Semantic relationships: reducing the separation between theory and practice

Robert Milne

rem@antelope.org.uk

The sixties

1960

1970

Basic attitude

"*It has long been my personal view that the separation of practical and theoretical work is artificial and injurious. Much of the practical work done in computing, both in software and in hardware design, is unsound and clumsy because the people who do it have not any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing.*"

Christopher Strachey, *Towards a formal semantics*, 1966.

"*We need to develop our insight into computing processes and to recognise and isolate the central concepts—things analogous to the concepts of continuity and convergence in analysis. To do this we must become familiar with them and give them names even before we are really satisfied that we have described them precisely. If we attempt to formalise our ideas before we have really sorted out the important concepts the result, though possibly rigorous, is of very little value—indeed it may well do more harm than good by making it harder to discover the really important concepts. Our motto should be 'No axiomatisation without insight'.*"

Christopher Strachey, *Fundamental concepts in programming languages*, 1967.

The Programming Research Group

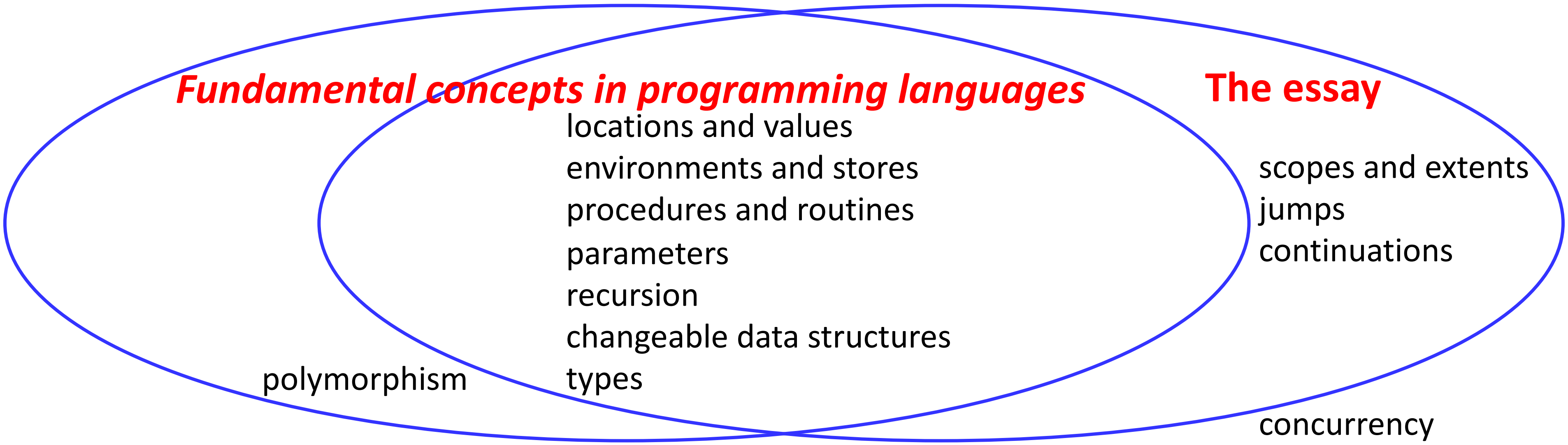
- Attracted because of these early papers and the subsequent progress.
- Unstructured and informal, perhaps as when Christopher had one employee.
- Occupied occasionally by up to twelve people (half being students).
- Slightly more structured when we wrote the essay for the Adams Prize.

Writing the essay

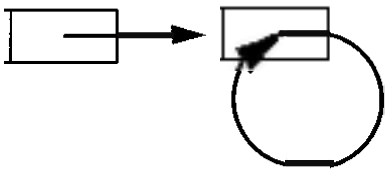
- **Typing**
  - **Multiple golf balls per line and at least four per page.**
  - **Up to fifty written or stamped script characters per page.**
- **Correction**
  - **Different alignments of moved and reinserted pages.**
  - **Different reflectances of original and amended characters.**
- **Notation**
  - **Few simplifications.**
  - **Detailed proofs to show feasibility.**
  - **Explicit entities to limit abstraction.**

