ORACLE®

It’s Time for a New Old Language

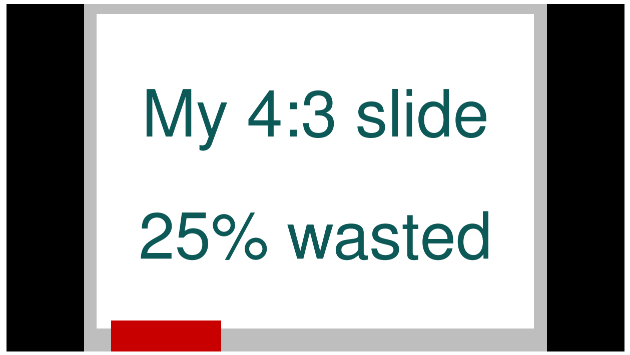
Guy L. Steele Jr.
Software Architect, Oracle Labs

PPoPP Keynote
Monday, February 6, 2017

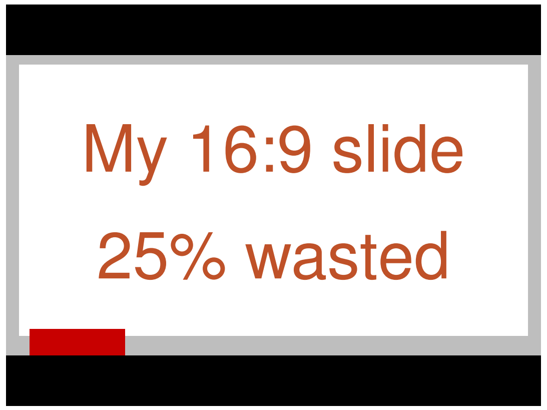
We Begin with a Digression

Why do these slides have a strange aspect ratio?

If my slides are 4:3 but the projector is 16:9, 25% of the screen is wasted:



And if my slides are 16:9 but the projector is 4:3, 25% of the screen is wasted:

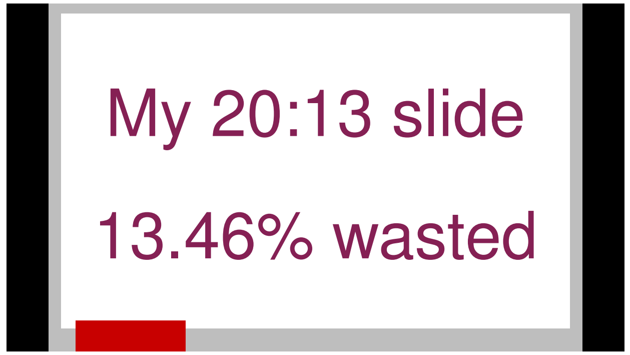


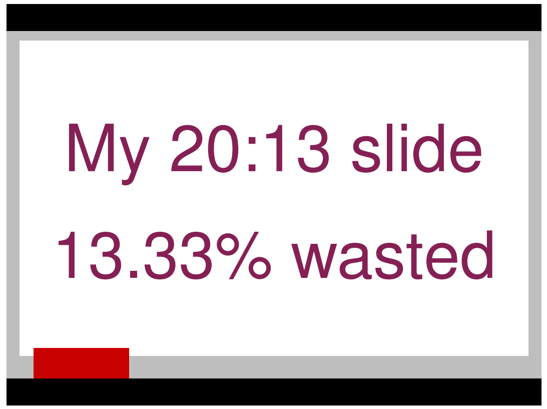
These Slides Have a 20:13 Aspect Ratio

The *optimal* compromise ratio is $8{:}3\sqrt{3} = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{5 + \cfrac{1}{1 + \cfrac{1}{4 + \cdots}}}}}$.

Sucessive truncations of this continued fraction produce approximants 1:1, 2:1, 3:2, 17:11, 20:13, 97:63, . . .

Using 20:13 with either 16:9 or 4:3 projection, less than 13.5% is wasted:





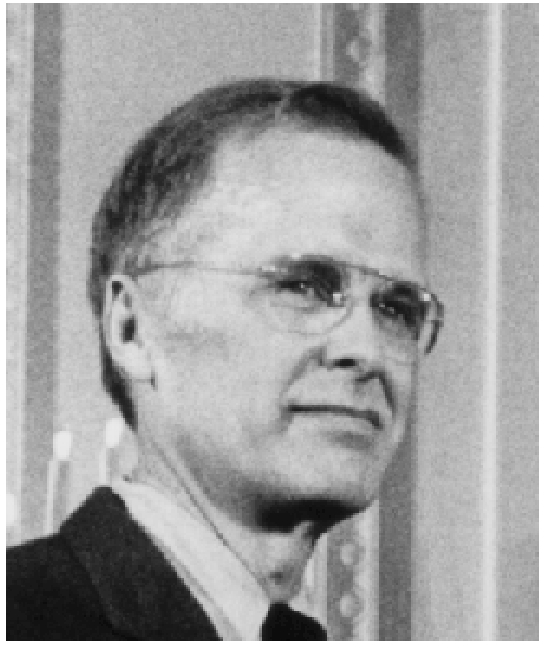
You may want to give this a try. If you can't be bothered with 20:13, try 3:2—at least until 16:9 projectors become ubiquitous.

(End of digression.)

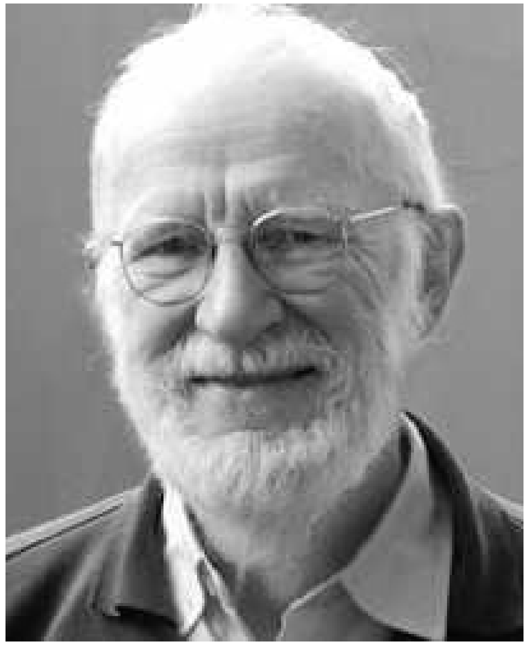
The most popular programming language in computer science

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 / ADT / symbolic expression (BNF)
- 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
- Special operation: capture-free substitution within a symbolic expression

Example of CSM Data Declarations (BNF)

Expressions:

$$\begin{array}{lcll}
e & ::= & x & \text{Variable} \\
 & | & \lambda x : \tau . e & \text{Abstraction} \\
 & | & e_1\ e_2 & \text{Application} \\
 & | & \Lambda \alpha : \kappa . e & \text{Type abstraction} \\
 & | & e\ \tau & \text{Type application}
\end{array}$$

Types:

$$\begin{array}{lcll}
\tau, \sigma, \psi, \upsilon & ::= & x & \text{Type variable} \\
 & | & \tau_1 \rightarrow \tau_2 & \text{Function type} \\
 & | & \forall \alpha : \kappa . \tau & \text{Polymorphic type} \\
 & | & \tau_1\ \tau_2 & \text{Application} \\
 & | & F(\overline{\tau}) & \text{Saturated type family} \\
 & & & \\
\kappa & ::= & \star \mid \kappa_1 \rightarrow \kappa_2 & \text{Kind} \\
 & & & \\
\Phi & ::= & [\overline{\alpha{:}\kappa}].\ F(\overline{\rho}) \sim \sigma & \text{Axiom equation}
\end{array}$$

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

Example of CSM Code (Nondeterministic?) (1 of 2)

$\boxed{\text{no\_conflict}(\Psi, i, \overline{\tau}, j)}$ Check for equation conflicts

$$\frac{\Psi = \overline{[\overline{\alpha{:}\kappa}].\ F(\overline{\rho}) \sim \upsilon} \qquad \text{apart}\left(\overline{\rho_j}, \overline{\rho_i[\overline{\tau/\alpha_i}]}\right)}{\text{no\_conflict}(\Psi, i, \overline{\tau}, j)} \ \text{[NC\_APART]}$$

$$\frac{\text{compat}(\Psi[i], \Psi[j])}{\text{no\_conflict}(\Psi, i, \overline{\tau}, j)} \ \text{[NC\_COMPATIBLE]}$$

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)

$\boxed{\text{no\_conflict}(\Psi, i, \overline{\tau}, j)}$ Check for equation conflicts

$$\frac{\Psi = \overline{[\overline{\alpha{:}\kappa}].\ F(\overline{\rho}) \sim \upsilon} \qquad \text{apart}(\overline{\rho_j}, \overline{\rho_i[\overline{\tau/\alpha_i}]})}{\text{no\_conflict}(\Psi, i, \overline{\tau}, j)} \ \text{[NC\_APART]}$$

$$\frac{\text{compat}(\Psi[i], \Psi[j])}{\text{no\_conflict}(\Psi, i, \overline{\tau}, j)} \ \text{[NC\_COMPATIBLE]}$$

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)

$$\frac{}{\Gamma \vdash x_i : \tau_i} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M N : \tau}$$

$$\frac{\Gamma \vdash M_i : \tau \quad \big(i = 1, \ldots, \mathrm{ar}(\mathtt{op})\big)}{\Gamma \vdash \mathtt{op}\big(M_1, \ldots, M_{\mathrm{ar}(\mathtt{op})}\big) : \tau}$$

$$\frac{\Gamma \vdash M : \tau \times \sigma}{\Gamma \vdash \mathtt{fst}(M) : \tau} \qquad \frac{\Gamma \vdash M : \tau \times \sigma}{\Gamma \vdash \mathtt{snd}(M) : \sigma} \qquad \frac{\Gamma \vdash M : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \langle M, N \rangle : \tau \times \sigma}$$

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)

$$\frac{}{\Gamma \vdash x_i : \tau_i} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M N : \tau}$$

input output

$$\frac{\Gamma \vdash M_i : \tau \quad \big(i = 1, \ldots, \mathrm{ar}(\mathtt{op})\big)}{\Gamma \vdash \mathtt{op}\big(M_1, \ldots, M_{\mathrm{ar}(\mathtt{op})}\big) : \tau}$$

$$\frac{\Gamma \vdash M : \tau \times \sigma}{\Gamma \vdash \mathtt{fst}(M) : \tau} \qquad \frac{\Gamma \vdash M : \tau \times \sigma}{\Gamma \vdash \mathtt{snd}(M) : \sigma} \qquad \frac{\Gamma \vdash M : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \langle M, N \rangle : \tau \times \sigma}$$

