South African Computer Journal
Number 19
February 1997

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 19
Februarie 1997

Computer Science and Information Systems

Proceedings of WOFACS'96

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarverenging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof Lucas Introna
Department of Informatics
University of Pretoria
Hatfield 0083
lintrona@econ.up.ac.za

Production Editor
Dr Riël Smit
Mosaic Software (Pty) Ltd
P.O.Box 23906
Claremont 7735
gds@mosaic.co.za

World-Wide Web: http://www.mosaic.co.za/sacj/

Subscriptions

	Annual	Single copy
Southern Africa:	R50,00	R25,00
Elsewhere:	$30,00	$15,00

An additional $15 per year is charged for airmail outside Southern Africa

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Introduction

WOFACS '96: Workshop on Formal and Applied Computer Science

Chris Brink
Laboratory for Formal Aspects and Complexity in Computer Science, Department of Mathematics and Applied Mathematics, University of Cape Town
cbrink@maths.uct.ac.za

"What I tell you three times is true", said the Bellman in Lewis Carroll's *The Hunting of the Snark.* By somewhat the same principle, it is often held that three events of the same kind serve to make a series. And so, after WOFACS '92, '94 and now '96, we may claim to have a well-established biennial Southern African series of Workshops in the area of Formal Aspects of Computer Science.

This issue of the SACJ is devoted to the *Proceedings* of WOFACS '96. In other words, it contains survey articles written especially for this volume by the invited speakers (and their collaborators), on the topics they lectured on during the Workshop. These were:

- **Dr Maarten de Rijke** (University of Warwick): *Reasoning with Incomplete and Changing Information.*
- **Dr Holger Schlingloff** (Technische Universität München): *Verification of Finite-state Systems with Temporal Model Checking.*
- **Prof Jan Peleska** (Universität Bremen): *Test Automation for Safety-Critical Reactive Systems.*
- **Dr Jeff Sanders** (Oxford University): *Application-oriented Program Semantics.*

The format of WOFACS '96 followed the same pattern as before. Each speaker gave a course of 10 lectures, at a rate of one lecture per day. Material was pitched at about Honours level, and students had the opportunity of offering WOFACS courses for credit in their degree programmes at their respective home universities. Those who took up this option did some exercises and assignments and were evaluated by the speaker(s) concerned, thus gaining valuable insight into material, methods and expectations at an international level. WOFACS '96 was organised by FACCS-Lab (the Laboratory for Formal Aspects and Complexity in Computer Science), and was co-hosted by the Department of Mathematics and Applied Mathematics and the Department of Computer Science at the University of Cape Town. Accommodation was available in a University residence, and we were able to make available some financial support for travel and accommodation to participants (especially students) who could not obtain funding from their home institutions. Attendance stood at about 50 participants. Cape Town is a pleasant place to visit, even in winter, and we took care to have suitable outings and social events for our visitors. I am pleased to be able to mention that the WOFACS series has now attracted international attention, and that WOFACS 98 is being planned under the auspices of IFIP (the International Federation of Information Processing), specifically Working Group 2.3 on Programming Methodology.

No event of this nature can succeed without the hard work of a number of people. I would like to express my grateful thanks to:

- the invited speakers, for the care they took and the quality of their presentations;
- the Foundation for Research Development and the UCT Research Committee, for sponsorship;
- Diana Dixon, Jeanne Weir, Peter Jipsen and other FACCS-Lab staff members, for their hard work, and
- all participants, for attending.

SACJ is produced with kind support from
Mosaic Software (Pty) Ltd.

Application-Oriented Program Semantics

A K McIver Carroll Morgan J W Sanders
Programming Research Group, Oxford University, U K

Abstract

This paper abridges lecture notes from WOFACS 96. It provides semantic models for a variety of programming and development formalisms, showing how different models for the same formalism are related by Galois connections. The formalisms are: the language of guarded commands and its refinement calculus; communicating sequential processes; and probabilistic imperative code. The semantic models include: binary relations; predicate transformers; traces; failures; and distributions. In each case denotations of code are characterised, in the more general specification space, by the healthiness conditions they satisfy. Models are evaulated for their elegance, expressive power and calculational facility.

Expanded notes containing proofs, an extra chapter on data refinement and exercises for each chapter, are available from the authors.

Keywords: *Semantics, formal methods, refinement, data refinement, process algebra, probability, Galois connection, binary relations, predicate transformers.*
Computing Review Categories: *D.1, D.2, F.3, G.3.*

1 Introduction

It is usual for a course in program semantics to be either general (comparing the different styles of semantics: operational, denotational and algebraic) or specific, studying in one style the semantics of a programming language. Each type of course has its advantages and disadvantages. The former conveys an appreciation of the conditions under which each type of semantics is appropriate, by being broad though shallow. The latter is usually more pragmatic, concentrating on the details of a (typically obsolete) language at the expense of theory.

This course in *Application-Oriented Program Semantics* has been designed to combine the advantages of both those types of course, in a contemporary setting. It emphasises the uses of the semantics of imperative programming languages without concentrating exclusively on theory. For each language studied, an appropriate domain is introduced and its salient features characterised mathematically; Galois connections are used to relate the various semantic models. A feature is our insistence that semantic domains be broad enough to embrace not just programs but also the constructs which arise in program development (like miracles and data refinement).

Outline

After settling upon notation and studying what we need of Galois connections between partial orders, we begin the course proper by studying the intuitively compelling relational model of imperative programs. It provides denotations of code, specification constructs, and development rules including two rules for data refinement which are together complete. However use of weakest pre and post -specification for program development is not simple, so the model has limited applicability.

On investigating the lattice structure of the relational model we are led to study predicate transformers: the *premier* semantic model of imperative programs. The two models are related by a Galois connection, which imparts healthiness conditions to the predicate-transformer denotations of code. Predicate- transformers are shown to provide a model of the refinement calculus in which there is a single complete refinement rule. In the guise of the refinement calculus they support a viable development method.

Next, the notation CSP (for Communicating Sequential Processes) is summarised, and its three basic semantic models related by Galois connections. An important feature of the study is its use of both the maximal elements and the compact elements in its semantic domains. It provides further evidence for our underlying claims about the utility of Galois connections and the particular form of their constituent functions.

Finally a probabilistic lifting is investigated on semantic domains and applied to the study of some random algorithms! Properties of the probabilistic semantic domain are considered and probabilistic refinement rules demonstrated. This example is included to demonstrate the utility of the techniques, exposed in these lectures, in a novel situation.

The semantic domains which arise in this course are studied both in their own right, and as vehicles for conveying the aims of the course.

