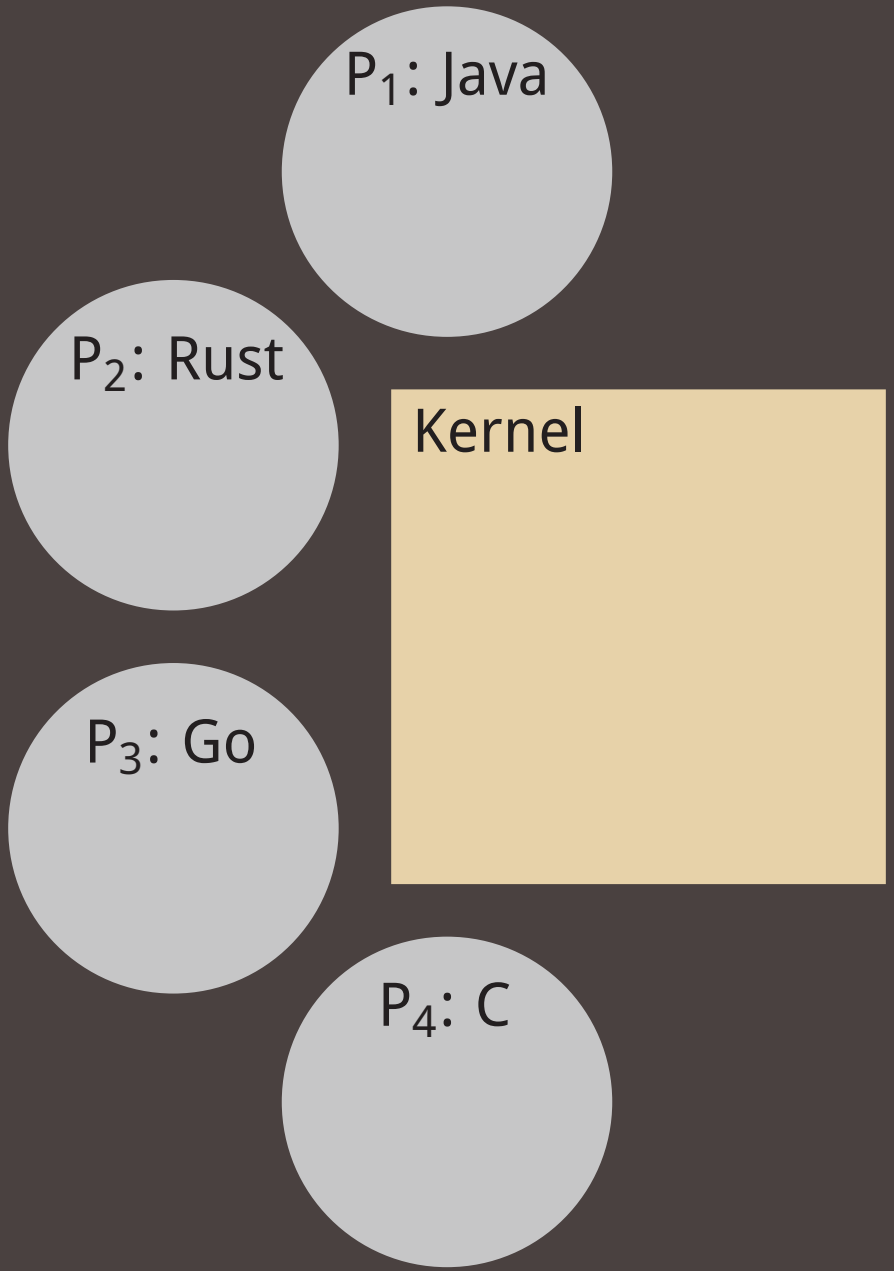
On the Generality and Convenience of Etypes