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)

$$\frac{}{\Gamma \vdash x_i : \tau_i} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M N : \tau}$$

iterator

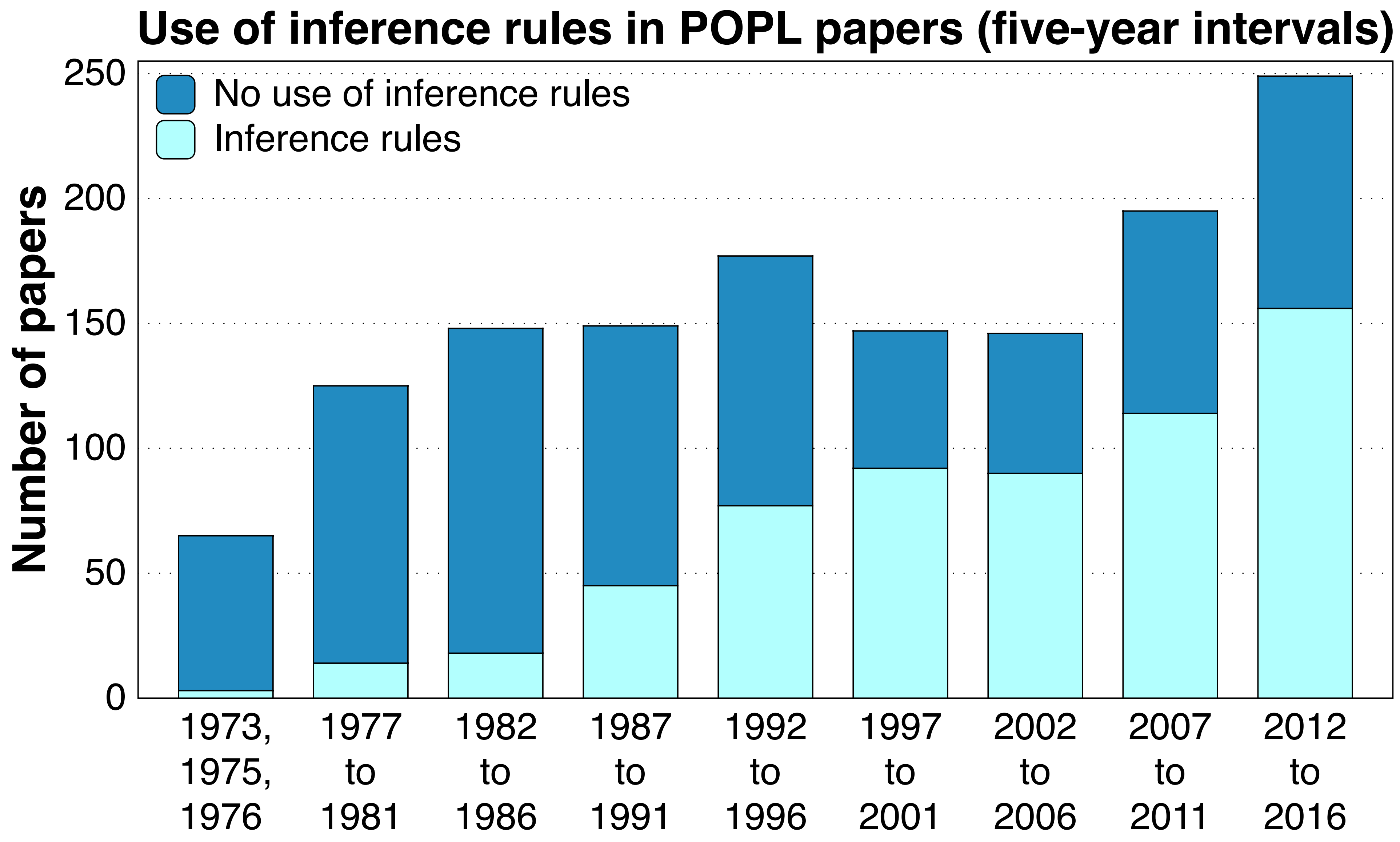
$$\frac{\Gamma \vdash M_i : \tau \quad \left(i = 1, \ldots, \mathrm{ar}(\mathtt{op}) \right)}{\Gamma \vdash \mathtt{op}\left(M_1, \ldots, M_{\mathrm{ar}(\mathtt{op})} \right) : \tau}$$

sequence

$$\frac{\Gamma \vdash M : \tau \times \sigma}{\Gamma \vdash \mathtt{fst}(M) : \tau} \qquad \frac{\Gamma \vdash M : \tau \times \sigma}{\Gamma \vdash \mathtt{snd}(M) : \sigma} \qquad \frac{\Gamma \vdash M : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \langle M, N \rangle : \tau \times \sigma}$$

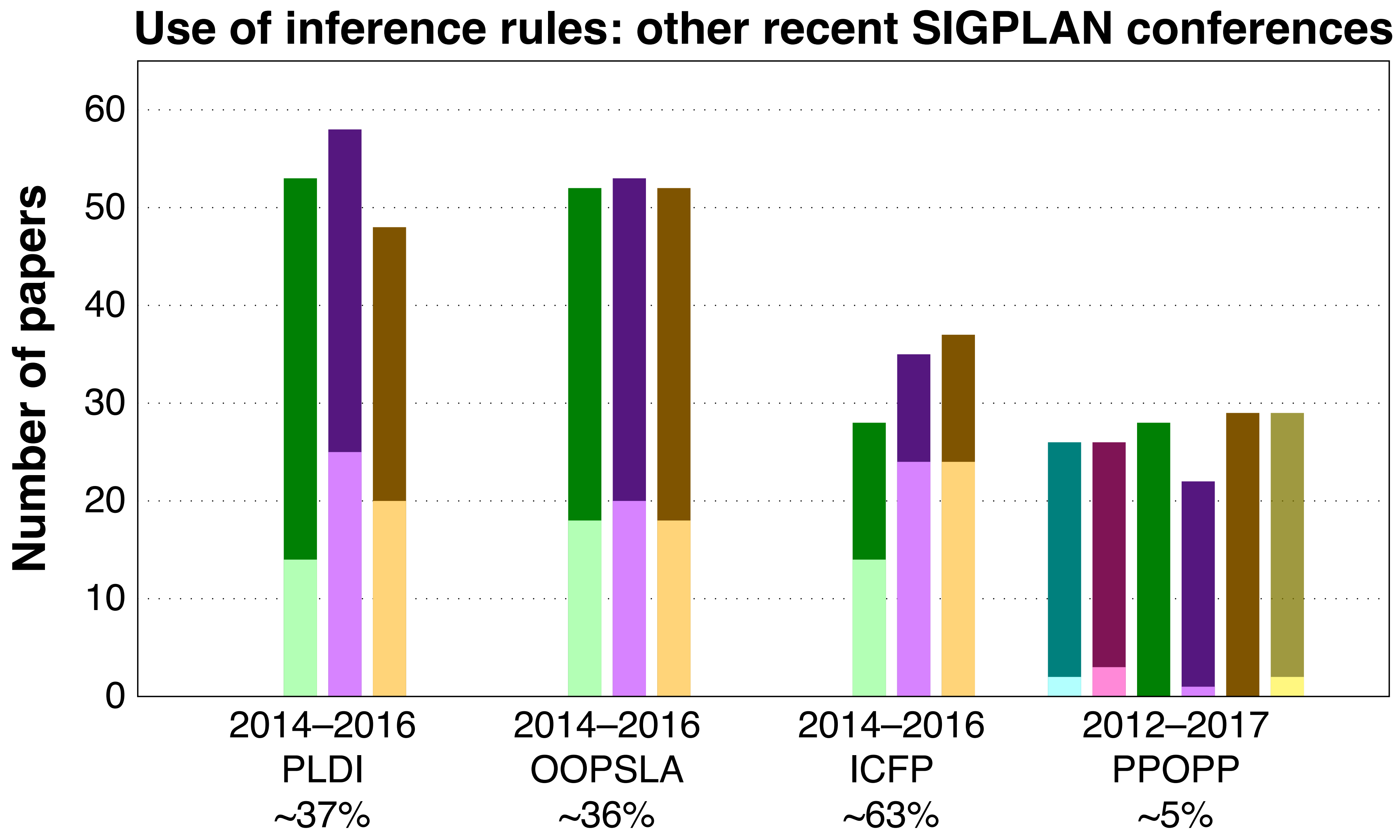
Adapted from Muroya, Hoshino, and Hasuo, *Memoryful Geometry of Interaction II*, ACM POPL 2016, Figure 1

Popularity of Computer Science Metanotation (1 of 2)



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

Popularity of Computer Science Metanotation (2 of 2)



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

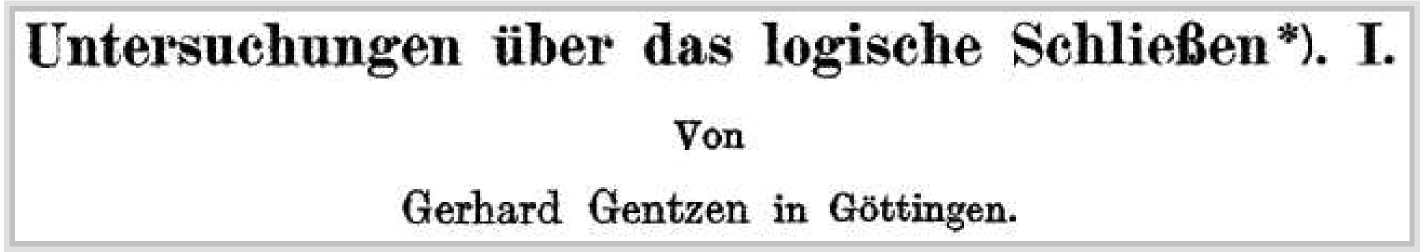
Structure of This Talk

- 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
  - In some cases I propose possible solutions
- Discuss how to formalize it (and eventually mechanize it)

INFERENCE RULES

Gentzen Notation (Natural Deduction)

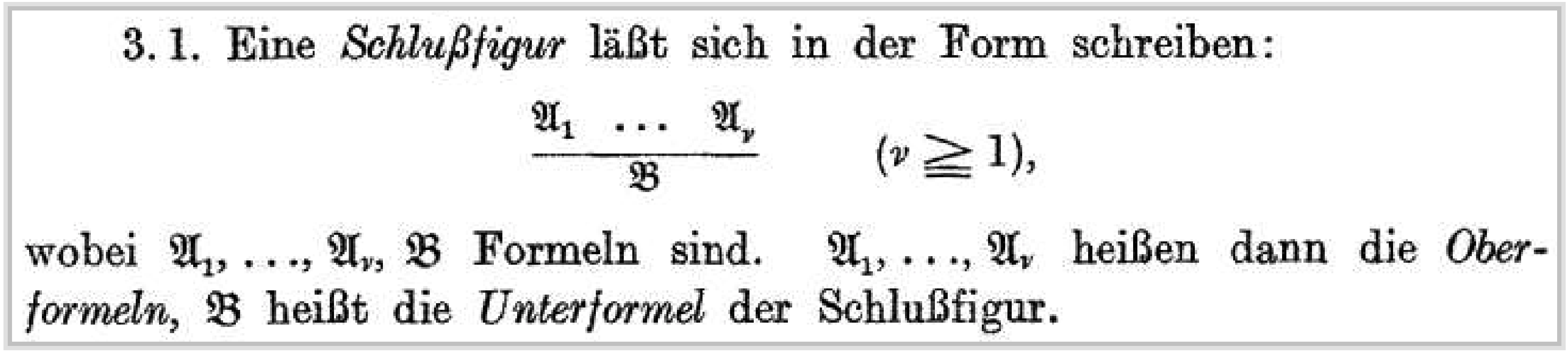
1935 Gerhard Gentzen creates a rule notation for *natural deduction*:



Untersuchungen über das logische Schließen*). I.

