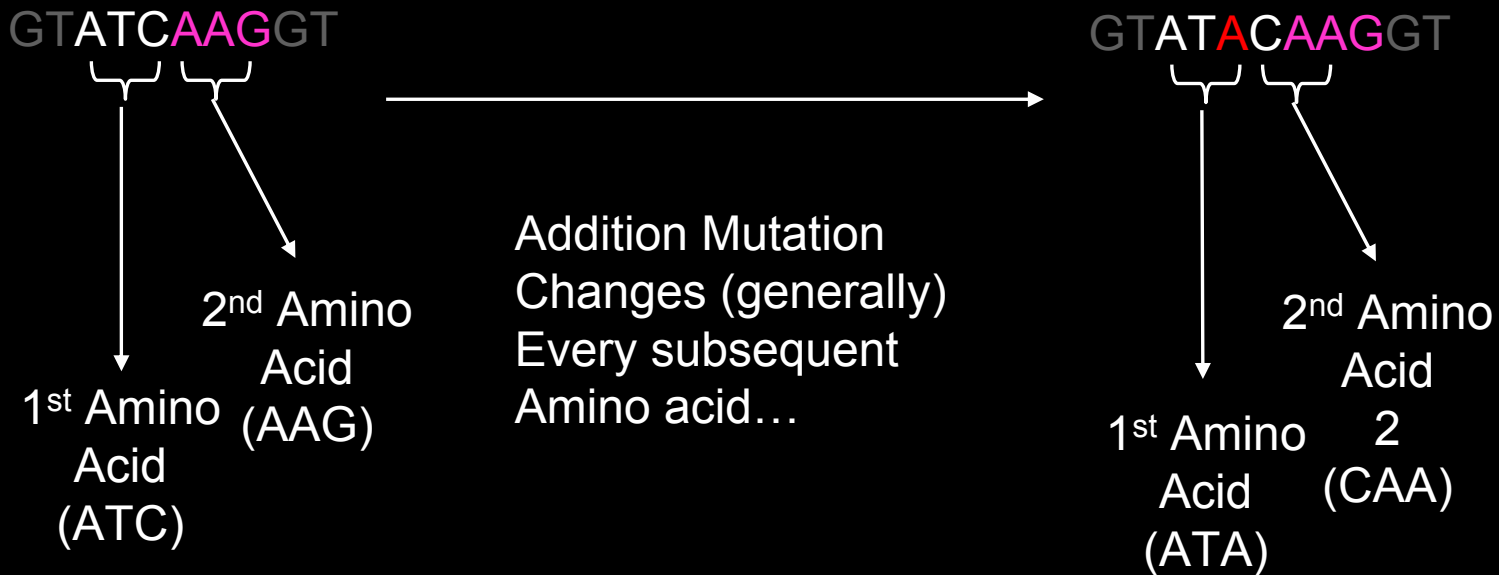
# Engineering Issues

- Is #Dasher a computer game or an IDE?
  - Precise timers, DirectX
- Explicit parallelism, but only in the code store, the rest inherits the parallelism
  - SMP/Dual Core optimised
  - But, the code store takes 700 lines of C# and 2200 lines of test cases to store a string!
- Extensible Pipeline
  - Graphics, Geometry mutators
  - Events, mutators are handed events to 'deconvolve'
  - Mutators must be trusted, otherwise results are arbitrary
- #Dasher's language processing core is de-coupled, and is built to support a standard IDE

# Frame-Shift Mutations



## Biological scenario:





# GA Scenario

- **Bit stream -> Decisions -> Src**
  - Overload 1 requires 2 arguments
  - Overload 2 requires 3 arguments
  - Frameshift in decisions
- **Possible solutions**
  - Make decisions proportional to probability
    - Apply an increasing probability for the return to previous code
      - Correction mechanism
  - Use atomic languages (Machine code, EvoCode...)
    - How can we use the compressor to make these as efficient as possible?



# Variable Roles

- Variable usage occurs in patterns
- Analysis of these patterns results in 'roles' -> Think Design Patterns on a variable scale
- Sajjanemi et al
- Use to predict future usage of an identifier
- Imperative languages only, but a similar analysis might yield results for functional languages



# Context Hashing

- "A rose by any other name would compile as well"
- Names are a meta-level, yet the distinction they represent is not
- Decorate with hash computed from the variable's structural declaration context
- Note - Meta levels can also be used for as information sources