W. Michael Petullo Joseph Suh

United States Military Academy
West Point, New York USA

May 21, 2014

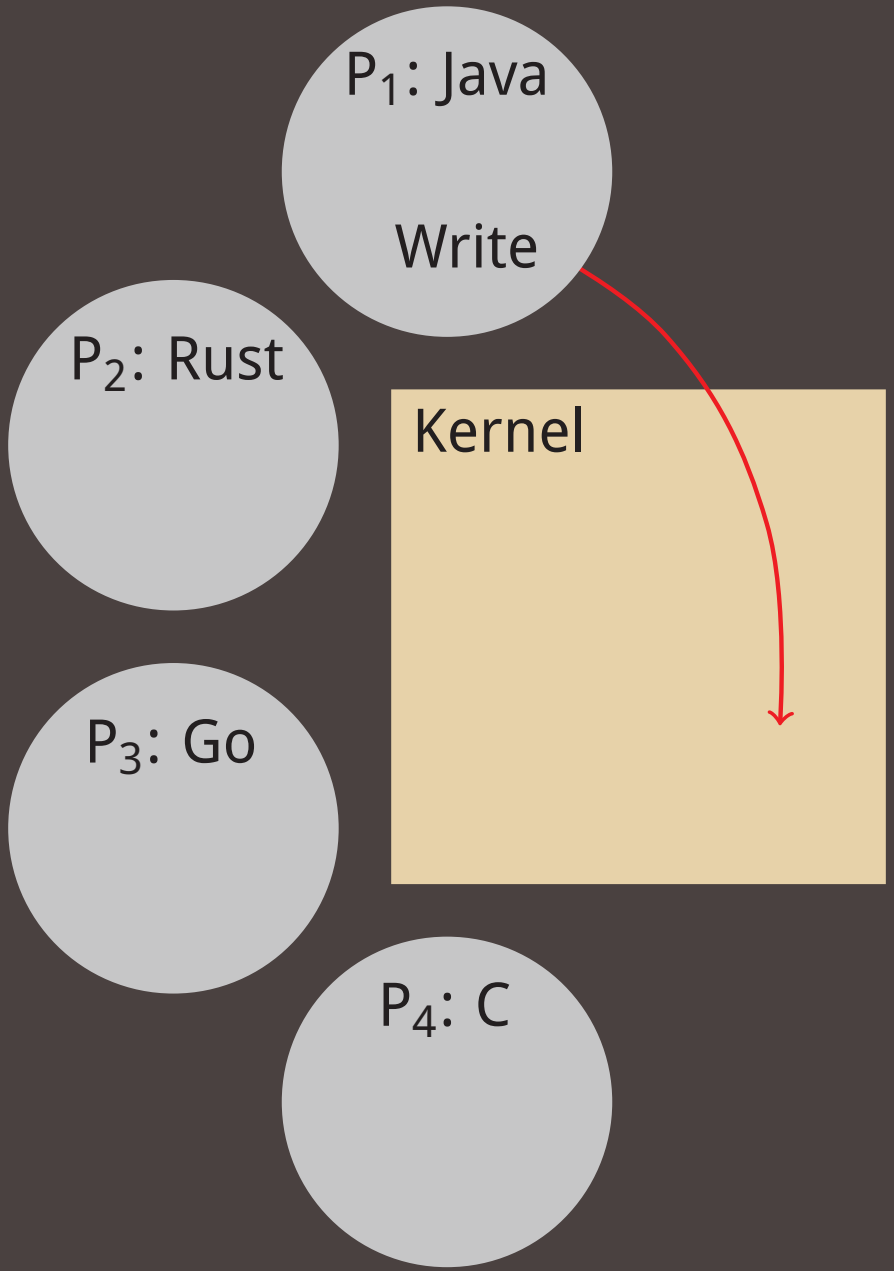
Ethos: an OS with security as its first goal



- Robust security services
- Higher-level abstractions
- Abstractions that are designed to compose
- Due to *complete mediation*, applications cannot avoid protections provided by the OS
- Declare types of OS objects
- LangSec protections inherent to system calls: recognize I/O