$\mathcal{C}[\![\mathrm{E}_0\mathrm{E}_1]\!] =$
$\lambda\rho\theta.\mathcal{E}[\![\mathrm{E}_0]\!]\rho(\lambda\varepsilon_0.\mathcal{E}[\![\mathrm{E}_1]\!]\rho(\lambda\varepsilon_1.apply\ \varepsilon_0\ \varepsilon_1\theta))$
would be used.
$\mathcal{C}[\![\mathrm{E}_0\mathrm{E}_1]\!] =$
$let\ \varepsilon_0 = \mathcal{E}[\![\mathrm{E}_0]\!]\ in\ let\ \varepsilon_1 = \mathcal{E}[\![\mathrm{E}_1]\!]\ in\ apply\ \varepsilon_0\ \varepsilon_1$
(with or without the brackets) could have served instead in all forms of semantics, not just this one.

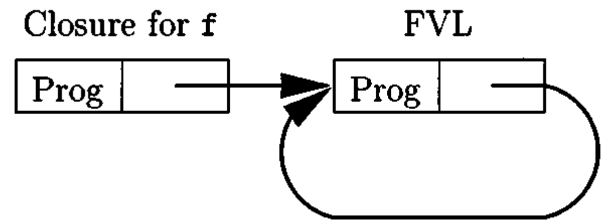
Describing the fundamental concepts



Relating theory to practice



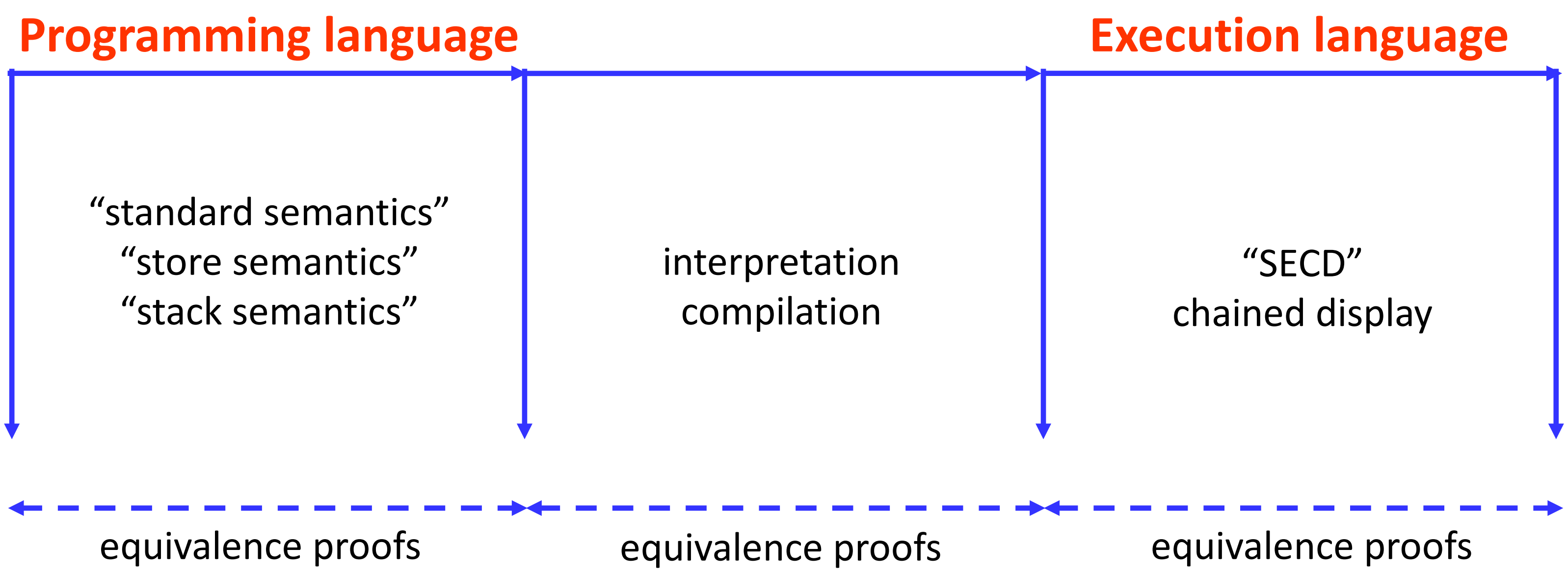
After *Fundamental concepts in programming languages*