Aims

The general aims of this course are:

1. to further familiarity with the use of mathematical techniques (including algebra, category theory and topology) in Computer Science;
2. to observe how domains arise in different semantic contexts and to learn how to recognise them and their salient features (e.g. compact elements, maximal elements);

3. to understand how Galois connections are used to relate different semantic models;
4. to appreciate the four major semantic models studied in the course (relational, predicate-transformer, communicating-process and probabilistic), their relationship and individual characteristics;
5. to understand the manner in which the laws of a refinement calculus are proved sound in a semantic model and be able to translate that understanding to specific domains;
6. to be able to choose an appropriate semantic model for a given application, to identify an appropriate 'healthy' subdomain of it, and to relate it to alternatives.

2 Notation

The purpose of this chapter is to provide basic notation for what follows. Since most of the underlying concepts are expected to be familiar, the treatment is brief.

Sets and logic

The symbol $\widehat{=}$ means *equal by definition*. The *natural numbers*, *real numbers* and *Booleans* are denoted respectively by $\mathbf{N}$, $\mathbf{R}$ and $\mathbf{B}$. Intervals of natural numbers are expressed $\mathbf{a}\,..\,\mathbf{b} \;\widehat{=}\; \{\mathbf{n} : \mathbf{N} \mid \mathbf{a} \leq \mathbf{n} < \mathbf{b}\}$.

When the "universe of discourse" $\mathbf{X}$ is clear, if $\mathbf{E} \subseteq \mathbf{X}$ then $\overline{\mathbf{E}}$ means the complement of $\mathbf{E}$ in $\mathbf{X}$; otherwise we write it explicitly as $\mathbf{X} \setminus \mathbf{E}$.

If $\mathbf{X}$ is a finite set, $\#\mathbf{X}$ denotes its cardinality.

The predicate "*property* $\mathbf{q}$ *holds for all* $\mathbf{x}$ *of type* $\mathbf{X}$" is written $\forall \mathbf{x} : \mathbf{X} \bullet \mathbf{q}$. The set of functions from $\mathbf{X}$ to $\mathbf{Y}$ is denoted $\mathbf{X} \to \mathbf{Y}$ with one exception: we regard a predicate as a Boolean-valued function, and write $\mathbf{PX}$ for the set of all predicates on $\mathbf{X}$ $\mathbf{PX} \;\widehat{=}\; \mathbf{X} \to \mathbf{B}$. The characteristic function of a set $\mathbf{E} \subseteq \mathbf{X}$ is denoted $\xi_{\mathbf{E}} : \mathbf{PX}$.

When clarity requires it, the conjunction of two predicates, traditionally written *why* $\wedge$ *wherefore*, is written

$$\begin{pmatrix} \mathbf{why} \\ \mathbf{wherefore} \end{pmatrix}.$$

We use ternary infix notation for the conditional: for predicates $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{r}$,

$$\mathbf{p} \lhd \mathbf{r} \rhd \mathbf{q} \;\widehat{=}\; (\mathbf{p} \wedge \mathbf{r}) \vee (\mathbf{q} \wedge \neg \mathbf{r})$$

which is read "$\mathbf{p}$ if $\mathbf{r}$ else $\mathbf{q}$".

The remaining logical symbols have their usual meanings. However we write $\mathbf{p} \equiv \mathbf{q}$ iff $\mathbf{p} \Leftrightarrow \mathbf{q}$ is a theorem; similarly for $\mathbf{p} \Rrightarrow \mathbf{q}$ iff $\mathbf{p} \Rightarrow \mathbf{q}$ is a theorem. One advantage of that notation is that by identifying $\equiv$-equivalent formulae, $\Rrightarrow$ becomes antisymmetric (see the example at the end of this section) although implication is not. Another is that we can unambiguously "chain" predicates to reason. Following Dijkstra and Scholten we include a justification for each link in the chain of such a proof. For example:

$$\begin{aligned} & \mathbf{p} \lhd \mathbf{r} \rhd \mathbf{q} \\ \equiv & \qquad \text{propositional calculus} \\ & \begin{pmatrix} \mathbf{r} \Rightarrow \mathbf{p} \\ \neg \mathbf{r} \Rightarrow \mathbf{q} \end{pmatrix}. \end{aligned}$$

If $\mathbf{X}$ is a set, $\mathbf{seq.X}$ denotes the set of all finite sequences of elements of $\mathbf{X}$. Sequence comprehension is expressed using angle brackets so that, for example, the empty sequence is written $\langle\rangle$.

Relations and functions

A *relation* from $\mathbf{X}$ to $\mathbf{Y}$ is a predicate on the Cartesian product of $\mathbf{X}$ with $\mathbf{Y}$. Thus the set of all relations from $\mathbf{X}$ to $\mathbf{Y}$ is defined $\mathbf{X} \leftrightarrow \mathbf{Y} \;\widehat{=}\; \mathbf{P}(\mathbf{X} \times \mathbf{Y})$. Relations are usually written in *infix*, so that for $\mathbf{r} : \mathbf{X} \leftrightarrow \mathbf{Y}$, $\mathbf{x} : \mathbf{X}$ and $\mathbf{y} : \mathbf{Y}$, $\mathbf{r}$ relates $\mathbf{x}$ to $\mathbf{y}$ is written $\mathbf{x}\,\mathbf{r}\,\mathbf{y}$ (traditionally expressed $(\mathbf{x}, \mathbf{y}) \in \mathbf{r}$ with $\mathbf{r}$ viewed as a *subset* of $\mathbf{X} \times \mathbf{Y}$).

If $\mathbf{r} : \mathbf{X} \leftrightarrow \mathbf{Y}$ and $\mathbf{s} : \mathbf{Y} \leftrightarrow \mathbf{Z}$, their *forward relational composition* $\mathbf{r} \fatsemi \mathbf{s} : \mathbf{X} \leftrightarrow \mathbf{Z}$ is defined

$$\mathbf{x}\,(\mathbf{r} \fatsemi \mathbf{s})\,\mathbf{z} \;\widehat{=}\; \exists \mathbf{y} : \mathbf{Y} \bullet \begin{pmatrix} \mathbf{x}\,\mathbf{r}\,\mathbf{y} \\ \mathbf{y}\,\mathbf{s}\,\mathbf{z} \end{pmatrix}.$$

The *identity*, $\iota_{\mathbf{X}}$, and *universal relation*, $\omega_{\mathbf{X}}$, on $\mathbf{X}$ have their standard meanings: $\iota_{\mathbf{X}}$ relates every element of $\mathbf{X}$ to itself whilst $\omega_{\mathbf{X}}$ relates all elements of $\mathbf{X}$

$$\forall \mathbf{x} : \mathbf{X} \bullet \mathbf{x}\,\iota_{\mathbf{X}}\,\mathbf{x}$$

$$\forall \mathbf{x}, \mathbf{y} : \mathbf{X} \bullet \mathbf{x}\,\omega_{\mathbf{X}}\,\mathbf{y}.$$

The former gains its name by being the identity for relational composition. Further laws are investigated when we discuss relational semantics.

