It’s Time for a New Old Language

Guy L. Steele Jr.
Software Architect, Oracle Labs

MIT 6.945 Guest Lecture
Wednesday, April 5, 2017
Copyright © 2017 Oracle and/or its affiliates ("Oracle"). All rights are reserved by Oracle except as expressly stated as follows. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted, provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or re-publish, to post on servers, or to redistribute to lists, requires prior specific written permission of Oracle.
The most popular programming language in computer science
Some Early Contributors

Gerhard Gentzen
John Backus
Peter Naur
Alonzo Church
Computer Science Metanotation (CSM)

- Built-in datatypes: boolean, integer, real, complex, sets, lists, arrays
- User-declared datatypes: record / abstract data type / symbolic expression (BNF = Backus-Naur Form)
- Code: Inference rules (Gentzen notation)
- Conditionals: rule dispatch via nondeterministic pattern-matching
- Repetition: overlines and/or ellipsis notations, and sometimes iterators
- Primitive expressions: logic and mathematics
- Capture-free substitution within a symbolic expression (Church)
Example of CSM Data Declarations (BNF)

Expressions:
\[ e ::= x \quad \text{Variable} \]
\[ \lambda x : \tau.e \quad \text{Abstraction} \]
\[ e_1 e_2 \quad \text{Application} \]
\[ \Lambda \alpha : \kappa.e \quad \text{Type abstraction} \]
\[ e \tau \quad \text{Type application} \]

Types:
\[ \tau, \sigma, \psi, \upsilon ::= x \quad \text{Type variable} \]
\[ \tau_1 \to \tau_2 \quad \text{Function type} \]
\[ \forall \alpha : \kappa.\tau \quad \text{Polymorphic type} \]
\[ \tau_1 \tau_2 \quad \text{Application} \]
\[ F(\bar{\tau}) \quad \text{Saturated type family} \]

\[ \kappa ::= \star \mid \kappa_1 \to \kappa_2 \quad \text{Kind} \]

\[ \Phi ::= [\alpha]:\kappa \cdot F(\bar{\rho}) \sim \sigma \quad \text{Axiom equation} \]

Adapted from Eisenberg, Vytiniotis, Peyton Jones, and Weirich,
Closed Type Families with Overlapping Equations, ACM POPL 2014, Figure 2
Check for equation conflicts

\[ \text{no\_conflict}(\Psi, i, \bar{\tau}, j) \]

\[ \Psi = [\alpha: \kappa]. F(\rho) \sim \nu \quad \text{apart}(\rho_j, \rho_i[\tau/\alpha_i]) \]  

\[ \text{no\_conflict}(\Psi, i, \bar{\tau}, j) \]  

\[ \text{compat}(\Psi[i], \Psi[j]) \]  

\[ \text{no\_conflict}(\Psi, i, \bar{\tau}, j) \]  

Adapted from Eisenberg, Vytiniotis, Peyton Jones, and Weirich, 
Closed Type Families with Overlapping Equations, ACM POPL 2014, Figure 4
Example of CSM Code (Nondeterministic?) (2 of 2)

```
no_conflict(Ψ, i, τ, j)  Check for equation conflicts

Ψ = \[\alpha:κ]. F(\bar{\rho}) \sim υ \quad \text{apart}(\bar{\rho}_j, \rho_i[τ/\alpha_i]) \quad [\text{NC\_APART}]

no_conflict(Ψ, i, τ, j)

\text{compat}(Ψ[i], Ψ[j]) \quad [\text{NC\_COMPATIBLE}]

no_conflict(Ψ, i, τ, j)
```

Adapted from Eisenberg, Vytiniotis, Peyton Jones, and Weirich, 
*Closed Type Families with Overlapping Equations*, ACM POPL 2014, Figure 4
Another Example of CSM Code (Deterministic?) (1 of 3)

\[
\begin{align*}
\Gamma &\vdash x_i : \tau_i \\
\Gamma, x : \sigma &\vdash M : \tau \\
\Gamma &\vdash \lambda x : \sigma.M : \sigma \rightarrow \tau \\
\Gamma &\vdash M : \sigma \rightarrow \tau \\
\Gamma &\vdash N : \sigma \\
\Gamma &\vdash MN : \tau \\
\Gamma &\vdash M_i : \tau \quad (i = 1, \ldots, \text{ar}(\text{op})) \\
\Gamma &\vdash \text{op}(M_1, \ldots, M_{\text{ar}(\text{op})}) : \tau \\
\Gamma &\vdash M : \tau \times \sigma \\
\Gamma &\vdash f\text{st}(M) : \tau \\
\Gamma &\vdash M : \tau \times \sigma \\
\Gamma &\vdash s\text{nd}(M) : \sigma \\
\Gamma &\vdash M : \tau \\
\Gamma &\vdash N : \sigma \\
\Gamma &\vdash \langle M, N \rangle : \tau \times \sigma
\end{align*}
\]