Von

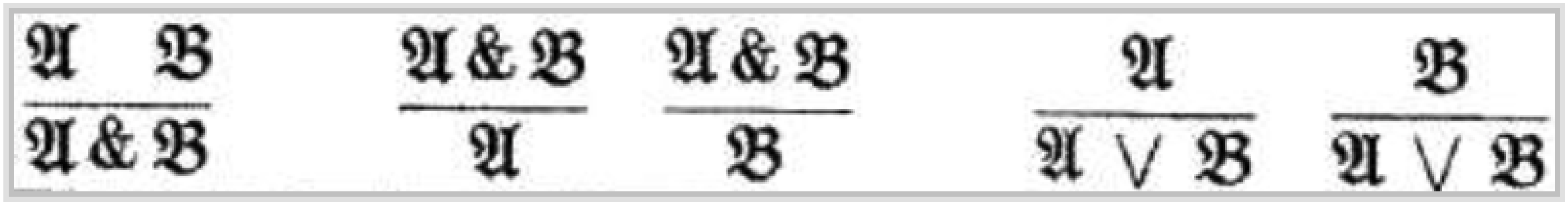
Gerhard Gentzen in Göttingen.



3. 1. Eine *Schlußfigur* läßt sich in der Form schreiben:

$$\frac{\mathfrak{A}_1 \quad \ldots \quad \mathfrak{A}_\nu}{\mathfrak{B}} \qquad (\nu \geqq 1),$$

wobei $\mathfrak{A}_1, \ldots, \mathfrak{A}_\nu$, $\mathfrak{B}$ Formeln sind. $\mathfrak{A}_1, \ldots, \mathfrak{A}_\nu$ heißen dann die *Oberformeln*, $\mathfrak{B}$ heißt die *Unterformel* der Schlußfigur.



$$\frac{\mathfrak{A} \quad \mathfrak{B}}{\mathfrak{A} \,\&\, \mathfrak{B}} \qquad \frac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{A}} \qquad \frac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{B}} \qquad \frac{\mathfrak{A}}{\mathfrak{A} \vee \mathfrak{B}} \qquad \frac{\mathfrak{B}}{\mathfrak{A} \vee \mathfrak{B}}$$

Gerhard Gentzen. Untersuchungen über das logische Schließen I.
Mathematische Zeitschrift 39, 1 (1935), 176–210.

Today’s Computer Science Inference Rule Notation

$$\frac{\begin{array}{c} premise \quad premise \quad premise \\ premise \quad premise \end{array}}{conclusion}\;[\textsc{Optional Label}]$$

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”.

History of Regular Expressions in One Slide

1951 Stephen Kleene develops regular expressions to describe McCulloch-Pitts (**1943**) nerve nets (uses $\vee$ for choice; considers postfix *, but decides to make it a *binary* operator to avoid empty strings).

1956 Journal publication of Kleene's technical report: binary * only.

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

1962 Janusz Brzozowki uses binary $+$ for $\vee$ and introduces postfix $^+$.

1968 Ken Thompson's paper "Regular Expression Search Algorithm" uses $|$.

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

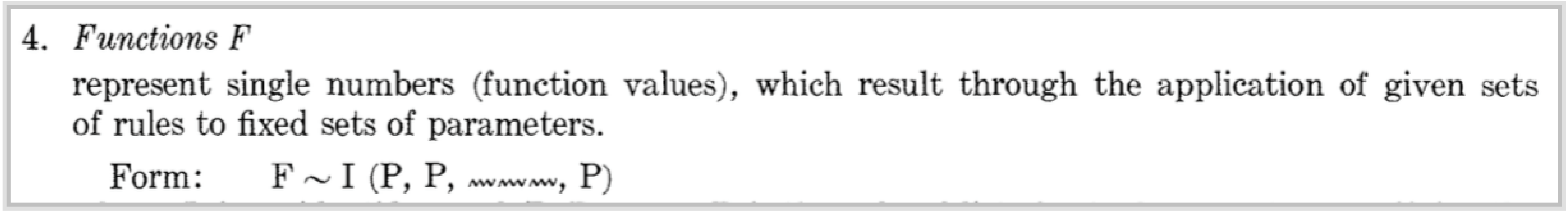
1975 Alfred Aho creates `egrep` (includes `( )`, `|`, `*`, `+`, `?`).

1978 CMU Alphard project uses regular expressions with `*`, `+`, and `#`.

1981 CMU FEG and IDL use regular expressions with `*`, `+`, and `?`.

Development of BNF: Perlis and Samelson

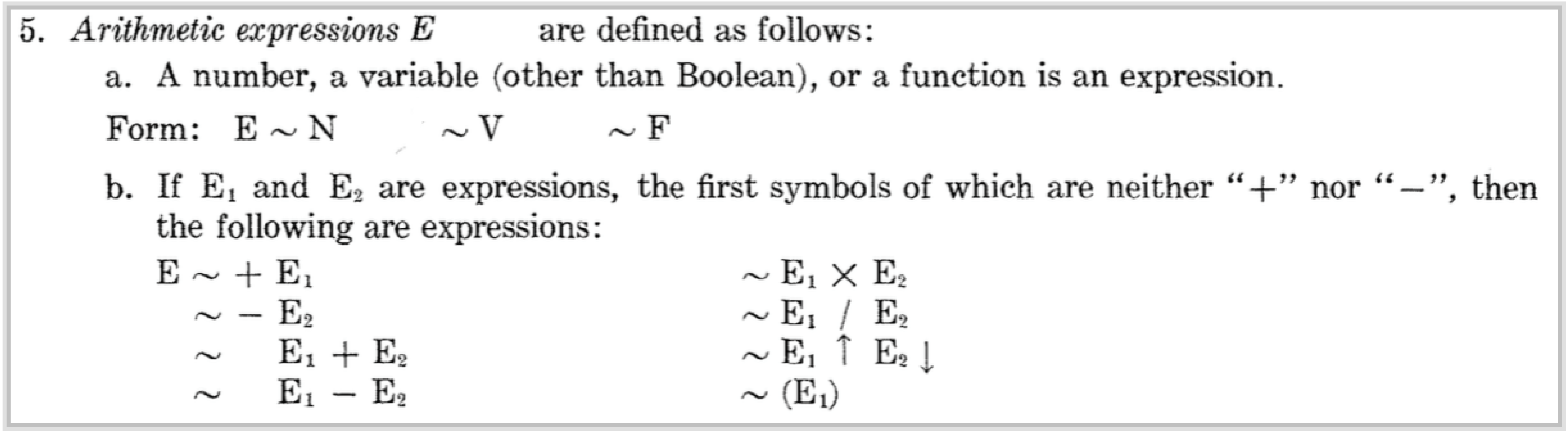
1958 Alan Perlis and Klaus Samelson report on the International Algebraic Language, including “forms” for various language features.



4. *Functions F*

represent single numbers (function values), which result through the application of given sets of rules to fixed sets of parameters.

Form: $F \sim I\,(P, P, \sim\sim\sim\sim, P)$



5. *Arithmetic expressions E* are defined as follows:

a. A number, a variable (other than Boolean), or a function is an expression.

Form: $E \sim N \qquad \sim V \qquad \sim F$

b. If E_1 and E_2 are expressions, the first symbols of which are neither “+” nor “−”, then the following are expressions:

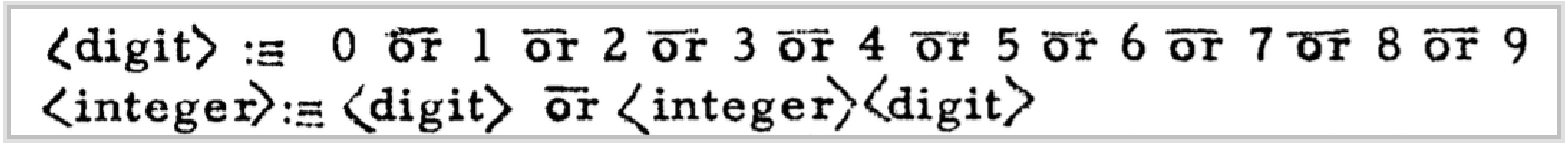
$$\begin{array}{llll} E & \sim + E_1 & \qquad & \sim E_1 \times E_2 \\ & \sim - E_2 & & \sim E_1 \;/\; E_2 \\ & \sim \quad E_1 + E_2 & & \sim E_1 \uparrow E_2 \downarrow \\ & \sim \quad E_1 - E_2 & & \sim (E_1) \end{array}$$

A. J. Perlis and K. Samelson. Preliminary report: International Algebraic Language. CACM 1, 12 (December 1958), 8–22.

Development of BNF: Backus

1959 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.