If $\mathbf{r} : \mathbf{X} \leftrightarrow \mathbf{Y}$ then $\mathbf{r}^{\sim} : \mathbf{Y} \leftrightarrow \mathbf{X}$ denotes the *converse* of $\mathbf{r}$, defined $\mathbf{y}\,\mathbf{r}^{\sim}\,\mathbf{x} \;\widehat{=}\; \mathbf{x}\,\mathbf{r}\,\mathbf{y}$.

A function can be viewed as a total relation $\mathbf{f}$ which at each point is single-valued:

$$\begin{pmatrix} \mathbf{x}\,\mathbf{f}\,\mathbf{y} \\ \mathbf{x}\,\mathbf{f}\,\mathbf{y}' \end{pmatrix} \Rrightarrow \mathbf{y} = \mathbf{y}'.$$

If $\mathbf{f} : \mathbf{X} \to \mathbf{Y}$ then we write $\mathbf{f.x} = \mathbf{y}$ instead of $\mathbf{x}\,\mathbf{f}\,\mathbf{y}$. Thus we avoid parentheses by writing $\mathbf{f.x}$ for the application of $\mathbf{f}$ to argument $\mathbf{x} : \mathbf{X}$. Functional application associates to the left. We shall also use notation $\mathbf{f}$ for the lifting of $\mathbf{f} : \mathbf{X} \to \mathbf{Y}$ to its action on subsets by defining, for $\mathbf{E} \subseteq \mathbf{X}$, $\mathbf{f.E} \;\widehat{=}\; \{\mathbf{f.e} \mid \mathbf{e} \in \mathbf{E}\}$. Ambiguity about the type of $\mathbf{f}$ will be avoided by context.

The functions *dom* and *ran* denote the domain and range, respectively, of a relation. Thus for $\mathbf{r} : \mathbf{X} \leftrightarrow \mathbf{Y}$,

$$\mathbf{dom.r} \;\widehat{=}\; \{\mathbf{x} : \mathbf{X} \mid \exists \mathbf{y} : \mathbf{Y} \bullet \mathbf{x}\,\mathbf{r}\,\mathbf{y}\}$$

$$\mathbf{ran.r} \;\widehat{=}\; \{\mathbf{y} : \mathbf{Y} \mid \exists \mathbf{x} : \mathbf{X} \bullet \mathbf{x}\,\mathbf{r}\,\mathbf{y}\}.$$

The set of *partial functions* from $\mathbf{X}$ to $\mathbf{Y}$ is denoted $\mathbf{X} \nrightarrow \mathbf{Y}$. For (partial) functions it is often convenient to exploit single-valuedness to rewrite relational composition in the standard way: if $\mathbf{f} : \mathbf{X} \nrightarrow \mathbf{Y}$ and $\mathbf{g} : \mathbf{Y} \nrightarrow \mathbf{Z}$, $\mathbf{g} \circ \mathbf{f}$ denotes

the *functional composition* $\mathbf{f}$ followed by $\mathbf{g}$, defined

$$(\mathbf{g} \circ \mathbf{f}).\mathbf{x} \;\widehat{=}\; \mathbf{g}.(\mathbf{f}.\mathbf{x}) \qquad (= (\mathbf{f} \fatsemi \mathbf{g}).\mathbf{x}) .$$

The converse $\mathbf{f}^{\sim}$ of a partial function $\mathbf{f}$ is usually called its *inverse*, although the result is again a partial function iff $\mathbf{f}$ is injective.

Order

$(\mathbf{X}, \leq)$ is *partially ordered* means that $\mathbf{X}$ is a nonempty set on which $\leq$ is a reflexive, antisymmetric and transitive relation:

$$\iota\mathbf{x} \subseteq \leq ,$$

$$(\leq \cap \geq) \subseteq \iota\mathbf{x} ,$$
where $\geq$ denotes the relational converse of $\leq$

$$(\leq \fatsemi \leq) \subseteq \leq .$$

If $(\mathbf{X}, \leq)$ is partially ordered then the relation $<$ denotes the *strict order* $< \;\widehat{=}\; (\leq \cap \neq)$. If $\mathbf{x} \leq \mathbf{y}$ in a partial order we say that $\mathbf{y}$ *dominates* $\mathbf{x}$; and if $\mathbf{x} < \mathbf{y}$ then we say that $\mathbf{y}$ dominates $\mathbf{x}$ *strictly*.

Basic examples of partial orders are provided by the standard number lines $(\mathbf{N}, \leq)$ and $(\mathbf{R}, \leq)$, and their subsets with the induced orderings. The most important example of the latter type is the two-element set $\mathbf{B}$ consisting of 0 (or *false*) and 1 (or *true*), with $0 < 1$; it is called the *Boolean* order.

Derived examples are provided by the product of a partial order with itself under the product order, and its extension: the set of all functions from a set $\mathbf{Y}$ into a set $\mathbf{X}$ where $(\mathbf{X}, \leq)$ is partially ordered, under the pointwise order

$$\mathbf{f} \leq \mathbf{g} \;\widehat{=}\; \forall \mathbf{y} : \mathbf{Y} \bullet \mathbf{f}.\mathbf{y} \leq \mathbf{g}.\mathbf{y} .$$

An important example of such a partial order is the set $\mathbf{PX}$ of all predicates on $\mathbf{X}$ under implication

$$\mathbf{p} \leq \mathbf{q} \;\widehat{=}\; \mathbf{p} \Rrightarrow \mathbf{q} .$$

That is thought of as expressing $\mathbf{p}$ *is stronger than* $\mathbf{q}$ or, equivalently, $\mathbf{q}$ *is weaker than* $\mathbf{p}$.

The space $(\mathbf{PX}, \leq)$ is isomorphic to the space of all subsets of $\mathbf{X}$ with the inclusion ordering, under the isotone bijection which assigns to predicate $\mathbf{p}$ the set of all elements satisfying $\mathbf{p}$.

Extrema

Assume that $(\mathbf{X}, \leq)$ is partially ordered.

If $\mathbf{x} : \mathbf{X}$ and $\mathbf{E} \subseteq \mathbf{X}$ then $\mathbf{x}$ is a *lower bound* of $\mathbf{E}$ means that it is dominated by each member of $\mathbf{E}$

$$\forall \mathbf{e} : \mathbf{E} \bullet \mathbf{x} \leq \mathbf{e} .$$

A *least element* of $\mathbf{E}$ is a lower bound which belongs to $\mathbf{E}$; if it exists, it is unique (why?).