Adapted from Muroya, Hoshino, and Hasuo,
Memoryful Geometry of Interaction II, ACM POPL 2016, Figure 1
Another Example of CSM Code (Deterministic?) (2 of 3)

\[
\begin{array}{c}
\Gamma \vdash x_i : \tau_i \\
\Gamma, x : \sigma \vdash M : \tau \\
\Gamma \vdash \lambda x : \sigma. M : \sigma \rightarrow \tau \\
\text{input} \\
\Gamma \vdash M : \sigma \rightarrow \tau \\
\Gamma \vdash N : \sigma \\
\Gamma \vdash MN : \tau \\
\Gamma \vdash M_i : \tau \quad (i = 1, \ldots, \text{ar(op)}) \\
\Gamma \vdash \text{op}(M_1, \ldots, M_{\text{ar(op)}}) : \tau \\
\Gamma \vdash M : \tau \times \sigma \\
\Gamma \vdash \text{fst}(M) : \tau \\
\Gamma \vdash \text{snd}(M) : \sigma \\
\Gamma \vdash \langle M, N \rangle : \tau \times \sigma \\
\end{array}
\]

Adapted from Muroya, Hoshino, and Hasuo, *Memoryful Geometry of Interaction II*, ACM POPL 2016, Figure 1
Another Example of CSM Code (Deterministic?) (3 of 3)

\[
\begin{align*}
\Gamma & \vdash x_i : \tau_i \\
\Gamma & \vdash \lambda x : \sigma. M : \sigma \rightarrow \tau \\
\Gamma & \vdash M : \sigma \rightarrow \tau \\
\Gamma & \vdash N : \sigma \\
\Gamma & \vdash M N : \tau \\
\Gamma & \vdash M_i : \tau \quad (i = 1, \ldots, \text{ar}(\text{op})) \\
\Gamma & \vdash \text{op}(M_1, \ldots, M_{\text{ar}(\text{op})}) : \tau \\
\Gamma & \vdash M : \tau \times \sigma \\
\Gamma & \vdash \text{fst}(M) : \tau \\
\Gamma & \vdash M : \tau \times \sigma \\
\Gamma & \vdash \text{snd}(M) : \sigma \\
\Gamma & \vdash \langle M, N \rangle : \tau \times \sigma \\
\end{align*}
\]

Adapted from Muroya, Hoshino, and Hasuo, Memoryful Geometry of Interaction II, ACM POPL 2016, Figure 1
Popularity of Computer Science Metanotation (1 of 2)

Use of inference rules in POPL papers (five-year intervals)

- No use of inference rules
- Inference rules

Analysis of 43 years of POPL conferences (1,401 papers / 17,160 pages)

Copyright © 2017 Oracle and/or its affiliates. All rights reserved.
Popularity of Computer Science Metanotation (2 of 2)

Use of inference rules: other recent SIGPLAN conferences

Analysis of 3 years of PLDI, OOPSLA, and ICFP, and 6 years of PPoPP (567 papers / 8,012 pages)
Structure of This Talk

- Examine history and variety of five aspects of the notation:
  - Inference rules
  - BNF
  - Substitution
  - Overline
  - Ellipsis

- Identify problems that have arisen with the last three
INFERENCE RULES
Gentzen Notation (Natural Deduction)

1935 Gerhard Gentzen creates a rule notation for *natural deduction*:

Unsere Untersuchungen über das logische Schließen (*). I.

Von

Gerhard Gentzen in Göttingen.

3.1. Eine Schlußfigur läßt sich in der Form schreiben:

\[
\frac{A_1 \ldots A_v}{B} \quad (v \geq 1),
\]

wobei \(A_1, \ldots, A_v, B\) Formeln sind. \(A_1, \ldots, A_v\) heißen dann die Oberformeln, \(B\) heißt die Unterformel der Schlußfigur.

Copyright © 2017 Oracle and/or its affiliates. All rights reserved.
Today’s Computer Science Inference Rule Notation

\[
\text{premise} \quad \text{premise} \quad \text{premise} \\
\hdashline
\text{premise} \quad \text{premise}
\]

conclusion

[Optional Label]

Wide variations in labels:

- Placement: left, right, upper left, upper center, lower right, . . .
- Separation: adjacent to rule, or against the margin?
- Capitalization: lowercase, title caps, all caps, small caps, caps + small caps
- Mathematical symbols, or just alphanumeric?
- Size and style: normalsize, small, footnotesize; roman, italic, boldface
- Word separator: space, hyphen, period, CamelCase
- Enclosers: parentheses, brackets, none

Not really a problem!
BNF
**BNF: Historical Background on Grammars**

6th–4th century BCE  Pāṇini writes the *Aṣṭādhyāyī*, a Sanskrit grammar containing numerous concise, technical rules that describe Sanskrit morphology unambiguously and completely.

1914  Axel Thue studies string-rewriting systems defined by rewrite rules.

1920s  Emil Post studies “tag systems” in which symbols are repeatedly replaced by associated strings (this work is not published until 1943).

1947  Andrey Markov and Emil Post independently prove that the word problem for semigroups (a problem posed by Thue) is undecidable.

1956  Noam Chomsky publishes “Three Models for the Description of Language,” which describes grammars with production rules and what we now call the “Chomskian hierarchy of grammars”.

Copyright © 2017 Oracle and/or its affiliates. All rights reserved.
History of **Regular Expressions** in One Slide

1951  Stephen Kleene develops regular expressions to describe McCulloch-Pitts (1943) nerve nets (uses ∨ for choice; considers postfix *, but decides to make it a *binary* operator to avoid having empty strings: “$x^*y$” means any number of copies of $x$, followed by $y$).