Details presented at LangSec 2014

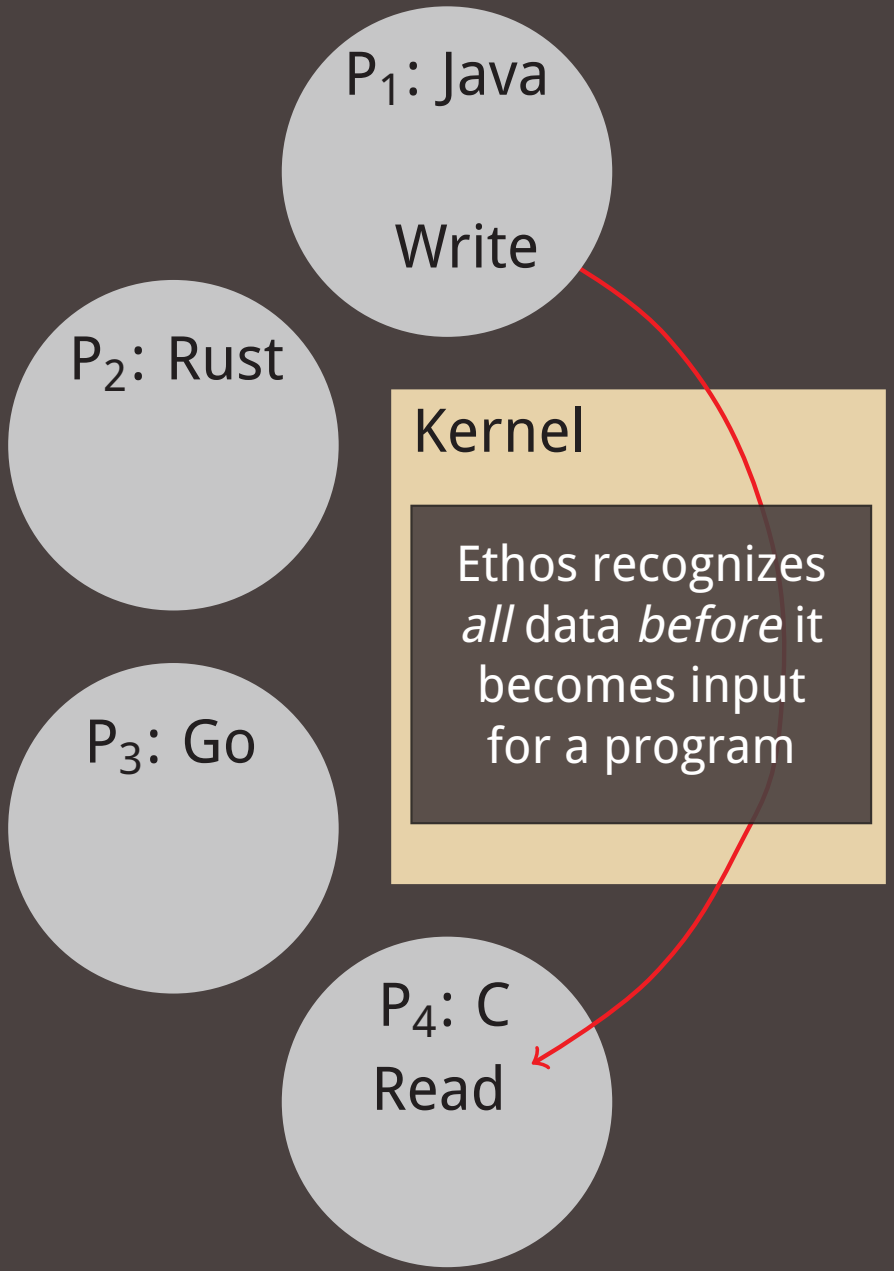
Ethos: an OS with security as its first goal



- Robust security services
- Higher-level abstractions
- Abstractions that are designed to compose
- Due to *complete mediation*, applications cannot avoid protections provided by the OS
- Declare types of OS objects
- LangSec protections inherent to system calls: recognize I/O