A *greatest lower bound*, or glb, or *infimum*, of $\mathbf{E}$ is a lower bound of $\mathbf{E}$ which dominates any lower bound of $\mathbf{E}$. Equivalently, but preferable for calculation, $\mathbf{x}$ is a greatest lower bound of $\mathbf{E}$ iff

$$\mathbf{y} \leq \mathbf{x} \equiv \forall \mathbf{e} : \mathbf{E} \bullet \mathbf{y} \leq \mathbf{e} .$$

If it exists a glb of $\mathbf{E}$ is unique; it is written $\sqcap\mathbf{E}$. Thus

$$\mathbf{y} \leq \sqcap\mathbf{E} \equiv \forall \mathbf{e} : \mathbf{E} \bullet \mathbf{y} \leq \mathbf{e} .$$

Similarly are defined *upper bound*, *greatest element*, *least upper bound* (or lub, or supremum):

$$\sqcup\mathbf{E} \leq \mathbf{y} \equiv \forall \mathbf{e} : \mathbf{E} \bullet \mathbf{e} \leq \mathbf{y} .$$

Observe that in order to reduce reasoning about $\sqcap$ to reasoning about $\sqcup$,

$$\sqcap\mathbf{E} = \sqcup\{\mathbf{x} : \mathbf{X} \mid \forall \mathbf{e} : \mathbf{E} \bullet \mathbf{x} \leq \mathbf{e}\}$$

(and analogously for $\sqcup$ in terms of $\sqcap$).

Completeness

Partially ordered $(\mathbf{X}, \leq)$ is a *lattice* means that each nonempty finite subset of $\mathbf{X}$ has a glb and a lub. Equivalently, the binary versions of $\sqcap$ and $\sqcup$ are idempotent, commutative, associative and absorptive.

A lattice is *complete* means that every subset has a glb and lub. In particular a complete lattice has a least element, $\sqcap\mathbf{X}$, and a greatest element, $\sqcup\mathbf{X}$. The former is denoted $\bot$ and the latter $\top$.

A *directed* subset of partially ordered $(\mathbf{X}, \leq)$ means a subset $\mathbf{E}$ of $\mathbf{X}$ of which every finite subset has an upper bound in $\mathbf{E}$. $(\mathbf{X}, \leq)$ is a *complete partial order*, or cpo, means every directed subset of $\mathbf{X}$ has a lub.

An element $\mathbf{x} : \mathbf{X}$ of a cpo is *compact* means that for any directed set $\mathbf{E} \subseteq \mathbf{X}$, if the lub of $\mathbf{E}$ dominates $\mathbf{x}$ then some member of $\mathbf{E}$ dominates $\mathbf{x}$

$$\mathbf{x} \leq \sqcup\mathbf{E} \Rrightarrow \exists \mathbf{e} : \mathbf{E} \bullet \mathbf{x} \leq \mathbf{e} .$$

The set of all compact elements of $(\mathbf{X}, \leq)$ is denoted $\mathbf{K}(\mathbf{X})$.

A cpo is a *domain*, or *algebraic*, means each element is the lub of the compact elements it dominates

$$\forall \mathbf{x} : \mathbf{X} \bullet \mathbf{x} = \sqcup\{\mathbf{y} : \mathbf{K}(\mathbf{X}) \mid \mathbf{y} \leq \mathbf{x}\} .$$

Morphisms

Suppose that $(\mathbf{X}, \leq)$ and $(\mathbf{Y}, \leq)$ are partially ordered; their orders, glb's, lub's, etc. will be distinguished by context (rather than by say subscripts).

A function $\mathbf{f} : \mathbf{X} \to \mathbf{Y}$ is *monotone* means that if $\mathbf{x}'$ dominates $\mathbf{x}$ in $\mathbf{X}$ then $\mathbf{f}.\mathbf{x}'$ dominates $\mathbf{f}.\mathbf{x}$ in $\mathbf{Y}$

$$\mathbf{x} \leq \mathbf{x}' \Rrightarrow \mathbf{f}.\mathbf{x} \leq \mathbf{f}.\mathbf{x}' .$$

$\mathbf{f}$ is *isotone* means $\mathbf{x} \leq \mathbf{x}' \equiv \mathbf{f}.\mathbf{x} \leq \mathbf{f}.\mathbf{x}'$.

Now assume that $(\mathbf{X}, \leq)$ is a complete lattice and con-

sider

$$\mathbf{f}.\sqcup \mathbf{E} \;=\; \sqcup \mathbf{f}.\mathbf{E}\,. \qquad (1)$$

Then $\mathbf{f}$ is

- *universally* $\sqcup$*-junctive* means (1) holds for all $\mathbf{E} \subseteq \mathbf{X}$;
- *positively* $\sqcup$*-junctive* means (1) holds for all nonempty $\mathbf{E} \subseteq \mathbf{X}$; and
- $\sqcup$*-junctive* means (1) holds for all nonempty finite $\mathbf{E} \subseteq \mathbf{X}$.

Similarly for $\sqcap$-junctivity.

If $(\mathbf{X}, \leq)$ is a cpo then $\mathbf{f} : \mathbf{X} \to \mathbf{Y}$ is *continuous* means $\mathbf{f}$ is monotone and (1) holds for each directed $\mathbf{E} \subseteq \mathbf{X}$.

Fixed points

If $\mathbf{f} : \mathbf{X} \to \mathbf{X}$ then $\mathbf{x} : \mathbf{X}$ is a *fixed point of* $\mathbf{f}$ means

$$\mathbf{f}.\mathbf{x} = \mathbf{x}\,.$$

If $(\mathbf{X}, \leq)$ is a cpo with least element $\bot$ and $\mathbf{f} : \mathbf{X} \to \mathbf{X}$ is monotone then $\mathbf{f}$ has a least fixed point $\mu\mathbf{f} : \mathbf{X}$

$$\left(\begin{array}{l} \mathbf{f}.\mu\mathbf{f} = \mu\mathbf{f} \\ \forall \mathbf{x} : \mathbf{X} \bullet \mathbf{x} = \mathbf{f}.\mathbf{x} \;\Rightarrow\; \mu\mathbf{f} \leq \mathbf{x} \end{array} \right).$$

(Proof is by transfinite induction.) If moreover $\mathbf{f}$ is continuous then $\mu\mathbf{f} = \sqcup\{\mathbf{f}^{\mathbf{n}}.\ \bot \mid \mathbf{n} : \mathbf{N}\}$; that is, the least fixed point is attained over the natural numbers.

Complements and conjugation

A lattice $(\mathbf{X}, \leq)$ *is complemented* means firstly that it has greatest and least elements, and secondly that each element of the lattice has a complement

$$\forall \mathbf{x} : \mathbf{X} \bullet \exists \mathbf{y} : \mathbf{X} \bullet \left(\begin{array}{l} \mathbf{x} \sqcup \mathbf{y} = \top \\ \mathbf{x} \sqcap \mathbf{y} = \bot \end{array} \right).$$

Complement $\mathbf{y}$ is unique, for each $\mathbf{x}$, if the lattice is *distributive*

$$\left(\begin{array}{l} \mathbf{x} \sqcup (\mathbf{y} \sqcap \mathbf{z}) = (\mathbf{x} \sqcup \mathbf{y}) \sqcap (\mathbf{x} \sqcup \mathbf{z}) \\ \mathbf{x} \sqcap (\mathbf{y} \sqcup \mathbf{z}) = (\mathbf{x} \sqcap \mathbf{y}) \sqcup (\mathbf{x} \sqcap \mathbf{z}) \end{array} \right),$$

in which case the complement of $\mathbf{x}$ is written $\neg\mathbf{x}$. A complemented distributive lattice is called a *Boolean algebra*.