1956  Journal publication of Kleene’s technical report: binary $* \text{ only}$.  

1958  Copi, Elgot, and Wright formulate REs using $\cdot$ and $\lor$ and postfix $*$.  

1962  Janusz Brzozowki uses binary $+$ for $\lor$ and introduces postfix $\oplus$.  

1968  Ken Thompson’s paper “Regular Expression Search Algorithm” uses $\mid$.  

1973  Thompson creates `grep` from `ed` editor for use by Doug McIlroy.  

1975  Alfred Aho creates `egrep` (includes `( )`, $\mid$, $\ast$, $\pm$, ?).  

1978  CMU Alphard project uses regular expressions with $\ast$, $\pm$, and #.  

1981  CMU FEG and IDL use regular expressions with $\ast$, $\pm$, and ?.  

Pretty much unchanged since 1981!
1958 Alan Perlis and Klaus Samelson report on the International Algebraic Language, including “forms” for various language features.

John Backus, influenced by “Post productions” of Emil Post, uses a specific syntax to write production rules for a context-free grammar for the International Algorithmic Language.

A single production may contain multiple alternatives.

\[
\text{\textless digit\textgreater} \; ::= \; 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\]

\[
\text{\textless integer\textgreater} \; ::= \; \text{\textless digit\textgreater} \mid \text{\textless integer\textgreater}\text{\textless digit\textgreater}
\]
Development of BNF: Naur

The “Report on Algol 60,” edited by Peter Naur, appears in CACM. It uses a slightly prettier (and easier to typeset) variant of the Backus notation. Naur introduces use of ::= and |, and makes names of nonterminals identical to equivalent English phrases used in the text.

\[
\begin{align*}
\langle \text{unsigned integer} \rangle &::= \langle \text{digit} \rangle | \langle \text{unsigned integer} \rangle \langle \text{digit} angle \\
\langle \text{integer} \rangle &::= \langle \text{unsigned integer} \rangle | + \langle \text{unsigned integer} \rangle | - \langle \text{unsigned integer} \rangle
\end{align*}
\]
An Alternative: COBOL Metanotation

1960 COBOL report uses a 2-D notation. Choices are stacked vertically within braces, brackets indicate optional items, and ellipsis indicates repetition of the preceding item. *The uses of braces and brackets are documented, but the use of the ellipsis is taken for granted.*

```
FUNCTION: To subtract one or a sum of quantities from a specified quantity and store the result in the last named field or the specified one.

SUBTRACT \{literal-1 \} [\{literal-2 \} , \{field-name-2\} . . . ] FROM \{literal-n \} \{field-name-n \}

[GIVING field-name-m] [UNROUNDED]

[; ON SIZE ERROR any imperative statement]
```
1965 IBM’s PL/I specification combines BNF with COBOL metanotation.

An ellipsis indicates a nonzero number of repetitions of the preceding item; "[item] ..." indicates zero or more (not "[item ...]").
Parameterized BNF

1965 Niklaus Wirth’s PL360 used a *parameterized* form of BNF:

If in the denotations of constituents of the rule the script letters $\alpha$, $\kappa$, or $\iota$ occur more than once, they must be replaced consistently, or possibly according to further rules given in the accompanying text. As an example, the syntactic rule

$$\langle \kappa \text{ register} \rangle ::= \langle \kappa \text{ register identifier} \rangle$$

is an abbreviation for the set of rules:

- $$\langle \text{long real register} \rangle ::= \langle \text{long real register identifier} \rangle$$
- $$\langle \text{integer register} \rangle ::= \langle \text{integer register identifier} \rangle$$
- $$\langle \text{real register} \rangle ::= \langle \text{real register identifier} \rangle$$

1968 Adriaan van Wijngaarden et al. describe Algol 68 using a two-level grammar: one grammar has an infinite set of productions, which are generated by another grammar.


The BLISS language (William Wulf et al.) is described using BNF, but with a right-arrow instead of "::=". This notation is taken for granted.

```
block → begin declarations compoundexpression end
declarations → |declaration;|declarations; declaration;
compoundexpression → |e| e; compoundexpression
begin → BEGIN
end → END
```

The DEC BLISS documentation uses PL/I-style syntax descriptions.

Computer Science Department, Carnegie-Mellon University (January 15, 1970), page 1.2.  
1972 Burroughs CANDE language manual uses syntax charts only.

1974 PASCAL book uses both syntax charts and ALGOL 60–style BNF.

1978 Draft of FORTRAN 78 standard uses syntax charts plus PL/I-style BNF.

1979 The RED language (GREEN became Ada) uses syntax charts only.
Wirth Syntax Notation (WSN)

1977 Niklaus Wirth publishes ‘What can we do about the unnecessary diversity of notation for syntactic definitions?’ in CACM, solving the problem of having too many BNF variants by proposing yet another. It catches on.

```
syntax  =  {production}.
production = identifier "=" expression ".".
expression = term {"|" term}.
term     =  factor {factor}.
factor   =  identifier | literal | "(" expression ")" | "[" expression "]" | "{" expression "}".
literal  =  " " " " " " character {character} " " " " ".

Repetition is denoted by curly brackets, i.e. {a} stands for ε | a | aa | aaa | . . . . Optionality is expressed by square brackets, i.e. [a] stands for a | ε. Parentheses merely serve for grouping, e.g. (a|b)c stands for ac | bc.
```