⟨digit⟩ :≡ 0 $\overline{\text{or}}$ 1 $\overline{\text{or}}$ 2 $\overline{\text{or}}$ 3 $\overline{\text{or}}$ 4 $\overline{\text{or}}$ 5 $\overline{\text{or}}$ 6 $\overline{\text{or}}$ 7 $\overline{\text{or}}$ 8 $\overline{\text{or}}$ 9

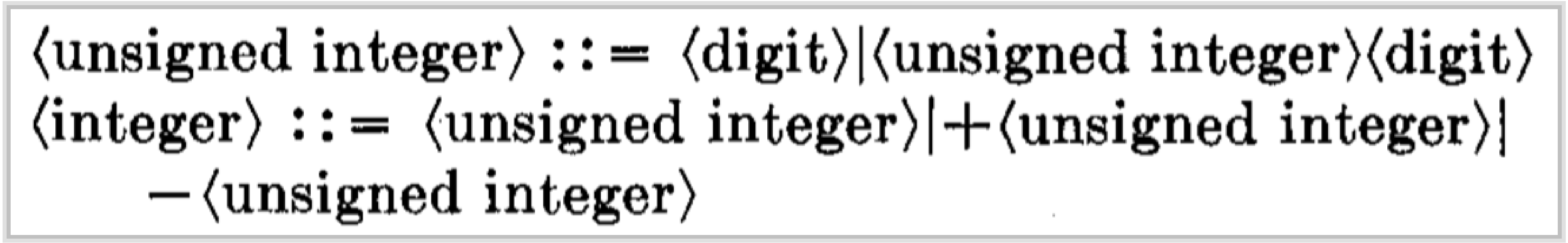
⟨integer⟩ :≡ ⟨digit⟩ $\overline{\text{or}}$ ⟨integer⟩⟨digit⟩

J. W. Backus. *The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference.* International Business Machines Corp., New York, page 14.

Development of BNF: Naur

1960 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.

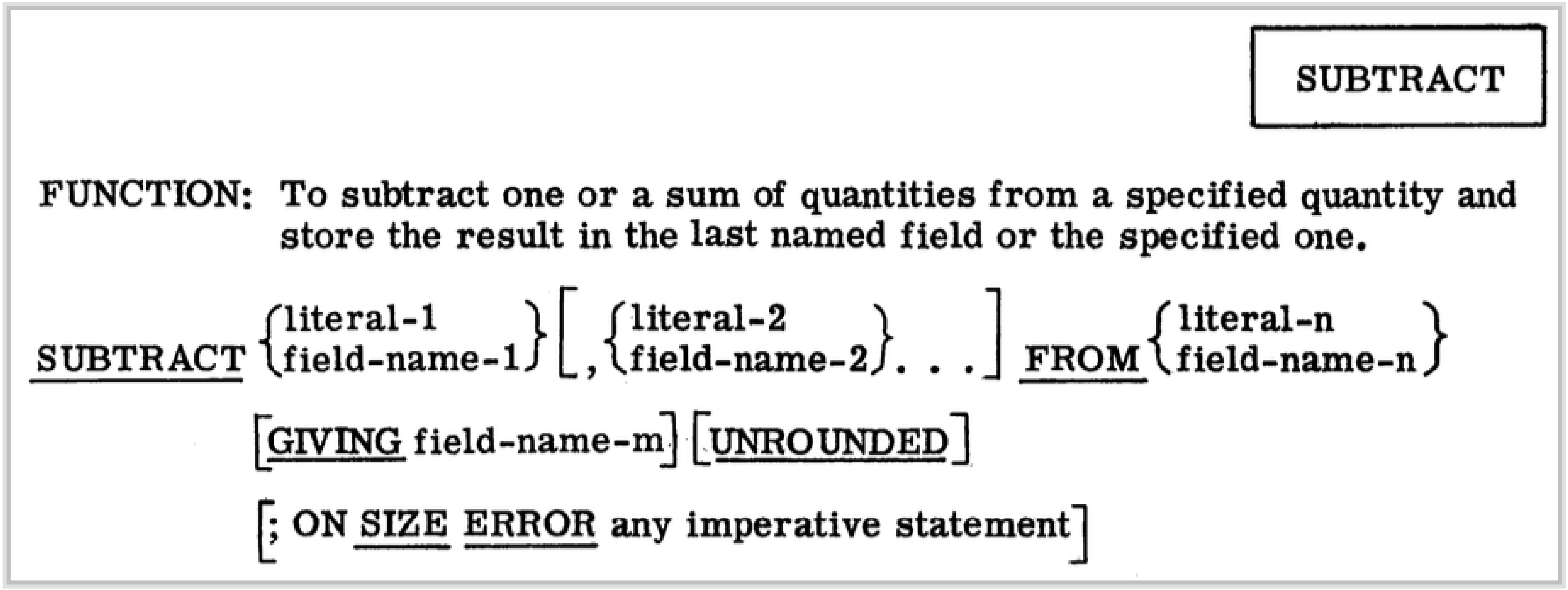


$$\langle\text{unsigned integer}\rangle ::= \langle\text{digit}\rangle | \langle\text{unsigned integer}\rangle\langle\text{digit}\rangle$$

$$\langle\text{integer}\rangle ::= \langle\text{unsigned integer}\rangle | + \langle\text{unsigned integer}\rangle | - \langle\text{unsigned integer}\rangle$$

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.*



$\boxed{\text{SUBTRACT}}$

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.

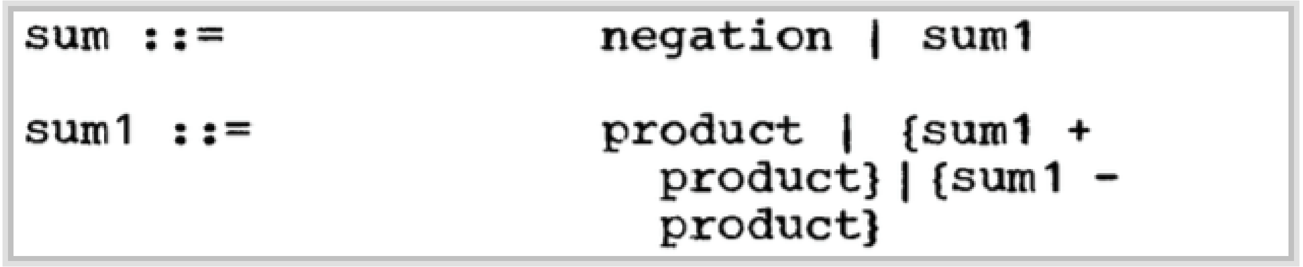
$$\underline{\text{SUBTRACT}} \left\{\begin{matrix}\text{literal-1}\\ \text{field-name-1}\end{matrix}\right\} \left[, \left\{\begin{matrix}\text{literal-2}\\ \text{field-name-2}\end{matrix}\right\} \ldots \right] \underline{\text{FROM}} \left\{\begin{matrix}\text{literal-n}\\ \text{field-name-n}\end{matrix}\right\}$$

$$\left[\underline{\text{GIVING}}\ \text{field-name-m}\right] \left[\underline{\text{UNROUNDED}}\right]$$

$$\left[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{any imperative statement}\right]$$

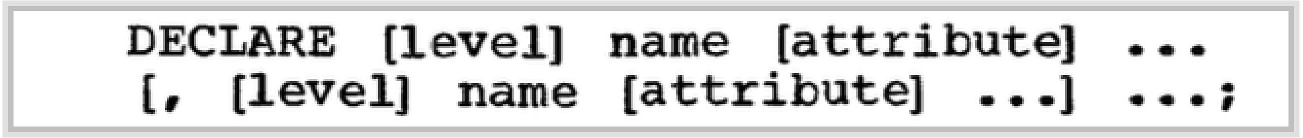
A Synthesis: PL/I Metanotation

1965 IBM's PL/I specification combines BNF with COBOL metanotation.

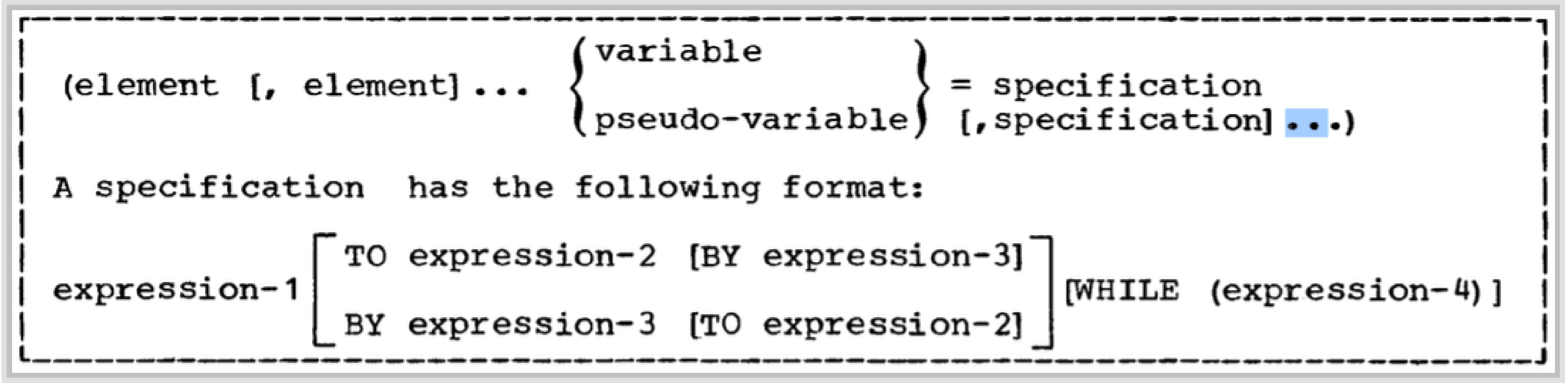


```
sum ::=                negation | sum1

sum1 ::=               product | {sum1 +
                          product} | {sum1 -
                          product}
```

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



```
    DECLARE [level] name [attribute] ...
    [, [level] name [attribute] ...] ...;
```

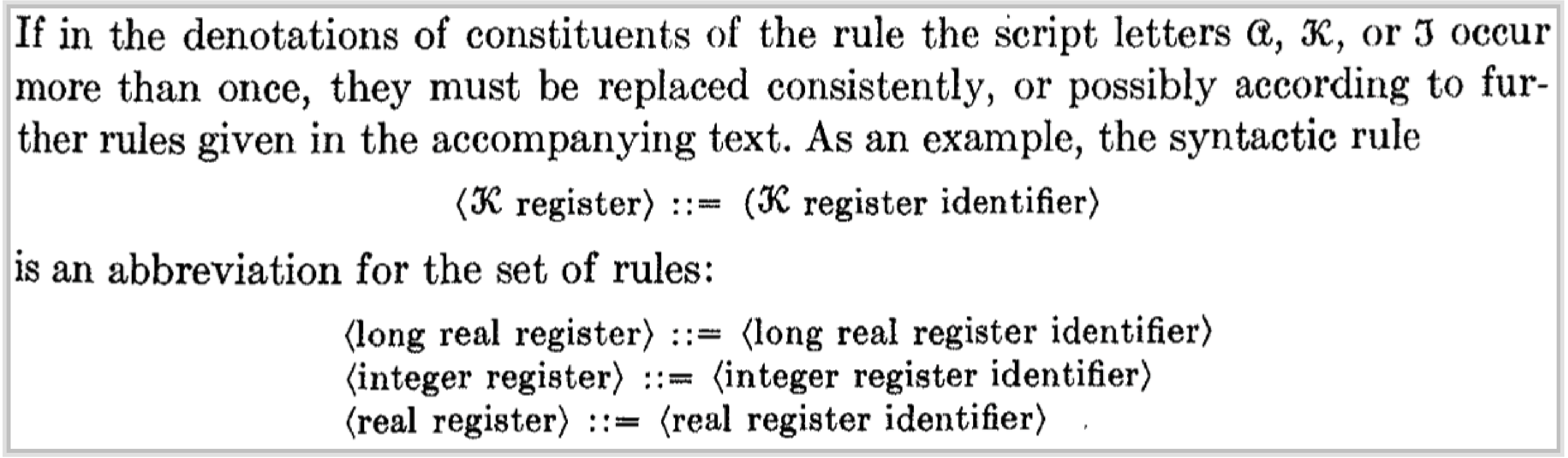


```
                              (variable        )
 (element [, element]...      {                } = specification
                              (pseudo-variable )  [,specification]...)

 A specification  has the following format:

               [TO expression-2  [BY expression-3]]
 expression-1  [                                  ] [WHILE (expression-4)]
               [BY expression-3  [TO expression-2]]
```

IBM Operating System/360: PL/I: Language Specifications. C28-6571-1 (July 1965), pages 37, 39, and 82.