In a Boolean algebra each function $\mathbf{f} : \mathbf{X} \to \mathbf{X}$ has *conjugate* $\mathbf{f}^* : \mathbf{X} \to \mathbf{X}$ defined $\mathbf{f}^*.\mathbf{x} \;\widehat{=}\; \neg\mathbf{f}.\neg\mathbf{x}$. Evidently the least element $\bot$ of $\mathbf{X}$ is a fixed point of $\mathbf{f}$ iff the greatest element $\top$ is a fixed point of $\mathbf{f}^*$.

Example

Predicates, since they express constraints between observables, are fundamental for describing (discrete) systems. The obvious order for our purposes is that which corresponds to the "implements" relation on descriptions. Thus if $\mathbf{p}$ and $\mathbf{q}$ are predicates whose free variables represent observations of a system then $\mathbf{p} \leq \mathbf{q}$ is the proof obligation for showing that description $\mathbf{p}$ meets specification $\mathbf{q}$ (or that $\mathbf{p}$ is a refinement of $\mathbf{q}$). Since it holds iff every observation described by $\mathbf{p}$ is an observation specified by $\mathbf{q}$, we have

$$\mathbf{p} \leq \mathbf{q} \;\widehat{=}\; \mathbf{p} \Rrightarrow \mathbf{q}\,.$$

Of course that ordering coincides with the pointwise ordering on predicates as elements of $\mathbf{X} \to \mathbf{B}$.

We consider the structure of the space of predicates $(\mathbf{PX}, \leq)$. It is partially ordered, as already observed, since if $(\mathbf{Y}, \leq)$ is partially ordered and $\mathbf{X}$ is a set then $\mathbf{X} \to \mathbf{Y}$ is partially ordered with the pointwise-lifting order. Also it is a complete Boolean algebra with: least element the predicate which is always false; greatest element the predicate which is always true; negation as complement; disjunction as least upper bound; and conjunction as greatest lower bound.

It is a cpo since a set of predicates $\mathbf{Q} \subseteq \mathbf{PX}$ has as lub the predicate $\mathbf{p}$ for which

$$\mathbf{p}.\mathbf{x} \;\widehat{=}\; \vee\{\mathbf{q}.\mathbf{x} \mid \mathbf{q} \in \mathbf{Q}\};$$

thus $\mathbf{Q}$ need not be directed.

A predicate is compact iff it is the characteristic function of a *finite* subset of $\mathbf{X}$. Indeed for $\mathbf{E} \subseteq \mathbf{X}$ and set $\mathbf{Q}$ of predicates,

$$\begin{array}{ll} & \xi_{\mathbf{E}} \leq \sqcup\mathbf{Q} \\ \equiv & \text{definitions of order and sup} \\ & \forall \mathbf{x} : \mathbf{X} \bullet \xi_{\mathbf{E}}.\mathbf{x} \leq \vee\{\mathbf{q}.\mathbf{x} \mid \mathbf{q} \in \mathbf{Q}\} \\ \equiv & \text{definitions of characteristic function and } \vee \\ & \forall \mathbf{x} : \mathbf{X} \bullet \mathbf{x} \in \mathbf{E} \Rightarrow \exists \mathbf{q} : \mathbf{Q} \bullet \mathbf{q}.\mathbf{x}\,. \end{array}$$

Now if $\mathbf{E}$ is finite that implies, if $\mathbf{Q}$ is directed,

$$\begin{array}{ll} & \exists \mathbf{q} : \mathbf{Q} \bullet \forall \mathbf{x} : \mathbf{X} \bullet \mathbf{x} \in \mathbf{E} \Rightarrow \mathbf{q}.\mathbf{x} \\ \equiv & \text{definitions of characteristic function and order} \\ & \exists \mathbf{q} : \mathbf{Q} \bullet \xi_{\mathbf{E}} \leq \mathbf{q}, \end{array}$$

as required for compactness of $\xi_{\mathbf{E}}$. But conversely if $\mathbf{E}$ is infinite, letting $\mathcal{F}$ denote a family of increasing finite subsets whose union equals $\mathbf{X}$ (why does $\mathcal{F}$ exist?), the previous reasoning applies to the increasing (and hence directed) set $\mathbf{Q} \;\widehat{=}\; \{\xi_{\mathbf{E} \cap \mathbf{F}} \mid \mathbf{F} \in \mathcal{F}\}$ to show

$$\left(\begin{array}{l} \xi_{\mathbf{E}} \leq \sqcup\mathbf{Q} \\ \neg \exists \mathbf{q} : \mathbf{Q} \bullet \xi_{\mathbf{E}} \leq \mathbf{q} \end{array} \right);$$

thus $\xi_{\mathbf{E}}$ is not compact.

Finally $(\mathbf{PX}, \leq)$ is a domain. For if $\mathbf{F} : \mathcal{F}$ and $\mathbf{p} : \mathbf{PX}$, letting $\mathbf{p}_{\mathbf{F}} \;\widehat{=}\; \mathbf{p} \wedge \xi_{\mathbf{F}}$, then for each $\mathbf{F}$,

$$\left(\begin{array}{l} \mathbf{p}_{\mathbf{F}} \leq \mathbf{p} \\ \mathbf{p}_{\mathbf{F}} \text{ compact} \\ \mathbf{p} = \sqcup\{\mathbf{p}_{\mathbf{F}} \mid \mathbf{F} \in \mathcal{F}\} \end{array} \right).$$

Thus $\mathbf{p}$ is the lub of the compact elements it dominates. Since $\mathbf{p}$ was arbitrary, $(\mathbf{PX}, \leq)$ is a domain.

We return to this example in section 3. In the meantime we have established:

Theorem(predicates). If $\mathbf{X}$ is a set then the

space $(\mathbf{PX}, \leq)$ of predicates on $\mathbf{X}$ with the pointwise order forms a complete Boolean algebra which is a domain whose compact elements are the characteristic functions of finite sets.

3 Galois connections

Introduction

We wish to compare different semantic models of the same formal method. In this chapter we study our main technique: Galois connections.

We consider the basic properties and examples of such connections, concentrating on their dual advantages:

- the theoretical properties Galois connections establish;
- the support Galois connections provide for calculation.

But we begin by motivating their definition.