Other BNF Variants

1976  Stanford’s SAIL language uses BNF with repeated “::= ” and no “|”.
1978  CMU Alphard project uses regular expressions in BNF with *, +, and #.
1980  Ada specification uses BNF, but with “is” for “::= ” and “or” for “|”.
1981  CMU FEG and IDL use regular expressions in BNF with *, +, and ?.
1984  C: A Reference Manual (Harbison and Steele) uses REs in BNF.
1984  Common Lisp: The Language (Steele et al.) uses REs in BNF.
1995  Python Reference Manual (Release 1.2) uses * and + in BNF, but brackets (rather than ?) for optional items.
1998  Haskell 98 Report uses BNF, with −→ for ::=, and also uses ellipsis.
1998  Ruby Language Reference Manual (1.4.6) uses * and + in “pseudo BNF” (somewhat like WSN), but brackets (rather than ?) for optional items.
C-style BNF

1978 Brian Kernighan and Dennis Ritchie publish *The C Programming Language*, which uses yet another format for grammar rules.

```
iteration-statement:
  while (expression) statement
  do statement while (expression);
  for (expression_opt; expression_opt; expression_opt) statement
```

```
assignment-operator: one of
  =  *=  /=  %=  +=  -=  <<=  >>=  &=  ^=  |=
```

1985 *The C++ Programming Language* (Bjarne Stroustrup) uses C-style BNF.

1996 *The Java Language Specification* (Gosling et al.) uses C-style BNF.

2000 *C# Language Specification* (Hejlsberg et al.) uses C-style BNF.

2012 *The F# 2.0 Language Specification* (Don Syme) uses C-style BNF but with special treatment of ellipsis (curiously defined as postfix).
We have seen a huge variety of BNF variations in the last six decades. It hasn’t been a problem.
Example of CSM Data Declarations [Again]

Expressions:
\[ e ::= x \quad \text{Variable} \]
\[ \lambda x : \tau . e \quad \text{Abstraction} \]
\[ e_1 e_2 \quad \text{Application} \]
\[ \Lambda \alpha : \kappa . e \quad \text{Type abstraction} \]
\[ e \tau \quad \text{Type application} \]

Types:
\[ \tau, \sigma, \psi, \nu ::= x \quad \text{Type variable} \]
\[ \tau_1 \rightarrow \tau_2 \quad \text{Function type} \]
\[ \forall \alpha : \kappa . \tau \quad \text{Polymorphic type} \]
\[ \tau_1 \tau_2 \quad \text{Application} \]
\[ F(\tau) \quad \text{Saturated type family} \]
\[ \kappa ::= \star \quad | \quad \kappa_1 \rightarrow \kappa_2 \quad \text{Kind} \]
\[ \Phi ::= [\alpha : \kappa] . F(\rho) \sim \sigma \quad \text{Axiom equation} \]

Adapted from Eisenberg, Vytiniotis, Peyton Jones, and Weirich, Closed Type Families with Overlapping Equations, ACM POPL 2014, Figure 2
The “Consistent Substitution” Convention

If we took the definition of BNF literally—every nonterminal can be replaced by a string derived from that nonterminal—then a sentence such as

A value of type \( \tau \) may be assigned to any variable of type \( \tau \).

could be expanded to

A value of type \( \text{int} \) may be assigned to any variable of type \( \text{bool} \).

which is nonsense. Instead, we require \textit{consistent substitution}: within a given context (\textit{other} than the RHS of a BNF rule), if a nonterminal is mentioned more than once, the same expansion must be used for each occurrence:

A value of type \( \text{int} \) may be assigned to any variable of type \( \text{int} \).
The “Decorated Nonterminals” Convention

If we took the definition of BNF literally—every nonterminal can be replaced by a string derived from that nonterminal—then a sentence such as

\[
\text{If } \tau_1 = \tau_2, \text{ then } \tau_1 <: \tau_2.
\]

would be expanded (for example, with \( \tau \rightarrow \text{int} \) to

\[
\text{If } \text{int}_1 = \text{int}_2, \text{ then } \text{int}_1 <: \text{int}_2.
\]

which is nonsense. Instead, we recognize a decorated nonterminal as being a distinct nonterminal having the same productions as the undecorated form:

\[
\text{If } \text{int} = \text{bool}, \text{ then } \text{int} <: \text{bool}.
\]

\[
\text{If } \text{int} = \text{int}, \text{ then } \text{int} <: \text{int}.
\]
SUBSTITUTION
Substitution Notation

1932 Alonzo Church uses the notation $S^X_Y U$ for substitution in a formula:

We assume an understanding of the operation of substituting a given symbol or formula for a particular occurrence of a given symbol or formula.

And we assume also an understanding of the operation of substitution throughout a given formula, and this operation we indicate by an $S, S^X_Y U$ representing the formula which results when we operate on the formula $U$ by replacing $X$ by $Y$ throughout, where $Y$ may be any symbol or formula but $X$ must be a single symbol, not a combination of several symbols.

Note: the variable to be substituted for is on top, and the replacing term is on the bottom!

1941 Alonzo Church publishes *The Calculi of Lambda-Conversion*.


Nowadays we write

\[ e[v/x] \]

(or something like it)

for the result of substituting \( v \) for \( x \) in \( e \).