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 $\mathcal{A}$, $\mathcal{K}$, or $\mathcal{J}$ 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

⟨$\mathcal{K}$ register⟩ ::= ⟨$\mathcal{K}$ register identifier⟩

is an abbreviation for the set of rules:

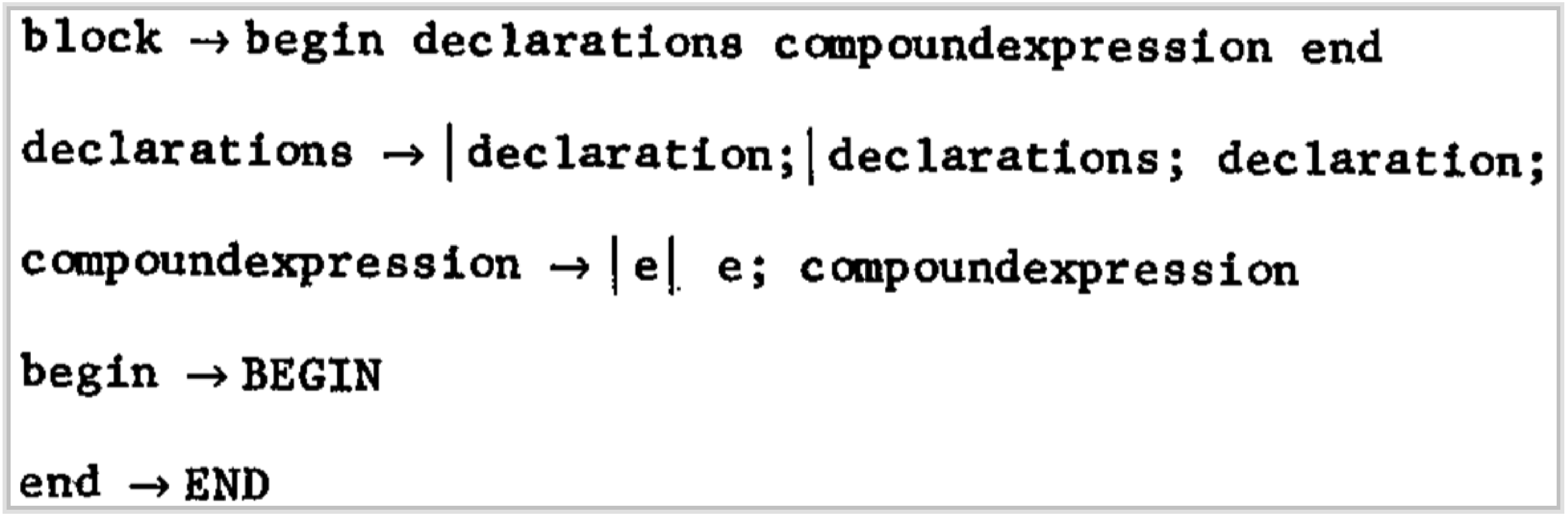
⟨long real register⟩ ::= ⟨long real register identifier⟩
⟨integer register⟩ ::= ⟨integer register identifier⟩
⟨real register⟩ ::= ⟨real register identifier⟩ ,

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.

Niklaus Wirth. *PL360, a Programming Language for the 360 Computers.* Stanford Computer Science Technical Report CS-TR-65-33, June 1965. Later published in *J. ACM* 15, 1 (January 1968), 37-74.
A. Van Wijngaarden, B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster. *Draft Report on the Algorithmic Language ALGOL 68.* Supplement to ALGOL Bulletin 26 (March 1968), 1–84.