From *Fundamental concepts in programming languages*

- **Procedure modelled by theory**
  - **Mathematical function.**
  - **Environment embedded in the function.**
  - **Recursion by introducing a fixed point of the function.**

- **Procedure implemented in practice**
  - **Executable statement.**
  - **Environment ("FVL") with an explicit pointer.**
  - **Recursion by pointing back to the statement through the location.**



Relationships between forms of the semantics

Programming language

Execution language

program fragment Γ → program fragment Γ → executable statement Π → executable statement Π

program fragment Γ → program fragment Γ: restricting program fragments to ones for which different forms of the semantics should be related

$c[\![\Gamma]\!]\acute{\rho}$

program fragment Γ → executable statement Π: translating program fragments into executable statements

$\Gamma \mapsto c[\![\Gamma]\!]\grave{\rho}\nu$

executable statement Π → executable statement Π: identifying executable statements for which different forms of the semantics should be related

$\Pi \mapsto \Pi$

denoting ↓ entities more deeply embedded in functions

$\mathcal{C}[\![\Gamma]\!]\acute{\rho}\acute{\theta}\acute{\sigma}$

denoting ↓ entities less deeply embedded in functions

$\mathcal{C}[\![\Gamma]\!]\grave{\rho}\grave{\zeta}\grave{\upsilon}\grave{\sigma}$

denoting ↓ execution states as arguments and executable code as functions

$\mathcal{V}[\![\Pi]\!]\grave{\zeta}\nu\grave{\upsilon}\grave{\sigma}$

denoting ↓ execution states as tuples and executable code as text

$\mathcal{Z}[\![\Pi]\!]\nu\upsilon\sigma$

first ↔ second: related by inclusive predicates (or "logical relations")

second ↔ third: related by inclusive predicates and partial orders

$$\mathcal{C}[\![\Gamma]\!]\grave{\rho}\grave{\zeta}\grave{\upsilon}\grave{\sigma} = \mathcal{V}[\![c[\![\Gamma]\!]\grave{\rho}1]\!]\grave{\zeta}1\grave{\upsilon}\grave{\sigma}$$

third ↔ fourth: related by inclusive predicates (or "logical relations")

The abstract model for storage

The effect of an assignment command is to change the contents of the store of the machine. Thus it alters the relationship between L-values and R-values and so changes σ. We can therefore regard assignment as an operator on σ which produces a fresh σ. If we update the L-value α (whose original R-value in σ was β*) by a fresh R-value* β*′ to produce a new store σ′, we want the R-value of α in σ′ to be* β*′, while the R-value of all other L-values remain unaltered.*

Christopher Strachey, *Fundamental concepts in programming languages*, 1967.

Thus storage is modelled by such functions as the following.

$area: \mathbf{L} \to \mathbf{S} \to \mathbf{T}$	$area\ \alpha(update\ \alpha'\beta\sigma) = if\ \alpha = \alpha'\ then\ true\ else\ area\ \alpha\sigma$
$hold: \mathbf{L} \to \mathbf{S} \to \mathbf{V}$	$hold\ \alpha(update\ \alpha'\beta\sigma) = if\ \alpha = \alpha'\ then\ \beta\ else\ hold\ \alpha\sigma$
$new: \mathbf{S} \to \mathbf{L}$	$area\ (new\ \sigma)\sigma = false$
$empty: \mathbf{S}$	$area\ \alpha(empty) = false$
$update: \mathbf{L} \to \mathbf{V} \to \mathbf{S} \to \mathbf{S}$	