How many variations are there?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$e\left</td>
<td>v\right</td>
<td>\ x$</td>
</tr>
<tr>
<td>$e\left</td>
<td>v\right</td>
<td>\ x$</td>
</tr>
<tr>
<td>$[v/x]e$</td>
<td>67</td>
<td>2</td>
</tr>
<tr>
<td>$[v/x]e$</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$[x := v]e$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$[x \rightarrow v]e$</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>$[x \leftarrow v]e$</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>$[x := v]$</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>$[x \leftarrow v]$</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>$[x \rightarrow v]$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$[x \leftarrow v]$</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>${v/x}$</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>${v/x}$</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>


Most popular during 1973–2016 are highlighted. Usage has grown over time; substitution used in over 1/3 of POPL papers 2012–2016.
Substitution: **POPL 2012–2016 and Others 2014–2016**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$e \mid v$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$e \frac{v}{x}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$[v/x]e$</td>
<td>67</td>
<td>17</td>
</tr>
<tr>
<td>$[v/x]e$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$[x := v]e$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$[x \rightarrow v]e$</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>$[x \rightarrow v]e$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$[[v/x]]e$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>${v/x}e$</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>${x \rightarrow v}e$</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$[v/x]e$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$e[v/x]$</td>
<td>133</td>
<td>37</td>
</tr>
<tr>
<td>$e[v/x]$</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$e(v/x)$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$e{v/x}$</td>
<td>25</td>
<td>7</td>
</tr>
<tr>
<td>$e{v/x}$</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$e{v/x}$</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>$e{x \leftarrow v}$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$e{x \rightarrow v}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$e{x \rightarrow v}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$e{v/x}}$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$e{x \leftarrow v}}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$e{x := v}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$e{x := v}$</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Of these 31, 15 were used at POPL in the last 5 years; 5 more were used at other SIGPLAN conferences in the last 3 years.
Substitution: A Moderate Problem

By far the most popular form is

\[ e[v/x] \]

but about every once every five years we see

\[ e[x/v] \]

which gets it backwards.

You can’t count on the variable names to tip you off, because different authors use different names.

One paper published in the last year used both forms.
The forms $e[x \mapsto v]$ and $e[x := v]$ (and variants that are prefix and/or use braces) are frequently used for substitution (about 1/6 of all POPL papers).

But they are also widely used for another purpose: function update (also called map update and storage update)!

$$(f[x \mapsto v])(z) = \text{if } z = x \text{ then } v \text{ else } f(z)$$

Use of both in one paper can make it very hard to read.

And lately most authors are taking all these notations for granted.
Substitution: **My Recommendations**

- Use postfix forms (clearly more popular).
- Use either / (most popular) or → (arguably clearer). *Never* use ←.
- If you use /, do *not* make names smaller (it only makes them less readable).
- Reserve ↦→ for function/map update and := for storage/heap update.
- Use brackets [ ] for operators; use braces { } for collections.

<table>
<thead>
<tr>
<th>applications</th>
<th>operators</th>
<th>(singleton) collections</th>
</tr>
</thead>
<tbody>
<tr>
<td>substitution</td>
<td>e[v/x]</td>
<td>a substitution σ = [v/x]</td>
</tr>
<tr>
<td>substitution</td>
<td>e[x → v]</td>
<td>a substitution σ = [x → v]</td>
</tr>
<tr>
<td>map update</td>
<td>Γ[x ↦ v]</td>
<td>a map u = [x ↦ v]</td>
</tr>
<tr>
<td>heap update</td>
<td>H[x := v]</td>
<td>a heap H = {x := v}</td>
</tr>
</tbody>
</table>

- Mnemonic for e[v/x]: “Within e, v *supersedes* (‘sits over’) x.”
- Mnemonic for e[x → v]: “Within e, x *becomes* v.”
Nicolas Chuquet uses an *underline* for mathematical grouping.

Christoff Rudolff uses the sign $\sqrt{}$ to indicate taking a square root, and also uses dots to indicate grouping: $\sqrt{12 + \sqrt{140}}$ means $\sqrt{12 + \sqrt{140}}$.

Niccolò Tartaglia uses parentheses ( ) for mathematical grouping.

William Oughtred uses double dots : to indicate grouping.

Thomas Harriot uses a long overbrace with $\sqrt{}$ for grouping.

René Descartes attaches an *overline* to $\sqrt{}$, producing $\overline{\sqrt{}_{12 + \sqrt{140}}}$.

Jan Stampioen uses all three together: ”$\sqrt{\overline{(aaa + 6aab + 9bba)}}$“.

Frans van Schooten, editing Vieta’s works, uses overline for grouping.

Gottfried Leibniz begins using parentheses in preference to overline.

*Acta eruditorum* officially adopts the Leibnizian symbolism.

Pierre Louis Maupertuis uses square brackets [ ].
Three notations for grouping duking it out for five centuries!
A Little Bit More about **Vectors**

1813 Jean-Robert Argand graphs complex numbers, speaks of $i = \sqrt{-1}$ as a rotation in the plane, and proposes the notation $\overrightarrow{ab}$ for vectors.

1833 William Rowan Hamilton recasts the theory of complex numbers as an algebra on pairs of reals $(a_1, a_2)$.

1833–43 Hamilton seeks an algebra for triplets and polyplets (that is, tuples).

1843 Hamilton discovers the quaternions $a + bi + cj + dk$.

1844–46 Hamilton reformulates quaternions without $ijk$ coordinates, describing a quaternion as the sum of a scalar and an (imaginary) vector.