BLISS

1970 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
```

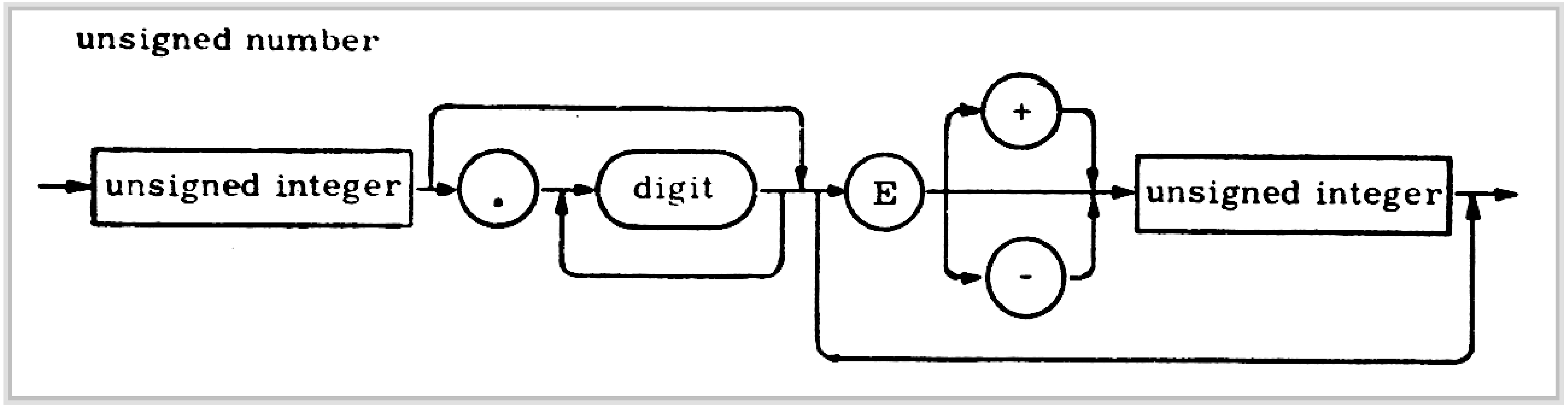
1980 The DEC BLISS documentation uses PL/I-style syntax descriptions.

W. A. Wulf, D. Russell, A. N. Habermann, C. Geschke, J. Apperson, and D. Wile.
BLISS Reference Manual: A Basic Language for Implementation of System Software for the PDP-10.
Computer Science Department, Carnegie-Mellon University (January 15, 1970), page 1.2.
Digital Equipment Corporation. *BLISS Language Guide*, Second Edition, AA-H275B-TK (January 1980).

Syntax Charts (Railway Diagrams)

1972 Burroughs CANDE language manual uses syntax charts only.

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



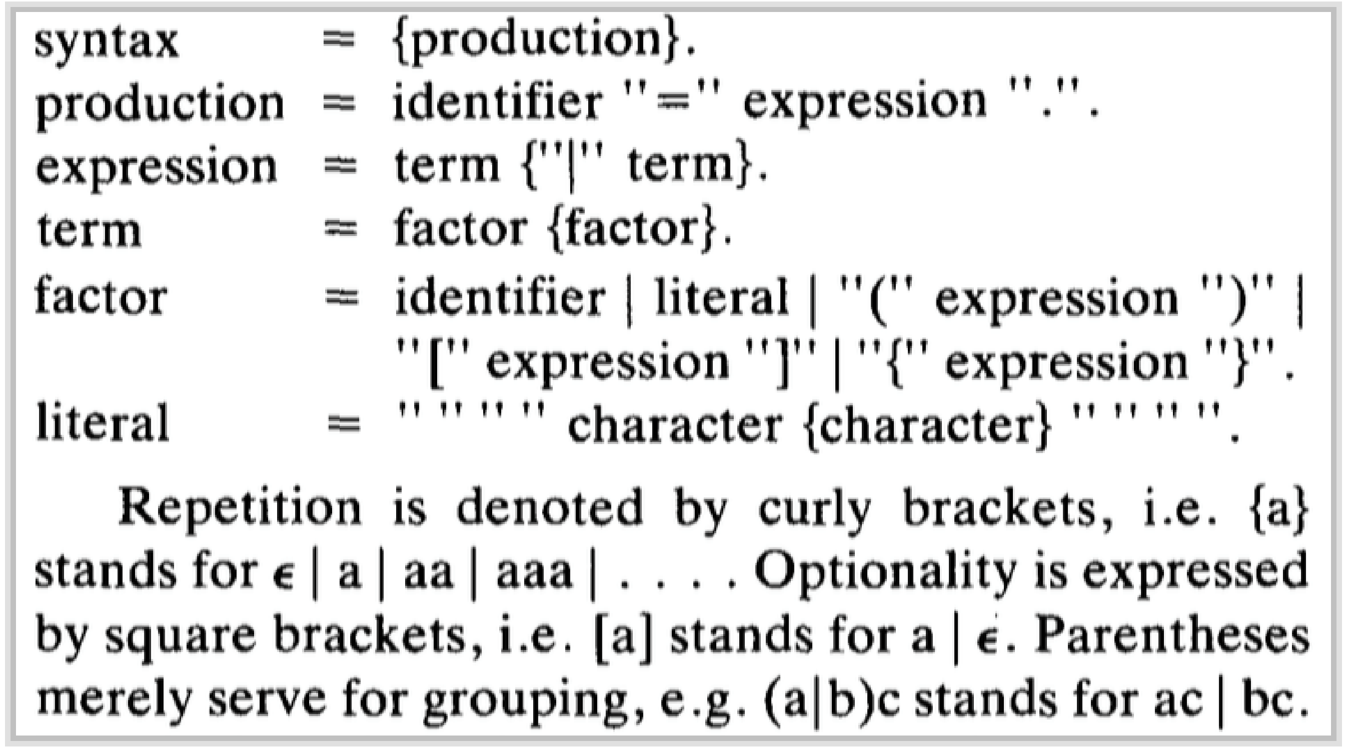
1978 Draft of FORTRAN 78 standard uses syntax charts plus PL/I-style BNF.

1979 The RED language (GREEN became Ada) uses syntax charts only.

Burroughs Corporation. *Burroughs B 6700 / C 7700 Command and Edit (CANDE) Language Information Manual.* 5000318 (2 October 1972).
Kathleen Jensen and Niklaus Wirth. *PASCAL User Manual and Report*. Springer-Verlag (1974), page 116.
Draft proposed ANS FORTRAN BSR X3.9 X3J3/76. *SIGPLAN Notices* 11, 3 (March 1976), 1-212.
John Nestor and Mary Van Deusen. *RED Language Reference Manual*. IR-310-2, Intermetrics (8 March 1979).

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.

1996 ISO/IEC Standard 14977:1996 *Extended BNF* (very similar to WSN).

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.

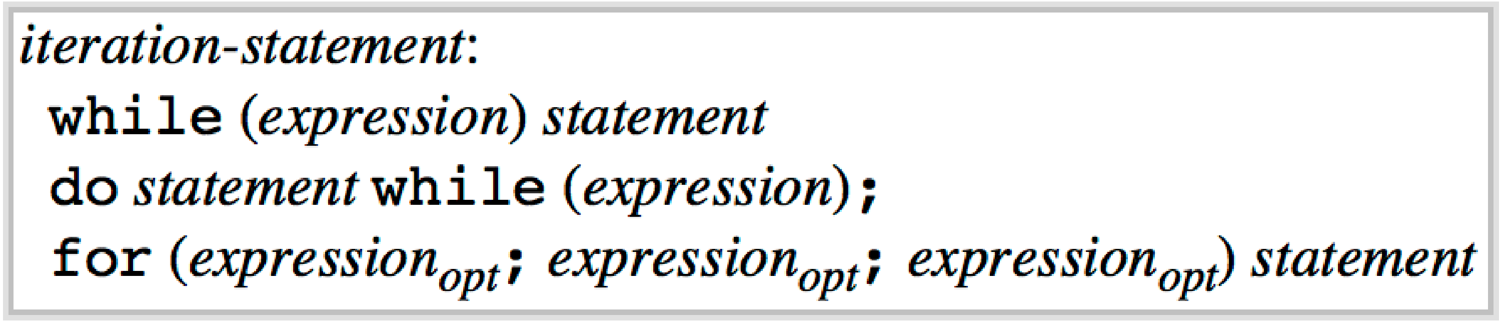
1995 *Python Reference Manual* (Release 1.2) uses ∗ and + in BNF, but brackets (rather than ?) for optional items.

1998 Haskell 98 Report uses BNF, with -> for ::=, and also uses ellipsis.

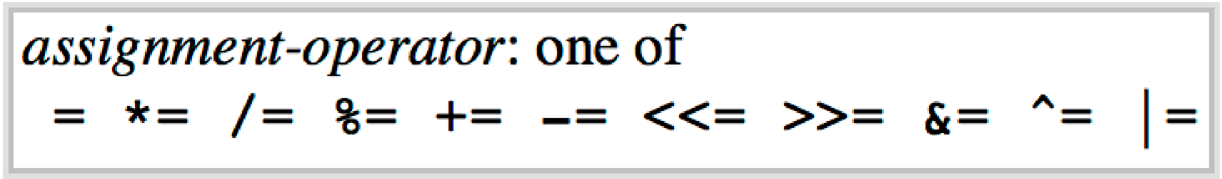
1998 *Ruby Language Reference Manual* (1.4.6) uses ∗ and + in "pseudo BNF" (somewhat like WSN), but brackets (rather than ?) for optional items.

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	Variable
	$\mid$	$\lambda x : \tau . e$	Abstraction
	$\mid$	$e_1\ e_2$	Application
	$\mid$	$\Lambda \alpha : \kappa . e$	Type abstraction
	$\mid$	$e\ \tau$	Type application

Types:

$\tau, \sigma, \psi, \upsilon$	$::=$	x	Type variable
	$\mid$	$\tau_1 \rightarrow \tau_2$	Function type
	$\mid$	$\forall \alpha : \kappa . \tau$	Polymorphic type
	$\mid$	$\tau_1\ \tau_2$	Application
	$\mid$	$F(\overline{\tau})$	Saturated type family
κ	$::=$	$\star \mid \kappa_1 \rightarrow \kappa_2$	Kind
Φ	$::=$	$[\overline{\alpha{:}\kappa}].\ F(\overline{\rho}) \sim \sigma$	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 τ may be assigned to any variable of type τ.

could be expanded to

A value of type `int` may be assigned to any variable of type `bool`.

which is nonsense. Instead, we require *consistent substitution*: within a given context, if a nonterminal is mentioned more than once, the same expansion must be used for each occurrence:

A value of type `int` may be assigned to any variable of type `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 $\tau \to \mathtt{int}$) to

$$\text{If } \mathtt{int}_1 = \mathtt{int}_2\text{, then } \mathtt{int}_1 <: \mathtt{int}_2.$$

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

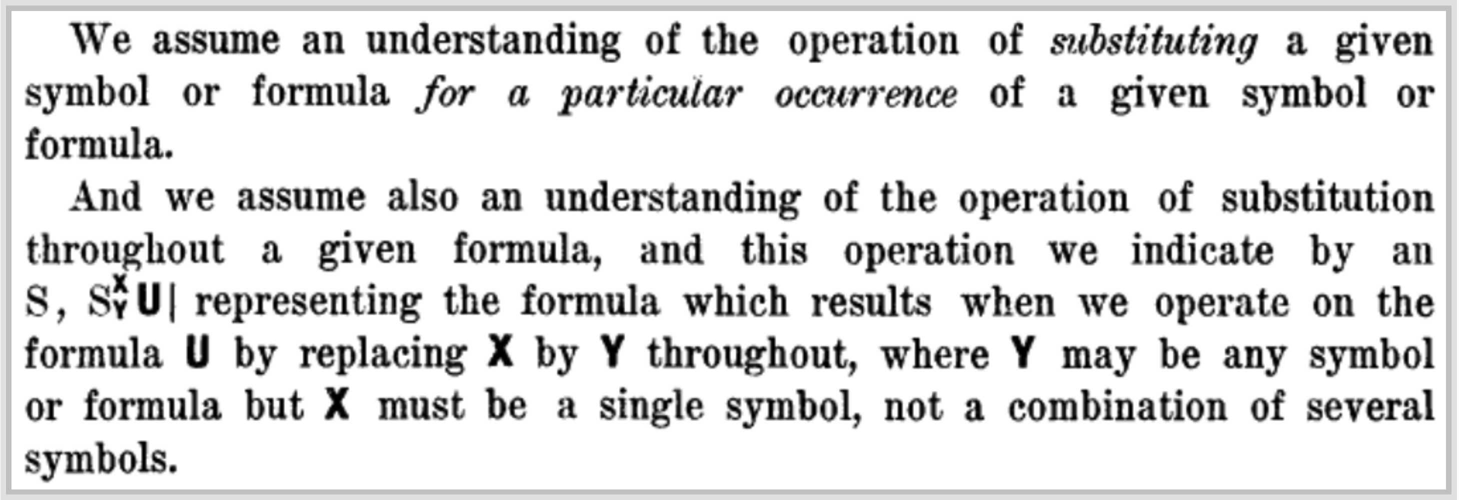
$$\text{If } \mathtt{int} = \mathtt{bool}\text{, then } \mathtt{int} <: \mathtt{bool}.$$

$$\text{If } \mathtt{int} = \mathtt{int}\text{, then } \mathtt{int} <: \mathtt{int}.$$

SUBSTITUTION

Substitution Notation

1932 Alonzo Church uses the notation $\mathrm{S}^{\mathrm{X}}_{\mathrm{Y}}\,\mathrm{U}|$ for *substitution* in a formula:

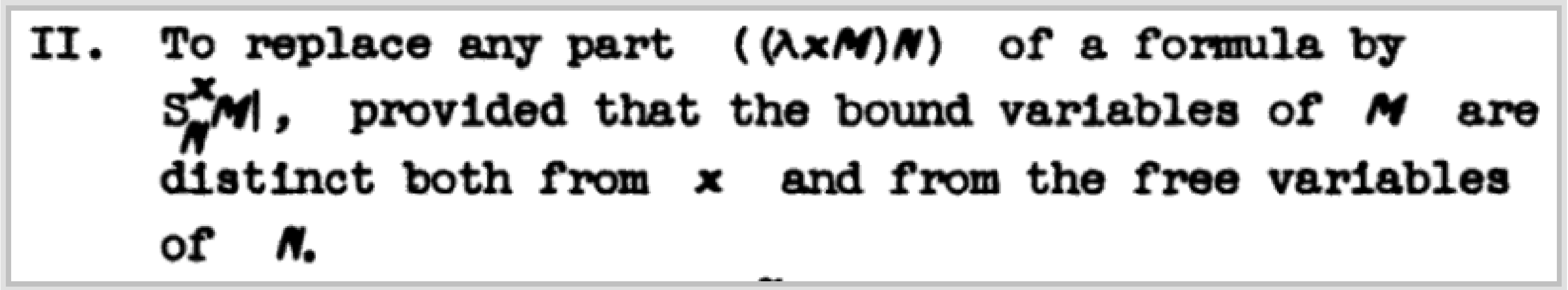


We assume an understanding of the operation of *substituting* a given symbol or formula *for a particular occurrence* of a given symbol or formula.

And we assume also an understanding of the operation of substitution throughout a given formula, and this operation we indicate by an S, $\mathrm{S}^{\mathbf{X}}_{\mathbf{Y}}\mathbf{U}|$ 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*.



II. To replace any part $((\lambda x M)N)$ of a formula by $\mathrm{S}^{x}_{N}M|$, provided that the bound variables of M are distinct both from x and from the free variables of N.

Alonzo Church. A Set of Postulates for the Foundation of Logic. *Annals of Mathematics* Second Series 33, 2 (April 1932), 346–366.
Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press (1941), page 12.

28 Varieties of Substitution Notation: POPL 1973–2016

$e\big\|^{v}_{x}$	1				
$e\frac{v}{x}$	1	$e[v/x]$	133	$e(v/x)$	1
$[v/x]e$	67	$e[{}^{v}/x]$	6	$e\{v/x\}$	25
$[{}^{v}/x]e$	1	$e[{}^{v}/_{x}]$	2	$e\{{}^{v}/x\}$	5
$[x := v]e$	2	$e[v \backslash x]$	1	$e\{{}^{v}/_{x}\}$	4
$[x \mapsto v]e$	9	$e[x/v]$	5	$e\{x \leftarrow v\}$	4
$[x \rightarrow v]e$	1	$e[x := v]$ *	21	$e\{x \mapsto v\}$	1
$[\![v/x]\!]e$	2	$e[x \leftarrow v]$	7	$e\{x \rightarrow v\}$	1
$\{v/x\}e$	6	$e[x \mapsto v]$	17	$e\{\|{}^{v}/x\|\}$	2
$\{x \mapsto v\}e$	4	$e[x \rightarrow v]$	2	$e\{\{x \leftarrow v\}\}$	1

* Used by H. P. Barendregt in *The Lambda Calculus: Its Syntax and Semantics* (1980).

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

ORACLE

Substitution: POPL 2012–2016 and Others 2014–2016

$e\big\vert^{v}_{x}$	1							
$e\frac{v}{x}$	1		$e[v/x]$	133	37	$e(v/x)$	1	
$[v/x]e$	67	17	$e[{}^{v}/x]$	6		$e\{v/x\}$	25	7
$[{}^{v}/x]e$	1		$e[{}^{v}/_{x}]$	2	2	$e\{{}^{v}/x\}$	5	
$[x := v]e$	2		$e[v\backslash x]$	1		$e\{{}^{v}/_{x}\}$	4	
$[x \mapsto v]e$	9	2	$e[x/v]$	5	1	$e\{x \leftarrow v\}$	4	1
$[x \rightarrow v]e$	1		$e[x := v]$	21	3	$e\{x \mapsto v\}$	1	
$[\![v/x]\!]e$	2		$e[x \leftarrow v]$	7	1	$e\{x \rightarrow v\}$	1	
$\{v/x\}e$	6		$e[x \mapsto v]$	17	8	$e\{\vert{}^{v}/x\vert\}$	2	
$\{x \mapsto v\}e$	4		$e[x \rightarrow v]$	2	1	$e\{\!\{x \leftarrow v\}\!\}$	1	
$[{}^{v}/_{x}]e$		1	$e\{{}^{v}/_{x}\}$		1	$e\{x := v\}$		1

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.

ORACLE

Substitution: A Small Problem

It's sort of weird to have in live use both

$e[x \to v]$ and $e[x \leftarrow v]$.

You would think that, if anything, it would be

$e[x \to v]$ and $e[v \leftarrow x]$

or maybe

$e[v \to x]$ and $e[x \leftarrow v]$.

Same thing for $e\{x \to v\}$ and $e\{x \leftarrow v\}$.

But it's only a small problem, because these uses are rare.

ORACLE®

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.

Substitution: A Huge Problem

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)!

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

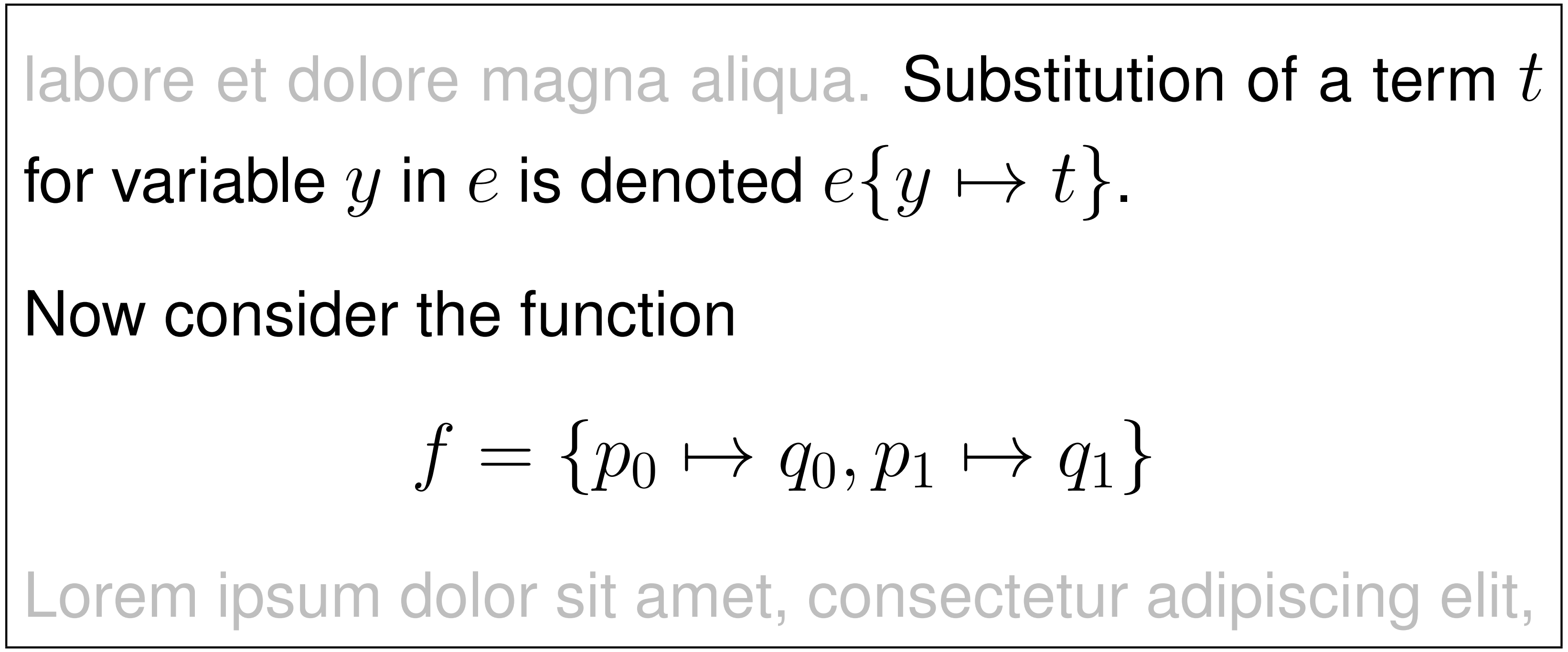
Use of both in one paper can make it very hard to read.

And lately most authors are taking all these notations for granted.

ORACLE®

An (Unpublished) Paper I Read Just Last Month