Details presented at LangSec 2014

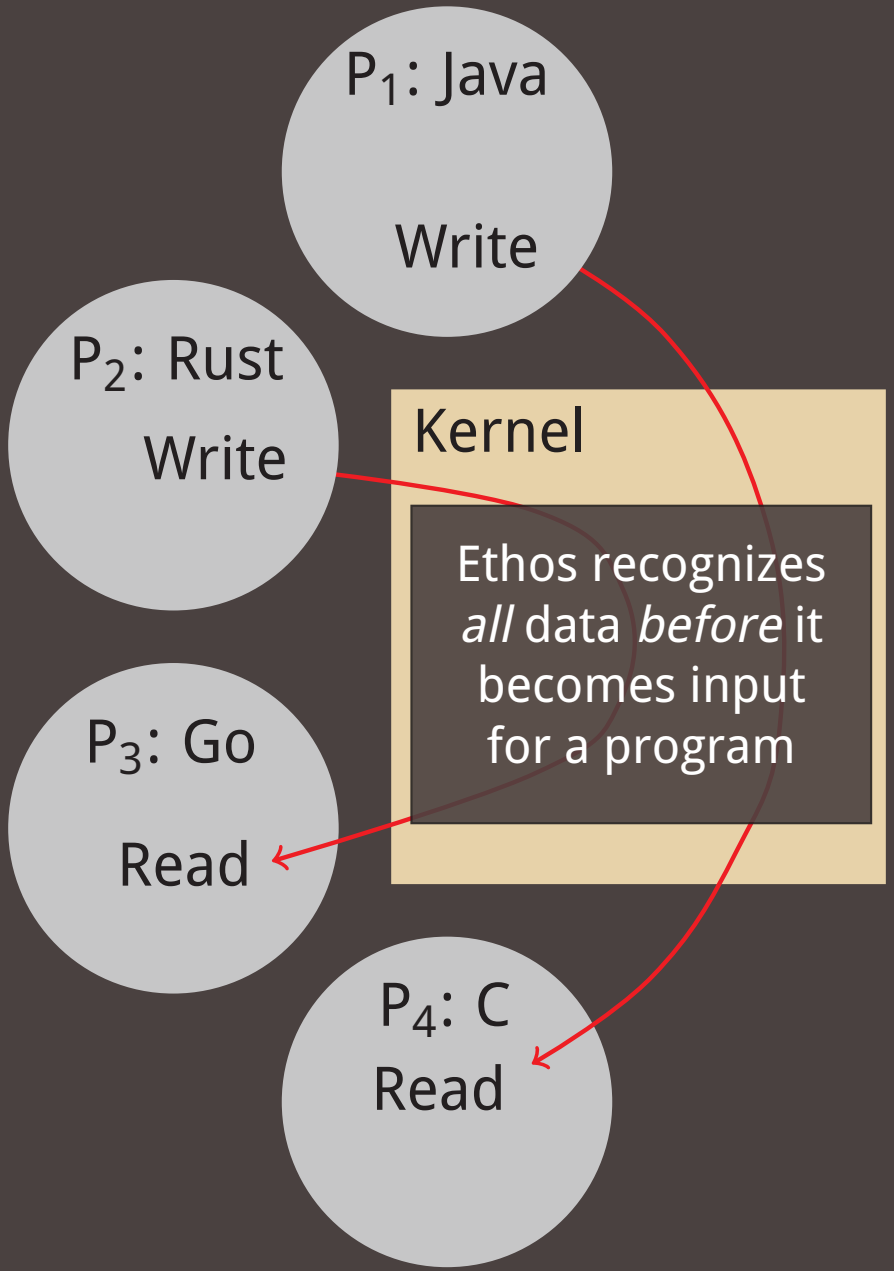
Ethos: an OS with security as its first goal



- Robust security services
- Higher-level abstractions
- Abstractions that are designed to compose
- Due to *complete mediation*, applications cannot avoid protections provided by the OS
- Declare types of OS objects
- LangSec protections inherent to system calls: recognize I/O

Details presented at LangSec 2014

Ethos: an OS with security as its first goal



- Robust security services
- Higher-level abstractions
- Abstractions that are designed to compose
- Due to *complete mediation*, applications cannot avoid protections provided by the OS
- Declare types of OS objects
- LangSec protections inherent to system calls: recognize I/O

Details presented at LangSec 2014

Important components

eNotation A type and interface description language

eCoding Ethos' wire format

et2g Takes eNotation and produces a type hash and machine-readable type descriptions (*type graph*)

eg2source Takes a type graph and produces programming-language procedures which:

1. Given a programming-language type, produce its eCoding (*encode*)
2. Given an eCoding produce its programming-language type (*decode*)
3. Given an eCoding and a type, either accept or reject the eCoding (*recognize*)