1873 James Maxwell uses quaternions to describe electricity and magnetism.

1881 Josiah Willard Gibbs establishes $\cdot$ and $\times$ for dot and cross product.

1882 Oliver Heaviside advocates ditching scalars and simply using vectors.

1890–94 Big fight between “quaternionists” and “vectorists” in physics!
1975–1981 Both $\vec{a}$ and $\overline{a}$ are used to denote a vector, list, sequence, or set that is *enclosed*: $\vec{a} = \langle a_1, a_2, \ldots, a_m \rangle$ or $\overline{x} = \{x_1, \ldots, x_k\}$.

1978 One paper defines $\overline{x} : \overline{T}$ to be a sequence of variable declarations.

1981 For the first time at POPL, overline notation is *taken for granted*.

1989 For the first time at POPL, $\vec{X}$ indicates an *unenclosed sequence*.

So far, the *semantic model* is that an overline marks a variable as representing a vector or sequence, and the obvious *syntactic model* is that you can make copies of the overlined variable name and attach sequential subscripts starting from 1. (These copies may be enclosed and may be comma-separated.)
1990 First explicit claim that the elements may be *metasyntactic variables*: 
“we use the notation $\overline{\chi}$, for some metasyntactic variable $\chi$ to stand for some finite, comma-separated list of the form $(\chi_1, \ldots, \chi_n)$.”

1990 First use of an implicit unit of replication: “If $\overline{m} = m_1 \ldots m_k$ and $\overline{\sigma} = \sigma_1 \ldots \sigma_k$, we write $\overline{m} : \overline{\sigma}$ for $m_1 : \sigma_1 \ldots, m_k : \sigma_k$”

1993 First claim that overline may apply to *any syntactic object*: 
“a list of syntactic objects $s_1, \ldots, s_n$ is abbreviated by $\overline{s_n}$. For instance, $\forall \alpha_n : \overline{\sigma_n}.\sigma$ is equivalent with $\forall \alpha_1 : \sigma_2, \ldots, \alpha_n : \sigma_n.\sigma$.”

1994 First use of overline on a syntactic fragment containing an operator (in this case, a semicolon): “Let $\overline{c_1}, \overline{c_2}, \overline{d_1}$, etc., be tuples of coercions. Then $\hat{\rho}(\overline{c_1}; \overline{c_2}, \overline{d_1}; d_2) = \hat{\rho}(\overline{c_1}, \overline{c_2}); \hat{\rho}(\overline{d_1}, d_2)$.”
Problem: Unit of Replication versus Subscript Attachment

We have already seen “\( \overline{m} : \overline{\sigma} \)” used for “\( m_1 : \sigma_1, \ldots, m_k : \sigma_k \)”.

(Later we see “\( \overline{T x} \)” for “\( T_1 \ x_1, \ldots, T_n \ x_n \)” when describing Java-like languages.) This raises a question: in a general and purely syntactic model of overline notation, just how large is the implicit unit of syntactic replication?

Others have written “\( \overline{m : \sigma} \)” for “\( m_1 : \sigma_1 \ldots, m_k : \sigma_k \)”. Now the unit of syntactic replication is clear: it is exactly everything covered by one overline. But this raises a different question: where should subscripts be attached?

Why is the result “\( m_1 : \sigma_1, \ldots, m_k : \sigma_k \)” rather than “\( m : \sigma_1, \ldots, m : \sigma_k \)” or “\( m_1 : 1 \sigma_1, \ldots, m_k : 1 \sigma_k \)”?

(It’s easy to come up with reasons; but so far no one has stated them!)
1994  First use of nested overlines.

1996  First explicit definition of $\vec{a}$ as an *unenclosed* comma-separated list.

1996  Overline notation taken for granted, but first explicit statement of the “equal-length convention”: “We implicitly assume in $[\vec{z}/\vec{y}]$ that the sequence $\vec{y}$ is linear and of the same length as $\vec{z}$.”

1996  First use of tilde for repetition: “sequences of types are written $\tilde{T}$ instead of $T_1, \ldots, T_n$.” First mention of using an adjacent comma to *concatenate* overlined things: “Type environments are extended with bindings for new variables writing $\Gamma, x : T$ or $\Gamma, \tilde{x} : \tilde{T}$.”

1997  Also uses tilde. First statement of general *pointwise extension*: “By abuse of notation, operations on singletons are implicitly extended pointwise to sequences.” But immediately we run into a problem!
What is the meaning of $\Gamma(\tilde{b}) = \tilde{T}/\tilde{X}\tilde{P}$?

If we regard substitution “$[\cdot/\cdot]$” and equality “$\cdot = \cdot$” as operations on singletons, we can certainly extend them pointwise. Therefore we can replicate the entire equation, so that $\Gamma(\tilde{b}) = [\tilde{T}/\tilde{X}]\tilde{P}$ stands for this conjunction of assertions:

$$\Gamma(b_1) = [T_1/X_1]P_1 \quad \text{and} \quad \ldots \quad \text{and} \quad \Gamma(b_n) = [T_n/X_n]P_n$$

But semantic analysis of the rest of the paper indicates that the authors really wanted $\Gamma(\tilde{b}) = [\tilde{T}/\tilde{X}]\tilde{P}$ to stand for a different conjunction of assertions:

$$\Gamma(b_1) = [T_1/X_1, \ldots, T_m/x_m]P_1$$
$$\text{and} \quad \ldots$$
$$\text{and} \quad \Gamma(b_n) = [T_1/X_1, \ldots, T_m/x_m]P_n$$
A Solution? Nested Overlines (1 of 2)

Instead of $p = [v/x]q$, some authors write $\overline{p} = [\overline{v}/\overline{x}]\overline{q}$.