Problems and solutions for storage

```
fun f(z) = y := !ref(0)
f(2)
val x = ref(1)
```

```
fun f(z) = y := !ref(0)
val x = ref(1)
f(2)
```

equivalent

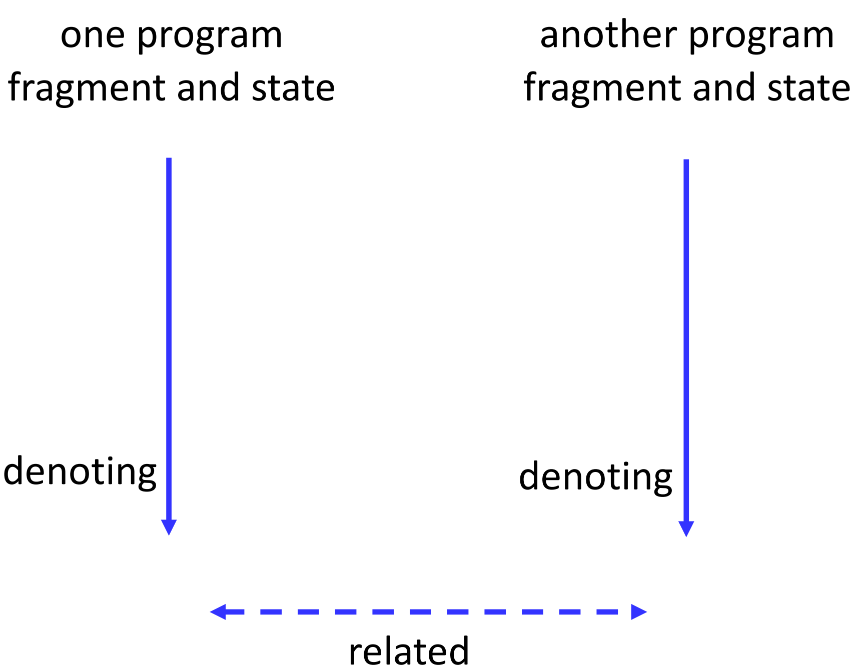
- **Assignment of an integer**
  - **The location for x is inaccessible in f.**
  - **The fragments should be equivalent.**
  - **Their denotations might be unequal.**

```
fun f(z) = y := ref(0)
f(2)
val x = ref(1)
```

```
fun f(z)  = y := ref(0)
val x = ref(1)
f(2)
```

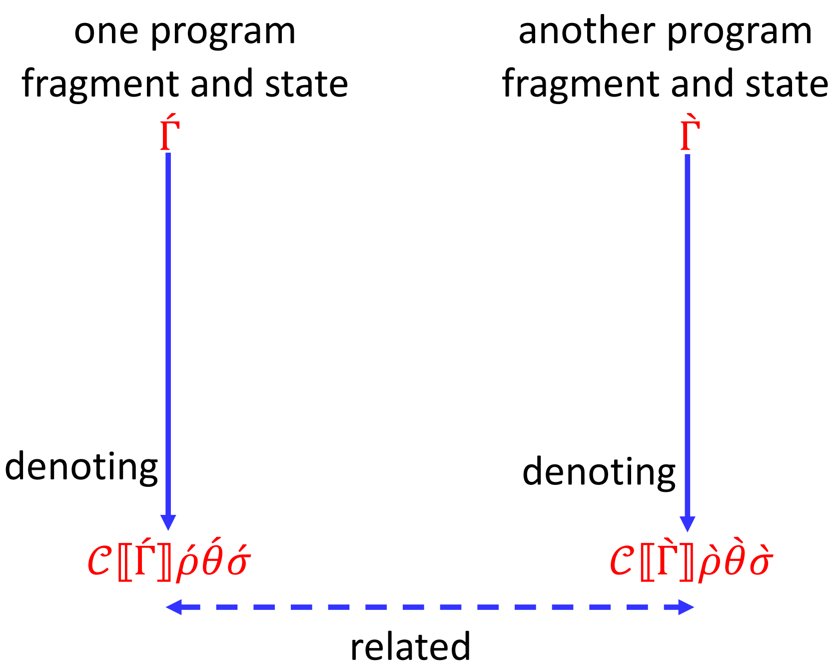
inequivalent

- **Assignment of a reference**
  - **The location for x is dependent on f.**
  - **The fragments should be inequivalent.**
  - **Their denotations should be unequal.**



- **Relations are based on states such as:**
  - **Stores (if locations can be paired with other entities).**
  - **Locations (if locations are paired only with locations).**
  - **Stacks and stores (if, as in the essay, the relations are between "stack semantics" and "store semantics", with states ordered by *match* and restricted by *seen*).**