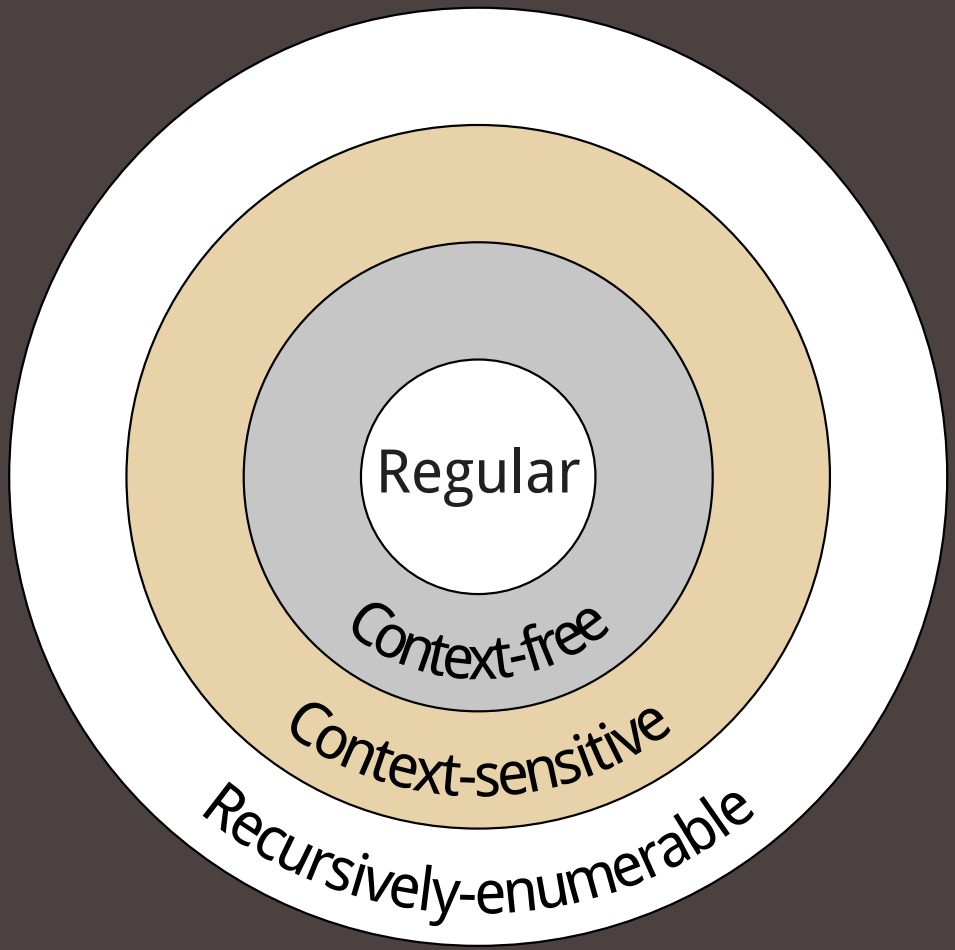
Resulting procedures used both in kernel and user space.

Remaining challenges

Such a system must be:

1. general, due to its universality, and
2. convenient, or programmers will bypass its protections, e.g.:

Generality



$\langle S \rangle ::= \epsilon$

$\langle A \rangle ::= \langle B \rangle \langle C \rangle$

$\langle A \rangle ::= a$

We show that eNotation can express any CFG and further it has some features which satisfy context-sensitive requirements

bnf2etn

Input: Grammar G in relaxed CNF.

Output: eNotation which defines types sufficient to describe any syntax tree which follows from statements legal under G.

Relaxation:

$$\langle A \rangle ::= \langle \mathrm{B} \rangle \langle \mathrm{C} \rangle \quad \Rightarrow \quad \langle X \rangle ::= \langle \mathrm{X}_0 \rangle \langle \mathrm{X}_1 \rangle \ldots \langle \mathrm{X}_n \rangle,$$

where each X need not be unique.

Transformation output

Three transformation rules which represent syntax trees using:

- Union
- Typedef
- Struct

Transformation rule 1

1. A appears two or more times on left, e.g.:

$$\langle A \rangle := \text{‘uint64’}$$

$$\langle A \rangle := \langle B \rangle \langle C \rangle$$

Generate a tagged-union type, e.g.:

```
1 A union {
2    uint64_0   uint64
3    BC_1       BC
4 }
```

Terminals reference some type already known to eTypes

Transformation rule 2

2. D appears on the left of only one production, e.g.:

$\langle D \rangle$:= ‘uint64’ $\langle E \rangle$:= $\langle \mathrm{F} \rangle \langle \mathrm{G} \rangle$

Generate a type synonym, e.g.:

```
1 D uint64
```

```
1 E FG
```

Transformation rule 3

3. Two or more elements appear on the right side, e.g.:

$$\langle H \rangle := \langle \mathrm{I} \rangle \langle \mathrm{J} \rangle$$

Generate a struct type, e.g.:

```
1 IJ struct {
2   I_0 I
3   J_1 J
4 }
```

N = bnf2etn(G)

If and only if data is well-formed with respect to eNotation N, then the data represents a valid parse tree under G.