Superficially, this seems natural. But how do we know that this means

$p_1 = [v_1/x_1, \ldots, v_m/x_m]q_1$

and \ldots

and $p_n = [v_1/x_1, \ldots, v_m/x_m]q_n$

and not something like

$p_1 = [v_{11}/x_{11}, \ldots, v_{1m}/x_{1m}]q_1$

and \ldots

and $p_n = [v_{n1}/x_{n1}, \ldots, v_{nm}/x_{nm}]q_n$

where $v$ and $x$ are one-dimensional

and not something like

$p_1 = [v_{11}/x_{11}, \ldots, v_{1m}/x_{1m}]q_1$

and \ldots

and $p_n = [v_{n1}/x_{n1}, \ldots, v_{nm}/x_{nm}]q_n$

where $v$ and $x$ are two-dimensional
A Solution? Nested Overlines (2 of 2)

Even without nesting, some authors write \( \Gamma \vdash x : \tau \), intending

\[
\Gamma \vdash x_1 : \tau_1 \quad \Gamma \vdash x_2 : \tau_2 \quad \ldots \quad \Gamma \vdash x_n : \tau_n
\]

How do we know it isn’t supposed to be

\[
\Gamma_1 \vdash x_1 : \tau_1 \quad \Gamma_2 \vdash x_2 : \tau_2 \quad \ldots \quad \Gamma_n \vdash x_n : \tau_n
\]

And we would have the same problem with \( \Gamma(b) = [T/X]P \):

why should \( b \) and \( T \) and \( X \) and \( P \) get subscripts, but not \( \Gamma \)?

It is possible to do a global dimensional analysis, but it’s difficult, especially when the language typically does not contain explicit declarations of vector variables. (And this is a semantic analysis.)
The Essential Contradiction

In about the last 15 years, we have found that we want *both* of these usages:

We want \( p = \left[ \frac{v}{x} \right] q \) to mean

\[
p_1 = \left[ \frac{v_1}{X_1}, \ldots, \frac{v_m}{x_m} \right] q_1
\]

and \( \ldots \)

and \( p_n = \left[ \frac{v_1}{X_1}, \ldots, \frac{v_m}{x_m} \right] q_n \)

but we want *case e of* \( K \bar{y} \rightarrow e' \) to mean

*case e of*

\[
K_1 \, y_{11} \ldots \, y_{1m_1} \rightarrow e'_1
\]

\[
\ldots
\]

\[
K_n \, y_{n1} \ldots \, y_{nm_n} \rightarrow e'_n
\]

where each case clause may have *different* \( y \) variables and indeed a *different number of* \( y \) variables

With a purely syntactic theory, we can’t have it both ways.
What Do We Want From Overline Notation? (1 of 2)

- \text{str} can expand to any number of copies of \text{str}.
  - More concise than ellipsis notation.
  - Question: whether and how copies are separated (comma by default?).
  - If we want “\overline{x}, \overline{y}” for concatenation, sequences should be unenclosed.

- Each copy of \text{str} may be expanded \textit{differently}.
  - BNF nonterminals may be expanded differently in each copy.
  - Nested overlines may be expanded differently in each copy.
    \begin{itemize}
    \item This suggests that nested overlines should be processed outside-in.
    \end{itemize}

- \textit{But}, if \text{str} is mentioned more than once in a given context (such as an inference rule or a text sentence or paragraph), the expansion of each occurrence must be the \textit{same} (similar to treatment of BNF nonterminals).
What Do We Want From Overline Notation? (2 of 2)

- *Within each copy* of $str$, multiple occurrences of the *same* BNF nonterminal must be expanded in the *same way* (as usual).
- If a variable $v$ occurs within $str$, copy $i$ of the $str$ must refer to $v_i$.
- All variables occurring in $str$ must have the *same length*.

A formal theory of overline expansion must track two kinds of constraints:

- Requirements for identical expansion.
- Requirements that variables be the same length.

Various constraints suggest that:

- Overlines should be expanded *before* BNF nonterminals.
- Substitutions should be expanded *after* BNF nonterminals.
Solving the **Essential Contradiction**

We propose to borrow an idea from *quasiquoting*:

- ‘(lambda (,vars) ,body) means “make a copy of the S-expression (lambda (,vars) ,body), but a comma means ‘except here’: the value of the expression following the comma is used”.

This idea was also used for parallelism in Connection Machine Lisp (1986):

- α(+ (* 9/5 ⋅temps) 32) means “evaluate many copies of the expression (+ (* 9/5 ⋅temps) 32), but a bullet means ‘except here’: the value of the expression following the bullet is a vector, so please use a different vector element in each copy”.

---


*Connection Machine Lisp: Fine-grained Parallel Symbolic Processing.*

Adding **Underlines** to Overlines

We propose this modification to overline notation in CSM:

- \( \overline{\text{str}} \) can expand to any number of copies of \( \text{str} \), and each copy of \( \text{str} \) may be expanded *differently*, but an underline means “except here”: underlined portions of \( \text{str} \) must be expanded the *same* way in each copy.