Principles for reasoning about storage



$c[\![\acute{\Gamma}]\!]\acute{\chi} \wedge c[\![\grave{\Gamma}]\!]\grave{\chi} \Rightarrow consistent\ \acute{\chi}\acute{\pi}\acute{\rho} \wedge consistent\ \grave{\chi}\grave{\pi}\grave{\rho} \Rightarrow$
$u_{\hat{\pi}}\hat{\rho} \Rightarrow (c_{\hat{\pi}} \to c_{\hat{\pi}})\ \langle \mathcal{C}[\![\acute{\Gamma}]\!]\acute{\rho}, \mathcal{C}[\![\grave{\Gamma}]\!]\grave{\rho}\rangle$

$c[\![\acute{\Gamma}]\!](extract\ \acute{\pi}\acute{\rho}) \wedge c[\![\grave{\Gamma}]\!](extract\ \grave{\pi}\grave{\rho}) \Rightarrow$
$u_{\hat{\pi}}\hat{\rho} \Rightarrow (c_{\hat{\pi}} \to c_{\hat{\pi}})\ \langle \mathcal{C}[\![\acute{\Gamma}]\!]\acute{\rho}, \mathcal{C}[\![\grave{\Gamma}]\!]\grave{\rho}\rangle$

- **Constrain fragments to be consistent with the expected relations.**
  $c[\![\acute{\Gamma}]\!]\acute{\chi} \wedge consistent\ \acute{\chi}\acute{\pi}\acute{\rho}$
  $c[\![\grave{\Gamma}]\!](extract\ \grave{\pi}\grave{\rho})$
- **Introduce binary relations that both fit the domain constructors and reflect the intentions of the constraints.**
  $\hat{\pi}$ means $\langle \acute{\pi}, \grave{\pi}\rangle$
  $(c_{\hat{\pi}} \to c_{\hat{\pi}})\hat{\gamma}$ means $\forall \hat{\theta}.\ c_{\hat{\pi}}\hat{\theta} \Rightarrow c_{\hat{\pi}}\langle \acute{\gamma}\acute{\theta}, \grave{\gamma}\grave{\theta}\rangle$
- **Relate (or make assertions about) fragments through states.**
  $u_{\hat{\pi}}\hat{\rho} \Rightarrow (c_{\hat{\pi}} \to c_{\hat{\pi}})\ \langle \mathcal{C}[\![\acute{\Gamma}]\!]\acute{\rho}, \mathcal{C}[\![\grave{\Gamma}]\!]\grave{\rho}\rangle$
  $l_{\hat{\pi}}\hat{\alpha} \Rightarrow v_{\hat{\pi}\dagger\hat{\alpha}}\langle hold\ \acute{\alpha}\acute{\sigma}, hold\ \grave{\alpha}\grave{\sigma}\rangle$
- **Order states partially according to whether one extends another.**
  $\pi \le \pi'$ means $\exists \alpha.\ \pi = \pi' \dagger \alpha$ where $\pi' \dagger \alpha$ has no locations in the state π' "newer" than α.
- **Apply fragments in states that extend those for their definitions.**
  $\hat{\pi} \le \widehat{\pi'} \Rightarrow$
  $(c_{\hat{\pi}} \to c_{\hat{\pi}})\ \langle \mathcal{C}[\![\acute{\Gamma}]\!]\acute{\rho}, \mathcal{C}[\![\grave{\Gamma}]\!]\grave{\rho}\rangle \Rightarrow$
  $(c_{\widehat{\pi'}} \to c_{\widehat{\pi'}})\ \langle \mathcal{C}[\![\acute{\Gamma}]\!]\acute{\rho}, \mathcal{C}[\![\grave{\Gamma}]\!]\grave{\rho}\rangle$

Relationships for storage

In the current application, a store can be extracted from a state π by $store\ \pi$, with
$\forall\pi.\forall\pi'.\forall\alpha.\pi \le \pi' \Rightarrow area\ \alpha(store\ \pi) \Rightarrow area\ \alpha(store\ \pi')$
$\forall\pi.\forall\pi'.\forall\alpha.\pi \le \pi' \Rightarrow area\ \alpha(store\ \pi) \Rightarrow hold\ \alpha(store\ \pi) = hold\ \alpha(store\ \pi')$