Example

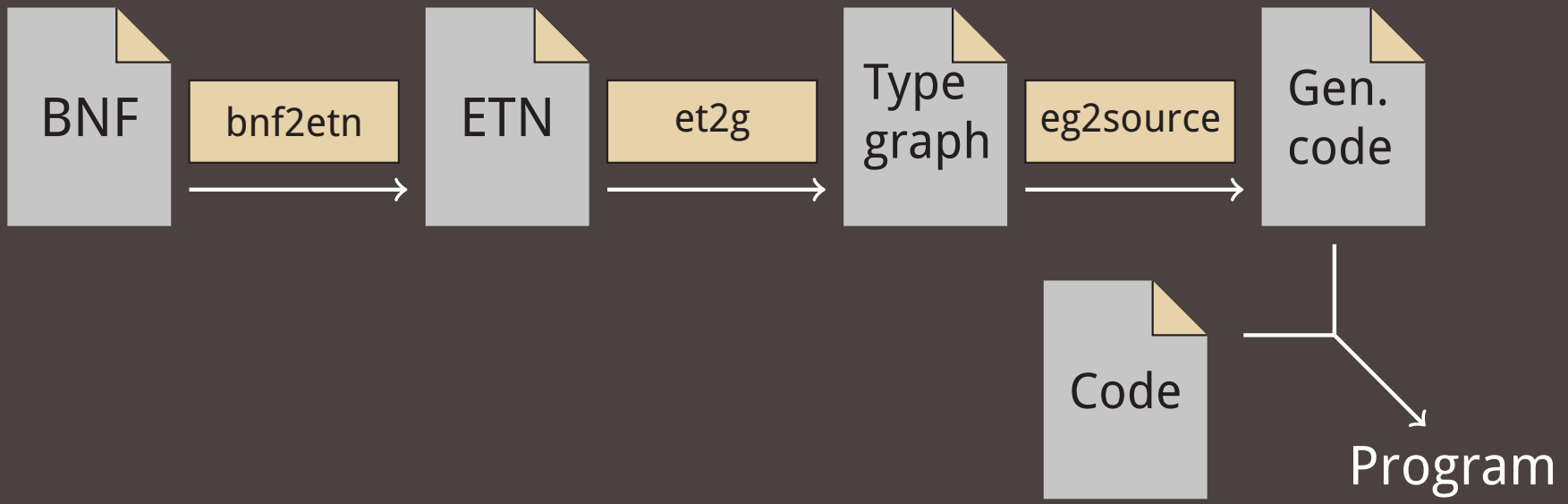
```
1 {
2   Expr = 'uint64';
3   Expr = Expr, AddOp, Expr;
4   AddOp = 'bool';
5 };
```

```
1 Expr union {
2   uint64_0 uint64
3   Expr_AddOp_Expr_1 *Expr_AddOp_Expr
4 }

6 AddOp bool

8 Expr_AddOp_Expr struct {
9   Expr_0 Expr
10  AddOp_1 *AddOp
11  Expr_2 Expr
12 }
```

Hyper

HTML-like grammar and corresponding client/server.

Programming flow



1. Specify communication grammar using Backus-Naur Form
2. Use bnf2etn to generate corresponding eNotation types
3. Use et2g to generate type graph
4. Use eg2source to generate enc./dec./rec. routines
5. Combine these routines with program logic to produce program

Convenience

Buy in gains:

- Fewer lines of code without lex/parse
- formal documentation of communication formats (eNotation)
- Use of routines which resemble existing expression libraries:

Python renderSnake:

```
1 HtmlCanvas html = new
      HtmlCanvas();
2 html
3    .ol()
4       .li().content("One")
5       .li().content("Two")
6       .li().content("Three")
7       .li().content("Four")
8       .li().content("Five")
9    ._ol();
```

eg2source-generated:

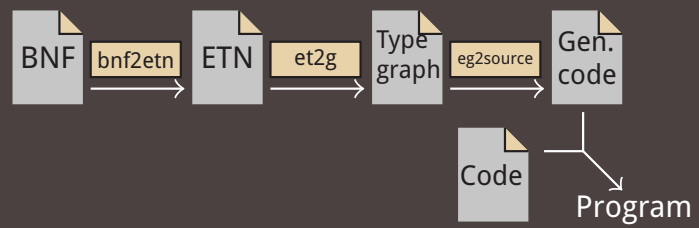
```
1 OrdList ol =
      mkOrdList("One",
2   mkOrdListPair("Two",
3     mkOrdListPair("Three",
4       mkOrdListPair("Four",
            "Five"))))
```

Conclusion and future work

Summary:

- Ethos recognizes *all* data before it serves as input to *any* application
- eNotation can specify a data structure equivalent to CFG
- Incentives in Ethos discourage cheating the recognition system

Future work:



- Support for unrestricted EBNF
  - Fewer productions
  - More closely matches “reasonable” grammar
- New programming language or preprocessor: eNotation and type graphs follow directly from programming language syntax

`http://www.ethos-os.org` — Open source this summer