(Slightly modified for purposes of anonymization.)



labore et dolore magna aliqua. Substitution of a term t for variable y in e is denoted $e\{y \mapsto t\}$.

Now consider the function

$$f = \{p_0 \mapsto q_0, p_1 \mapsto q_1\}$$

Lorem ipsum dolor sit amet, consectetur adipiscing elit,

(The substitution notation was not used until six pages later.)

Substitution: My Recommendations

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

applications		operators		(singleton) collections	
substitution	$e[v/x]$	a substitution	$\sigma = [v/x]$		
substitution	$e[x \rightarrow v]$	a substitution	$\sigma = [x \rightarrow v]$		
map update	$\Gamma[x \mapsto v]$	a map update	$u = [x \mapsto v]$	a map	$\Gamma = \{x \mapsto v\}$
heap update	$H[x := v]$	a heap update	$u = [x := v]$	a heap	$H = \{x := v\}$

- Mnemonic for $e[v/x]$: “Within e, v *supersedes* (‘sits over’) x.”
- Mnemonic for $e[x \rightarrow v]$: “Within e, x *becomes* v.”

OVERLINE

Overline Notation (and Dots and Parentheses) (1 of 2)

1484 Nicolas Chuquet uses an *underline* for mathematical grouping.

1525 Christoff Rudolff uses the sign $\surd$ to indicate taking a square root, and also uses dots to indicate grouping: $\surd .12+\surd 140$ means $\sqrt{12+\sqrt{140}}$.

1556 Niccolò Tartaglia uses parentheses $(\ )$ for mathematical grouping.

1631 William Oughtred uses double dots $:$ to indicate grouping.

1631 Thomas Harriott uses a long overbrace with $\surd$ for grouping.

1637 René Descartes attaches an *overline* to $\surd$, producing $\sqrt{}$.

1640 Jan Stampioen uses all three *together*: “$\surd .\overline{(aaa+6aab+9bba)}$”.

1646 Frans van Schooten, editing Vieta’s works, uses overline for grouping.

1702 Gottfried Leibniz begins using parentheses in preference to overline.

1708 *Acta eruditorum* officially adopts the Leibnizian symbolism.

1709 Pierre Louis Maupertuis uses square brackets $[\ ]$.

Overline Notation (and Dots and Parentheses) (2 of 2)

1728– Leonhard Euler, Johann Bernoulli, and Daniel Bernoulli use parentheses and brackets in their publications.

> "The constant use of parentheses in the stream of articles from the pen of Euler that appeared during the eighteenth century contributed vastly toward accustoming mathematicians to their use."
>
> —Florian Cajori, *A History of Mathematical Notations*

1857 Giuseppe Peano reintroduces dots (after a century and a half of disuse), letting dot count indicate "binding weakness": "$a\!:\!bc.d$" means "$a((bc)d)$".

1881 Josiah Willard Gibbs notates a vector as $\overline{AB}$.

1910 Russell and Whitehead (*Principia Mathematica*) adopt Peano's dots.

Three notations for grouping duking it out for five centuries!

A Little Bit 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 $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$.

1844–46 Hamilton reformulates quaternions without $\mathbf{ijk}$ 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”!

Vectors and Overlines at POPL (1 of 3)

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 $\overline{x : \tau}$ 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.)

Vectors and Overlines at POPL (2 of 3)

1990 First explicit claim that the elements may be *metasyntactic variables*:
"we use the notation $\overline{\chi}$, for some metasyntactic variable χ 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, $\forall\overline{\alpha_n : \sigma_n}.\sigma$ 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 … $\hat{\rho}(\overline{\overline{c_1}; \overline{c_2}}, \overline{d_1; d_2}) = \hat{\rho}(\overline{c_1}, \overline{c_2}); \hat{\rho}(\overline{d_1}, \overline{d_2})$."

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 “$\overline{T}\ \ \overline{x}$” 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!)

Vectors and Overlines at POPL (3 of 3)

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 $[\overline{z}/\overline{y}]$ that the sequence $\overline{y}$ is linear and of the same length as $\overline{z}$."

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!

The Problem (1997)

What is the meaning of $\Gamma(\tilde{b}) = [\tilde{T}/\tilde{X}]\tilde{P}$?

If we regard substitution "$[\,\cdot\,/\,\cdot\,]\,\cdot$" 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:

$$\begin{array}{ll} & \Gamma(b_1) = [T_1/X_1, \ldots, T_m/x_m]P_1 \\ \text{and} & \ldots \\ \text{and} & \Gamma(b_n) = [T_1/X_1, \ldots, T_m/x_m]P_n \end{array}$$

A Solution? Nested Overlines (1 of 2)

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

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

$$\begin{array}{ll} & p_1 = [v_1/x_1, \ldots, v_m/x_m]q_1 \\ \text{and} & \ldots \\ \text{and} & p_n = [v_1/x_1, \ldots, v_m/x_m]q_n \end{array}$$

where v and x are one-dimensional

and not something like

$$\begin{array}{ll} & p_1 = [v_{1\,1}/x_{1\,1}, \ldots, v_{1\,m}/x_{1\,m}]q_1 \\ \text{and} & \ldots \\ \text{and} & p_n = [v_{n\,1}/x_{n\,1}, \ldots, v_{n\,m}/x_{n\,m}]q_n \end{array}$$

where v and x are two-dimensional

?

A Solution? Nested Overlines (2 of 2)

Even without nesting, some authors write $\overline{\Gamma \vdash x : \tau}$, intending

$$\Gamma \vdash x_1 : \tau_1 \qquad \Gamma \vdash x_2 : \tau_2 \qquad \ldots \qquad \Gamma \vdash x_n : \tau_n$$

How do we know it isn't supposed to be

$$\Gamma_1 \vdash x_1 : \tau_1 \qquad \Gamma_2 \vdash x_2 : \tau_2 \qquad \ldots \qquad \Gamma_n \vdash x_n : \tau_n \quad ?$$

And we would have the same problem with $\overline{\Gamma(b) = [\overline{T/X}]P}$:
why should b and T and X and P get subscripts, but not Γ?

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 $\overline{p = [\overline{v/x}]q}$ to mean

$$
\begin{array}{ll}
 & p_1 = [v_1/X_1, \ldots, v_m/x_m]q_1 \\
\text{and} & \ldots \\
\text{and} & p_n = [v_1/X_1, \ldots, v_m/x_m]q_n
\end{array}
$$

where all the substitutions are the *same*

but we want $\texttt{case}\ e\ \texttt{of}\ \overline{K\ \overline{y} \rightarrow e'}$ to mean

$$
\begin{array}{l}
\texttt{case}\ e\ \texttt{of} \\
\quad K_1\, y_{1\,1}\ \ldots\ y_{1\,m_1} \rightarrow e'_1 \\
\quad \ldots \\
\quad K_n\, y_{n\,1}\ \ldots\ y_{n\,m_n} \rightarrow e'_n
\end{array}
$$

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.

ORACLE

What Do We Want From Overline Notation? (1 of 2)

- $\overline{string}$ can expand to any number of copies of $string$.
  - More concise than ellipsis notation.
  - Question: whether and how copies are separated (comma by default?).
  - If we want “$\overline{x}, \overline{y}$” for concatenation, sequence should be unenclosed.
- Each copy of $string$ may be expanded *differently*.
  - BNF nonterminals may be expanded differently in each copy.