Motivation

Two partially ordered spaces $(\mathbf{X}, \leq)$ and $(\mathbf{Y}, \leq)$ are *isomorphic* means there is an isotone bijection $\phi : \mathbf{X} \to \mathbf{Y}$ between them; that is, a bijection for which

$$\mathbf{x} \leq \mathbf{x}' \equiv \phi.\mathbf{x} \leq \phi.\mathbf{x}'.$$

Equivalently, since ϕ has a well-defined inverse $\phi^\sim$, choosing the unique $\mathbf{y} : \mathbf{Y}$ for which $\mathbf{x}' = \phi^\sim.\mathbf{y}$,

$$\begin{array}{ll} & \mathbf{x} \leq \phi^\sim.\mathbf{y} \\ \equiv & \quad \text{by choice of } \mathbf{y} \\ & \mathbf{x} \leq \mathbf{x}' \\ \equiv & \quad \text{equivalence above} \\ & \phi.\mathbf{x} \leq \phi.\mathbf{x}' \\ \equiv & \quad \text{in terms of } \mathbf{y} \text{ again} \\ & \phi.\mathbf{x} \leq \phi.\phi^\sim.\mathbf{y} \\ \equiv & \quad \phi \circ \phi^\sim = \iota \\ & \phi.\mathbf{x} \leq \mathbf{y}\,. \end{array}$$

Thus $\mathbf{x} \leq \phi^\sim.\mathbf{y} \equiv \phi.\mathbf{x} \leq \mathbf{y}$.

We are interested in pairs of semantic spaces of which one is "finer" or "contains more information" than the other. In that case ϕ is merely a monotone injection and $\phi^\sim$ is replaced by a monotone projection π, with the same equivalence

$$\mathbf{x} \leq \pi.\mathbf{y} \equiv \phi.\mathbf{x} \leq \mathbf{y}\,.$$

Here are two examples, one from arithmetic and the other from program semantics, which support a definition of that form. Captured in these simple examples is much of the theory to follow.

Floor

The partially-ordered space $(\mathbf{R}, \leq)$ of real numbers with its standard order is finer than the partially-ordered space $(\mathbf{Z}, \leq)$ of integers. The latter is embedded in the former by the natural monotone injection

$$\varepsilon : \mathbf{Z} \to \mathbf{R}$$

$$\varepsilon.\mathbf{n} = \mathbf{n}\,,$$

in which $\mathbf{n}$ on the right-hand side is regarded as a real number. But there are many monotone injections of $\mathbf{Z}$ into $\mathbf{R}$; the natural one is characterised by having as a kind of inverse the *floor*, or integer-part-of, function $\lfloor\ \rfloor : \mathbf{R} \to \mathbf{Z}$, traditionally defined

$$\lfloor \mathbf{r} \rfloor \ \widehat{=}\ \text{the (unique) integer } \mathbf{n} \text{ for which } \mathbf{n} \leq \mathbf{r} < \mathbf{n} + 1\,.$$

Equivalently,

$$\lfloor \mathbf{r} \rfloor = \sqcup\{\mathbf{n} : \mathbf{Z} \mid \varepsilon.\mathbf{n} \leq \mathbf{r}\}\,. \tag{2}$$

Those definitions have a third equivalent form (why is it equivalent?) which is superior for calculations:

$$\mathbf{n} \leq \lfloor \mathbf{r} \rfloor \equiv \varepsilon.\mathbf{n} \leq \mathbf{r}\,. \tag{3}$$

All of those equations express that $\lfloor \mathbf{r} \rfloor$ is the greatest integer which is at most $\mathbf{r}$. Although it is clear that the definition of π characterises ε amongst monotone embemddings of $\mathbf{Z}$ in $\mathbf{R}$, it is surprising (at first sight) that it is in fact determined by ε and that equation (2) follows from equation (3); see the treatment at the end of this section.

Now $\lfloor\ \rfloor$ is monotone and inverse to ε in this sense

$$\forall \mathbf{n} : \mathbf{Z} \bullet \lfloor \varepsilon.\mathbf{n} \rfloor = \mathbf{n}\,.$$

Of course the dual identity fails, since $\varepsilon.\lfloor \mathbf{r} \rfloor \neq \mathbf{r}$ unless $\mathbf{r} \in \mathbf{ran}\ \varepsilon$. Fortunately equivalence (3) demonstrates the sense in which ε and $\lfloor\ \rfloor$ are inverse; indeed in spite of the previous inequality it follows from (3) that $\varepsilon.\lfloor \mathbf{r} \rfloor \leq \mathbf{r}$.

In summary,

$$\left(\begin{array}{l} \varepsilon : \mathbf{Z} \to \mathbf{R} \text{ monotone} \\ \lfloor\ \rfloor : \mathbf{R} \to \mathbf{Z} \text{ monotone} \\ \lfloor\ \rfloor \circ \varepsilon = \iota_{\mathbf{Z}} \\ \varepsilon \circ \lfloor\ \rfloor \leq \iota_{\mathbf{R}} \end{array} \right)\,. \tag{4}$$

We shall see (section 3) that most of those conditions follow simply from equivalence (3); and the one that does not is of independent interest.

We now demonstrate the calculational facility of equivalence (3) (following Aarts) by proving

$$\forall \mathbf{r} : \mathbf{R} \bullet \mathbf{r} \geq 0 \Rightarrow \lfloor \sqrt{\lfloor \mathbf{r} \rfloor} \rfloor = \lfloor \sqrt{\mathbf{r}} \rfloor\,.$$

The layout of the following proof is to be compared with that, essentially equivalent but expressed without a Galois connection, in Graham, Knuth and Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison Wesley, 1989, result 3.9.

Proof. Recall that for $\mathbf{r} : \mathbf{R}$, if $\mathbf{r} \geq 0$ then

$$\sqrt{\mathbf{r}} = \mathbf{x} \equiv \mathbf{r} = \mathbf{x}^2\,.$$

Of course $\mathbf{x}$ may be negative, so there may be two solutions $\mathbf{x}$. For such $\mathbf{r}$ we reason as follows, being slightly more careful about extracting the root of only non-negative *real* numbers:

$$\begin{array}{ll}
& \mathbf{n} \leq \lfloor \sqrt{\varepsilon}.\lfloor \mathbf{r} \rfloor \rfloor \\
\equiv & \qquad \mathbf{gc}(\varepsilon, \lfloor\ \rfloor; \mathbf{Z}, \mathbf{R}) \\
& \varepsilon.\mathbf{n} \leq \sqrt{\varepsilon}.\lfloor \mathbf{r} \rfloor \\
\equiv & \qquad \mathbf{r} \leq \sqrt{\mathbf{s}} \equiv (\mathbf{r} \geq 0 \Rightarrow \mathbf{r}^2 \leq \mathbf{s}) \\
& \varepsilon.\mathbf{n} \geq 0 \Rightarrow (\varepsilon.\mathbf{n})^2 \leq \varepsilon.\lfloor \mathbf{r} \rfloor \\
\equiv & \qquad (\varepsilon.\mathbf{n})^2 \in \mathbf{Z} \\
& \varepsilon.\mathbf{n} \geq 0 \Rightarrow \varepsilon.(\varepsilon.\mathbf{n})^2 \leq \varepsilon.\lfloor \mathbf{r} \rfloor \\
\equiv & \qquad \varepsilon.\mathbf{m} \leq \varepsilon.\mathbf{n} \equiv \mathbf{m} \leq \mathbf{n} \\
& \varepsilon.\mathbf{n} \geq 0 \Rightarrow (\varepsilon.\mathbf{n})^2 \leq \lfloor \mathbf{r} \rfloor \\
\equiv & \qquad \mathbf{gc}(\varepsilon, \lfloor\ \rfloor; \mathbf{Z}, \mathbf{R}) \text{ again} \\
& \varepsilon.\mathbf{n} \geq 0 \Rightarrow \varepsilon.(\varepsilon.\mathbf{n})^2 \leq \mathbf{r} \\
\equiv & \qquad \varepsilon.(\varepsilon.\mathbf{n})^2 = (\varepsilon.\mathbf{n})^2 \\
& \varepsilon.\mathbf{n} \geq 0 \Rightarrow (\varepsilon.\mathbf{n})^2 \leq \mathbf{r} \\
\equiv & \qquad \mathbf{r} \leq \sqrt{\mathbf{s}} \equiv (\mathbf{r} \geq 0 \Rightarrow \mathbf{r}^2 \leq \mathbf{s}) \text{ again} \\
& \varepsilon.\mathbf{n} \leq \sqrt{\mathbf{r}} \\
\equiv & \qquad \mathbf{gc}(\varepsilon, \lfloor\ \rfloor; \mathbf{Z}, \mathbf{R}) \text{ one last time} \\
& \mathbf{n} \leq \lfloor \sqrt{\mathbf{r}} \rfloor .
\end{array}$$

Though protracted, that proof requires little motivation and so is readily followed and equally readily constructed. Its use of Galois-connection *clichés* (like switching between the sides of the Galois equivalence) distinguishes it from the proof of Graham *et al.* in which mathematical creativity is seen as playing an important role. It is for such reasons that we exploit the calculational facility of Galois connections.

Pre- and post-conditions

If $\mathbf{X}$ is a set representing the state space (including input) on which a program, say $\mathbf{r}$, operates and $\mathbf{Y}$ is a set representing the state space in which the result (including output) lies, then $\mathbf{r}$ may be viewed as a relation from $\mathbf{X}$ to $\mathbf{Y}$. As a predicate transformer there are (at least) two interpretations of $\mathbf{r}$

- $\mathbf{wp.r} : \mathbf{PY} \to \mathbf{PX}$ gives the *weakest precondition* of its argument. Thus for *postcondition*, (or predicate on final states) $\mathbf{q} : \mathbf{PY}$, it holds at just those states of $\mathbf{X}$ from which execution of $\mathbf{r}$ terminates in a state satisfying $\mathbf{q}$

$$\mathbf{wp.r.q.x} \ \widehat{=}\ \forall \mathbf{y} : \mathbf{Y} \bullet \mathbf{x}\,\mathbf{r}\,\mathbf{y} \Rightarrow \mathbf{q.y} .$$

- $\mathbf{sp.r} : \mathbf{PX} \to \mathbf{PY}$ gives the *strongest postcondition* of its argument. Thus for *precondition*, (or predicate on initial states) $\mathbf{p} : \mathbf{PX}$, it holds at just those states of $\mathbf{Y}$ which can be reached if execution of $\mathbf{r}$ starts in a state satisfying $\mathbf{p}$

$$\mathbf{sp.r.p.y} \ \widehat{=}\ \exists \mathbf{x} : \mathbf{X} \bullet \begin{pmatrix} \mathbf{x}\,\mathbf{r}\,\mathbf{y} \\ \mathbf{p.x} \end{pmatrix} .$$

Now equivalence of those two representations of $\mathbf{r}$ is expressed: $\mathbf{p} \Rightarrow \mathbf{wp.r.q} \ \equiv\ \mathbf{sp.r.p} \Rightarrow \mathbf{q}$.

In summary, for $\mathbf{r} : \mathbf{X} \leftrightarrow \mathbf{Y}$,

$$\begin{pmatrix} \mathbf{sp.r} : \mathbf{PX} \to \mathbf{PY} \\ \mathbf{wp.r} : \mathbf{PY} \to \mathbf{PX} \\ \mathbf{p} \leq \mathbf{wp.r.q} \ \equiv\ \mathbf{sp.r.p} \leq \mathbf{q} \end{pmatrix} .$$

As we shall see in section 7, important properties of $\mathbf{wp.r}$ and $\mathbf{sp.r}$ follow directly from that equivalence.

These considerations lead us to the following.

Definition

Suppose that $(\mathbf{X}, \leq)$ and $(\mathbf{Y}, \leq)$ are partially ordered with $\varepsilon : \mathbf{X} \to \mathbf{Y}$ and $\pi : \mathbf{Y} \to \mathbf{X}$. Then the pair ε, π forms a *Galois connection*, written $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$, means

$$\forall \mathbf{x} : \mathbf{X} \bullet \forall \mathbf{y} : \mathbf{Y} \bullet \mathbf{x} \leq \pi.\mathbf{y} \ \Leftrightarrow\ \varepsilon.\mathbf{x} \leq \mathbf{y} .$$

In that case ε is called the *embedding* and π the *projection*; each is described (loosely) as being adjoint to the other.

The next two results explore basic properties of the set $\mathbf{gc(X, Y)}$ of all Galois connections between par- tially-ordered spaces $\mathbf{X}$ and $\mathbf{Y}$.

Galois connections act as morphisms between partially-ordered spaces:

Theorem(combining connections). The identity (and its converse) form a Galois connection on any partially-ordered space:

$$\mathbf{gc}(\iota_{\mathbf{X}}, \iota_{\mathbf{X}}; \mathbf{X}, \mathbf{X}) .$$

Galois connections compose as follows:

$$\begin{array}{l} \begin{pmatrix} \mathbf{gc}(\varepsilon 0, \pi 0; \mathbf{X}, \mathbf{Y}) \\ \mathbf{gc}(\varepsilon 1, \pi 1; \mathbf{Y}, \mathbf{Z}) \end{pmatrix} \\ \Rrightarrow \\ \mathbf{gc}(\varepsilon 1 \circ \varepsilon 0, \pi 0 \circ \pi 1; \mathbf{X}, \mathbf{Z}) . \end{array}$$

The space of Galois connections inherits order properties:

Theorem(ordering connections). If $\mathbf{X}$ and $\mathbf{Y}$ are partially ordered so too is $\mathbf{gc(X, Y)}$, under

$$(\varepsilon 0, \pi 0) \leq (\varepsilon 1, \pi 1) \ \widehat{=}\ \begin{pmatrix} \varepsilon 0 \leq \varepsilon 1 \\ \pi 1 \leq \pi 0 \end{pmatrix} .$$

If $\mathbf{X}$ and $\mathbf{Y}$ are lattices then so too is $\mathbf{gc(X, Y)}$ with: minimum $(\bot, \top)$; maximum $(\top, \bot)$; and with $(\varepsilon 0, \pi 0)$ and $(\varepsilon 1, \pi 1)$ having glb $(\varepsilon 0 \sqcap \varepsilon 1, \pi 0 \sqcup \pi 1)$ and lub $(\varepsilon 0 \sqcup \varepsilon 1, \pi 0 \sqcap \pi 1)$.