$\pi: \mathbf{P}$	
$\alpha: \mathbf{L}$	$l_{\hat\pi}\hat\alpha = area\ \acute\alpha(store\ \acute\pi) \wedge area\ \grave\alpha(store\ \grave\pi)$
$\sigma: \mathbf{S}$	$s_{\hat\pi}\hat\sigma = \forall\hat\alpha.l_{\hat\pi}\hat\alpha \Rightarrow (area\ \acute\alpha\acute\sigma \wedge area\ \grave\alpha\grave\sigma) \wedge v_{\hat\pi\dagger\hat\alpha}\langle hold\ \acute\alpha\acute\sigma, hold\ \grave\alpha\grave\sigma\rangle$
$\beta: \mathbf{V} = \mathbf{B} + \mathbf{E}^* + \mathbf{F} + \mathbf{J}$	$v_{\hat\pi}\hat\beta = b_{\hat\pi} + e_{\hat\pi}{}^* + f_{\hat\pi} + j_{\hat\pi}$
$\beta: \mathbf{B}$	
$\phi: \mathbf{F} = \mathbf{E} \to \mathbf{C} \to \mathbf{C}$	$f_{\hat\pi}\hat\phi = \forall\widehat{\pi'}.\hat\pi \le \widehat{\pi'} \Rightarrow (e_{\widehat{\pi'}} \to c_{\widehat{\pi'}} \to c_{\widehat{\pi'}})\hat\phi$
$\theta: \mathbf{J} = \mathbf{C}$	$j_{\hat\pi}\hat\theta = \forall\widehat{\pi'}.\hat\pi \le \widehat{\pi'} \Rightarrow c_{\widehat{\pi'}}\hat\theta$
$\theta: \mathbf{C} = \mathbf{S} \to \mathbf{A}$	$c_{\hat\pi} = s_{\hat\pi} \to a_{\hat\pi}$
$o: \mathbf{A}$	
$\rho: \mathbf{U} = \mathbf{Ide} \to \mathbf{E}$	$u_{\hat\pi} = ide \to e_{\hat\pi}$
$\varepsilon: \mathbf{E} = \mathbf{L} + \mathbf{V}$	$e_{\hat\pi}\hat\varepsilon = (l_{\hat\pi} + v_{\hat\pi})\hat\varepsilon \wedge$
	$(\hat\varepsilon \in \mathbf{L}\times\mathbf{V} \Rightarrow area\ \acute\varepsilon(store\ \acute\pi) \wedge v_{\langle\acute\pi\dagger\acute\varepsilon,\grave\pi\rangle}\langle hold\ \acute\varepsilon(store\ \acute\pi), \grave\varepsilon\rangle) \wedge$
	$(\hat\varepsilon \in \mathbf{V}\times\mathbf{L} \Rightarrow area\ \grave\varepsilon(store\ \grave\pi) \wedge v_{\langle\acute\pi,\grave\pi\dagger\grave\varepsilon\rangle}\langle\acute\varepsilon, hold\ \grave\varepsilon(store\ \grave\pi)\rangle)$