  - Nested overlines may be expanded differently in each copy.
    - ⋆ This suggests that nested overlines should be processed outside-in.
- *But*, if $\overline{string}$ 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 *same* (similar to treatment of BNF nonterminals).

What Do We Want From Overline Notation? (2 of 2)

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

Guy L. Steele Jr. and W. Daniel Hillis.
Connection Machine Lisp: Fine-grained Parallel Symbolic Processing.
Proc. 1986 ACM Conference on LISP and Functional Programming, 279-297.

Adding Underlines to Overlines

We propose this modification to overline notation:

- Each copy of $string$ may be expanded *differently*, but underline means “except here”: any underlined portion of $string$ must be expanded the *same* way in each copy.

Therefore for our examples we can write:

$\overline{p = \underline{[\overline{v/x}]}q}$	same substitution in each outer copy
$\mathtt{case}\ e\ \mathtt{of}\ \overline{K\ \overline{y} \to e'}$	as before
$\overline{\underline{\Gamma}(b) = \underline{[\overline{T/X}]}P}$	same Γ and same substitution in each outer copy
$\overline{\underline{\Gamma} \vdash x : \tau}$	same Γ in each copy

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:

$$\overline{\Gamma(b_i) = [\overline{T_j/X_j}]P_i}$$

It takes some analysis to match the indices to the overlines.

Other writers explicitly mark the binding points: $\overline{\Gamma(b_i) = [\overline{T_j/X_j}^{\,j}]P_i}^{\,i}$

and some even explicitly specify ranges: $\overline{\Gamma(b_i) = \left[\overline{T_j/X_j}^{\,1\leq j\leq m}\right]P_i}^{\,0\leq i<n}$

I endorse these 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, ‘$1 + 2 + \cdots + 20 = 210$’

—Roland Backhouse, *Program Construction* (Wiley, 2003), p. 137

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?

We propose to explain the *ellipsis* notation

by providing a formal transformation to *overline* notation

(whose formal definition need not rely on ellipses).

Ellipsis Notation Has Many Forms

$1, 2, \ldots, n$

$x_1, \ldots, x_n$

$a_1 \oplus \cdots \oplus a_n$

$x_0, \ldots, x_n$

$x_0, \ldots, x_{n-1}$

$x_1, x_2, \ldots, x_{n-1}, x_n$

$x_1, x_2, \ldots$

$x_1, \ldots$

$x_1 + y_1 + c, \ldots, x_n + y_n + c$

$\textbf{if } p_1 \textbf{ then } e_1$
$\textbf{else if } p_2 \textbf{ then } e_2$
$\textbf{else if } \ldots$
$\textbf{else if } p_{n-1} \textbf{ then } e_{n-1}$
$\textbf{else } e_n$

$(e_1, \ldots, e_i, \ldots, e_n)$

$(e_1, \ldots, e_{i-1}, \texttt{this}, e_{i+1}, \ldots, e_n)$

$(\ldots, e_{i-1}, \texttt{this}, e_{i+1}, \ldots)$

$(q, x_1, \ldots, x_n, y_1, \ldots, y_n)$

$(x_1, y_1, \ldots, x_n, y_n)$

More Difficult Ellipsis Notation Forms

- Not really a series; each “. . . ” is just a list of don’t-cares:

  $$\text{class } C \; \{ \; \ldots \; T_1 \; m_1(\ldots) \; body_1 \; \ldots \; T_2 \; m_2(\ldots) \; body_2 \ldots \; \}$$

- Matrix elements $m \times n$ listed linearly (nested use of ellipsis notation!):

  $$x_{11}, x_{12}, \ldots, x_{1n}, \ldots, x_{m1}, x_{m2}, \ldots, x_{mn}$$

- Nested function calls (note use of two ellipses):

  $$f_1(f_2(\ldots(f_n(x))\ldots))$$

- Nested function calls with other arguments:

  $$f_1(f_2(\ldots(f_n(x, y_n))\ldots, y_2), y_1)$$

- Function calls nested the other way:

  $$f_n(f_{n-1}(\ldots(f_2(f_1(x, y_1), y_2), \ldots, y_{n-1}), y_n)$$

ORACLE

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_1 and x_n;

the common structure is x_i with substitutions $[0/i]$ and $[n-1/i]$;

the result is $\overline{x_i}^{0 \leq i \leq n-1}$.

Example 2: “$a_1 b_1 \oplus a_2 b_2 \oplus \ldots$”:

the separator is “$\oplus$”;

the matchable strings are $a_1 b_1$ and $a_2 b_2$;

the common structure is $a_i b_i$ with substitutions $[1/i]$ and $[2/i]$;

the pattern requires verification that 1 and 2 are consecutive integers;

and the result is $\overrightharpoon{a_i b_i \underbracket{\oplus}}^{i}$.

Overall Structure of the Approach

- Regard the text to be processed as a horizontally linear string of tokens.
- Repeat the following until some complete pass fails to change the text:
  - For every ellipsis in the text (processing as if in some sequential order):
    - ⋆ Attempt a left-and-right single-ellipsis replacement.
    - ⋆ If that fails, attempt a left-only single-ellipsis replacement.
    - ⋆ If that fails, attempt a double-ellipsis replacement.
    - ⋆ If that fails, . . . $\langle$there may be other kinds$\rangle$.
- Every remaining ellipsis that is both left-delimited and right-delimited is replaced with “$\overline{\_}$” (an overline cluster containing the don’t-care symbol).
- It is an error if there is any remaining ellipsis in the text.

Left-and-Right Single Ellipsis Replacement $(a_1, \ldots, a_n)$

Let i be a fresh variable. Try to identify a substring having pattern $\alpha'\kappa$ ▪▪▪ $\kappa\alpha''$ such that the "▪▪▪" in the pattern corresponds to the ellipsis in the text and:

- the substring is completely contained by any overline or underline containing the ellipsis;
- each of $\alpha'\kappa$ and $\kappa\alpha''$ is maximal, is balanced, and contains no ellipsis; and
- there exist nonempty α and p' and p'' such that:
  - i occurs at least once in α, $\alpha[p'/i] = \alpha'$, and $\alpha[p''/i] = \alpha''$; and
  - p' and p'' are balanced and minimal, and $p' \neq p''$.

If successful, then if κ is ",", replace the matched substring with $\overline{\alpha}^{p \le i \le q}$;
otherwise, if κ is empty, replace the matched substring with $\overrightarrow{\alpha}^{p \le i \le q}$;
otherwise replace the matched substring with $\overrightarrow{\alpha \underbracket{\boldsymbol{\kappa}}}^{p \le i \le q}$.

Left-only Two-term Single Ellipsis Replacement $(a_1, a_2, \ldots)$

Similar, but seek a substring of the pattern $\alpha'\kappa\alpha''\kappa$ ▪▪▪ such that:

- κ is a single token that is not a left or right encloser;
- each of α' and α'' is maximal, is balanced, and contains no ellipsis; and
- there exist nonempty α and p' and p'' such that:
  - i occurs at least once in α, $\alpha' = \alpha[p'/i]$, and $\alpha'' = \alpha[p''/i]$;
  - p' and p'' are balanced and minimal;
  - $p'' = p' + 1$ (as determined by a solver).

If $p' = 1$, let $\alpha''' = \alpha$; otherwise let $\alpha''' = \alpha[(i - 1 + p')/i]$.

If successful, then if κ is empty, replace the matched substring with $\overrightarrow{\alpha'''}$;
otherwise replace the matched substring with $\overrightarrow{\alpha'''\underbracket{\kappa}}$.

Conclusions

- Computer Science Metanotation is a 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 years.
- Some recent changes have caused problems. 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?