Therefore for our examples we can write:

\[
\begin{align*}
\overline{p} & = \overline{[v/x]q} & \text{same substitution in each outer copy} \\
\text{case} \ e \ \text{of} \ \overline{K \ y \ → \ e'} & \quad \text{as before} \\
\overline{\Gamma(b)} & = \overline{[T/X]P} & \text{same \( \Gamma \) and same substitution in each outer copy} \\
\overline{\Gamma \ ⊢ \ x : \tau} & \quad \text{same \( \Gamma \) in each copy}
\end{align*}
\]

The dimensionality of each variable is simply the number of overlines minus the number of underlines.
A Simple (?) Formal Model for Overline Expansion

The solution is to *integrate* the old “subscript attachment” model; the usual rules for BNF nonterminals will then enforce the necessary same-expansion constraints. (Length constraints must still be tracked separately.)

To expand an outermost overline:

- Freely choose an integer length $n$.
- Replaced the overlined string with $n$ copies of the string.
- In copy $k$ ($1 \leq k \leq n$), for every (possibly already decorated) single letter or BNF nonterminal that is *not underlined*, attach $k$ as a subscript.
  - Record the fact that all items to which subscripts are attached are constrained to have the same length.
- In each copy, from any underlined material remove just one underline.
- Now perform expansions in the replacement material.
A Simple Formal Model for Context Expansion

To expand an entire context (inference rule, right-hand side of a BNF rule, or text sentence or paragraph):

- Repeatedly expand outermost overlines until none are left.
- Expand all BNF nonterminals, obeying the same-expansion and decorated-nonterminal rules.
- Expand all substitution notations.

The expansion of an entire context is valid only if the various “free choices” for overline lengths have been made so that all length constraints are satisfied.
What about Cases Simple Overlines Can’t Handle?

Notations such as \( \overline{m_n} : \sigma_n \) (where \( n \) is a globally defined length) or \( \overline{m_i} : \sigma_i \) (where \( i \) is an implicitly bound index variable) clearly identify subscript attachment points, but do not extend well to nesting:

\[
\Gamma(b_i) = \left[ T_j/X_j \right] P_i
\]

It takes some analysis to match the indices to the overlines. Not so good.

Other writers explicitly mark the binding points:

\[
\Gamma(b_i) = \left[ T_j/X_j \right] P_i^i
\]

and some even explicitly specify ranges:

\[
\Gamma(b_i) = \left[ T_j/X_j \right]_{1 \leq j \leq m}^{0 \leq i < n} P_i
\]

I endorse these latter two explicit-binding overline notations for difficult cases.
ELLIPSIS
The **Ellipsis** (Dot Dot Dot)

Most readers will have encountered the *dotdotdot* notation already. It is a notation that is rarely introduced properly; mostly, it is just used without explanation as in, for example, ‘$1 + 2 + \cdots + 20 = 210$’

In the Past, We Have Used Ellipsis to Explain Overline

“\( \overline{x} \)” means “\( x_1, \ldots, x_n \)”

But what does “\( x_1, \ldots, x_n \)” mean?

(Or “\( x_1, x_2, \ldots \)”? Or “\( e_1, \ldots, e_i, \ldots, e_n \)”?)

We propose to explain the ellipsis notation by providing a formal transformation to overline notation (whose formal definition need not rely on ellipses).
The Basic Idea

- Predefine a set of standard *usage patterns* to be supported.
- For each use of ellipsis, expansion must identify a matching usage pattern.
- Each pattern includes (a) one or more ellipses, (b) some number of copies of a *separator string*, and (c) *matchable strings*.
- Use unification-like matching on the matchable strings to find a *common structure* parameterized by one variable (an integer index) and a set of unifying *substitutions* for that variable.
- Construct an overline notation using one copy of the common structure and one copy of the separator string, and use the substitution expressions to specify the range and/or verify constraints.
Examples

Example 1: \[x_0, \ldots, x_{n-1}\] :
the separator is ";",
the matchable strings are \(x_0\) and \(x_{n-1}\);
the common structure is \(x_i\) with substitutions \([0/i]\) and \([n-1/i]\);
the result is \(x_i; 0 \leq i \leq n-1\).

Example 2: \(a_1b_1 \oplus a_2b_2 \oplus \ldots\) :
the separator is \(\oplus\);
the matchable strings are \(a_1b_1\) and \(a_2b_2\);
the common structure is \(a_ib_i\) with substitutions \([1/i]\) and \([2/i]\);
the pattern requires verification that \(1\) and \(2\) are consecutive integers;
and the result is \(a_ib_i \oplus i\)
(the underbracket indicates that \(\oplus\) is the separator).
Conclusions

- Computer Science Metanotation is a symbolic programming language with its own distinctive syntax, semantics, and idioms.
- CSM should be an explicit object of study in our community.
- CSM is a living language and has changed over the last four decades (and some of its notational ideas go back centuries).
- We now have problems with substitution and overlines. These can be fixed.
- We should develop a complete formal theory of the language, including overline notation and ellipsis notation (including nested cases) and their interaction with BNF and substitution. I have made a start.
- We should apply the techniques developed for other languages to CSM to build interpreters, compilers, IDEs, correctness checkers, and other tools.
- There are interesting opportunities for parallel execution of CSM and the use of parallel algorithms in associated tools.
Questions?

Comments?