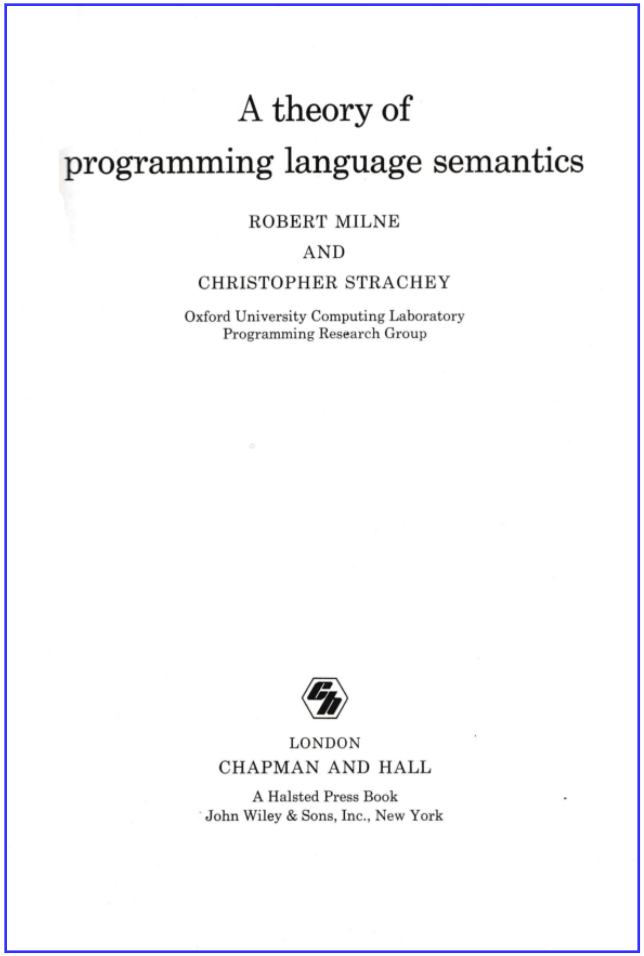
Most of the relations respect the ordering, in that if $\forall\hat\pi.\forall\widehat{\pi'}.\forall\hat\beta.\hat\pi \le \widehat{\pi'} \Rightarrow b_{\hat\pi}\hat\beta \Rightarrow b_{\widehat{\pi'}}\hat\beta$ then (for example) $\forall\hat\pi.\forall\widehat{\pi'}.\forall\hat\varepsilon.\hat\pi \le \widehat{\pi'} \Rightarrow e_{\hat\pi}\hat\varepsilon \Rightarrow e_{\widehat{\pi'}}\hat\varepsilon$.

Indeed, if $\forall\hat\pi.\forall\widehat{\pi'}.\forall\hat o.\hat\pi \le \widehat{\pi'} \Rightarrow a_{\hat\pi}\hat o \Rightarrow a_{\widehat{\pi'}}\hat o$ then $\forall\hat\pi.\forall\widehat{\pi'}.\forall\hat\theta.\hat\pi \le \widehat{\pi'} \Rightarrow c_{\hat\pi}\hat\theta \Rightarrow c_{\widehat{\pi'}}\hat\theta$.

However, $\forall\hat\pi.\forall\widehat{\pi'}.\forall\hat\sigma.\hat\pi \le \widehat{\pi'} \Rightarrow s_{\widehat{\pi'}}\hat\sigma \Rightarrow s_{\hat\pi}\hat\sigma$.

The constraint $consistent\ \chi\pi\rho$ requires that for all I that denote locations there is a monotonic mapping from χ to $\lambda\mathrm{I}.\pi \dagger \rho[\![\mathrm{I}]\!]$. If locations enter a store only in a sequence of new operations on an $empty$ store, then $extract\ \pi\rho[\![\mathrm{I}]\!]$ can signify the point in the sequence at which $\rho[\![\mathrm{I}]\!]$ enters; as $c[\![\Gamma]\!]$ depends only on the ordering of the values of $\chi[\![\mathrm{I}]\!]$ and $consistent\ (extract\ \pi\rho)\pi\rho$ holds, $extract\ \pi\rho$ can serve as χ.

Publishing the essay


A theory of programming language semantics

ROBERT MILNE

AND

CHRISTOPHER STRACHEY

Oxford University Computing Laboratory
Programming Research Group

LONDON
CHAPMAN AND HALL

A Halsted Press Book
John Wiley & Sons, Inc., New York

- **Motivations**
  - **Needing a coherent account of the developments.**
  - **Making the essay more widely accessible.**
  - **Bridging between theory and practice.**
- **Changes**
  - **Omission of personal historical remarks.**
  - **Inclusion of extra connections with other work.**
  - **Addition of more waymarking and explanation.**
- **Consequences**
  - **Paying for a possible visit to China (Barbara Halpern).**
  - **Ceasing involvement in the subject (Robert Milne).**

"I have managed to clear up my ideas on a number of points and am now even more convinced than before that we have a new branch of mathematics to deal with."

Christopher Strachey, letter to Leslie Fox, 1965.

The tens and twenties

1917

1921

1925