In the remaining sections of this chapter we investigate the more detailed structure exhibited by a single Galois connection.

Duality

The following result will enable us to expend half as much effort in proving subsequent results. Indeed it establishes the two steps in the previous proof that were justified as "similarly".

Theorem(duality). Writing $\mathbf{X}^{\sim}$ for the partial order $(\mathbf{X}, \geq)$ converse to $(\mathbf{X}, \leq)$,

$$\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y}) \equiv \mathbf{gc}(\pi, \varepsilon; \mathbf{Y}^{\sim}, \mathbf{X}^{\sim}) .$$

Equivalence

We now head towards characterising a Galois connection in terms that have already arisen in the first motivating example.

Lemma(weak inversion). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then, in terms of the pointwise ordering on functions (see section 2),

$$\left(\begin{array}{l} \iota_{\mathbf{X}} \leq \pi \circ \varepsilon \\ \varepsilon \circ \pi \leq \iota_{\mathbf{Y}} \end{array} \right) .$$

Lemma(monotonicity). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then ε and π are monotone.

The result we have been aiming at is

Theorem(equivalence).

$$\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y}) \equiv \left(\begin{array}{l} \varepsilon, \pi \text{ monotone} \\ \iota_{\mathbf{X}} \leq \pi \circ \varepsilon \\ \varepsilon \circ \pi \leq \iota_{\mathbf{Y}} \end{array} \right) .$$

Isomorphism

We now work towards establishing that in a Galois connection the ranges of the two functions are isomorphic.

Lemma(trading). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then these are equivalent

$$\begin{array}{l} \mathbf{x} \leq \pi.\mathbf{y} \\ \varepsilon.\mathbf{x} \leq \varepsilon.\pi.\mathbf{y} \\ \varepsilon.\mathbf{x} \leq \mathbf{y} \\ \pi.\varepsilon.\mathbf{x} \leq \pi.\mathbf{y} . \end{array}$$

Corollary (of trading). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then these are equivalent

$$\begin{array}{l} \pi.\mathbf{y}' \leq \pi.\mathbf{y} \\ \varepsilon.\pi.\mathbf{y}' \leq \varepsilon.\pi.\mathbf{y} \\ \varepsilon.\pi.\mathbf{y}' \leq \mathbf{y} . \end{array}$$

In particular $\pi.\mathbf{y}' = \pi.\mathbf{y} \equiv \varepsilon.\pi.\mathbf{y}' = \varepsilon.\pi.\mathbf{y}$. Dually, these are also equivalent

$$\begin{array}{l} \varepsilon.\mathbf{x} \leq \varepsilon.\mathbf{x}' \\ \pi.\varepsilon.\mathbf{x} \leq \pi.\varepsilon.\mathbf{x}' \\ \mathbf{x} \leq \pi.\varepsilon.\mathbf{x}' . \end{array}$$

In particular $\varepsilon.\mathbf{x} = \varepsilon.\mathbf{x}' \equiv \pi.\varepsilon.\mathbf{x} = \pi.\varepsilon.\mathbf{x}'$.

Lemma(sandwiching). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then

$$\left(\begin{array}{l} \varepsilon = \varepsilon \circ \pi \circ \varepsilon \\ \pi = \pi \circ \varepsilon \circ \pi \end{array} \right) .$$

The aim of the last few results has been the next theorem. It is to be compared with the case of an isotone bijection ϕ between partial orders $\mathbf{X}$ and $\mathbf{Y}$: the range of ϕ is isomorphic to its domain. Hence the range of ϕ is isomorphic to the range of its converse $\phi^{\sim}$, and ϕ itself provides an isotone bijection. That rather trivial observation is extended, to Galois connections, in the next result.

Theorem(isomorphism). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then $\mathbf{ran}.\varepsilon$ and $\mathbf{ran}.\pi$ are isomorphic and the restrictions of ε and π are isotone bijections.

The isomorphism theorem has an interesting and simple restatement in terms of fixed points.

Definition

If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then $\mathbf{x} : \mathbf{X}$ is *closed* means it is a fixed point of $\pi \circ \varepsilon$. $\mathbf{X}_0$ denotes the set of closed elements of $\mathbf{X}$, defined $\mathbf{X}_0 \;\widehat{=}\; \{\mathbf{x} : \mathbf{X} \mid \mathbf{x} = \pi.\varepsilon.\mathbf{x}\}$. Analogously $\mathbf{y} : \mathbf{Y}$ is *open* means it is a fixed point of $\varepsilon \circ \pi$, and $\mathbf{Y}_0 \;\widehat{=}\; \{\mathbf{y} : \mathbf{Y} \mid \varepsilon.\pi.\mathbf{y} = \mathbf{y}\}$.

Theorem(fixed-point isomorphism). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then

$$\left(\begin{array}{l} \mathbf{X}_0 = \mathbf{ran}.\pi \\ \mathbf{Y}_0 = \mathbf{ran}.\varepsilon \end{array} \right) ,$$

so that $\mathbf{X}_0$ and $\mathbf{Y}_0$ are isomorphic.

Adjunction

Next we show that each function in a Galois connection determines the other, and that it does so via an explicit formula (provided the underlying spaces are complete lattices) which we call adjunction. That characterises, for such underlying lattices, when a single function belongs to a Galois connection, and the form of its adjoint.

Theorem(determinism). If $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then each function determines the other.

The next result motivates adjunction as well as establishing further important properties of a Galois connection.

Theorem(junctivity). If $(\mathbf{X}, \leq)$ and $(\mathbf{Y}, \leq)$ are complete lattices and $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ then ε is universally $\sqcup$-junctive and π is universally $\sqcap$-junctive. In particular $\varepsilon.\bot = \bot$ and $\pi.\top = \top$.

There is an important converse to the previous result: if the underlying partial order $\mathbf{X}$ is a complete lattice and ε is universally $\sqcup$-junctive then a simple formula defines a projection which together with ε forms a Galois connection.

Theorem(adjunction). If $(\mathbf{X}, \leq)$ is a complete lattice and ε is universally $\sqcup$-junctive then the function $\pi : \mathbf{Y} \to \mathbf{X}$ given by

$$\pi.\mathbf{y} = \sqcup\{\mathbf{x} : \mathbf{X} \mid \varepsilon.\mathbf{x} \leq \mathbf{y}\}$$

is well defined and $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$.

Dually, for universally $\sqcap$-junctive π, the function $\varepsilon : \mathbf{X} \to \mathbf{Y}$ defined

$$\varepsilon.\mathbf{x} = \sqcap\{\mathbf{y} : \mathbf{Y} \mid \mathbf{x} \leq \pi.\mathbf{y}\}\,.$$

Galois embeddings

So far we have accounted, in a Galois connection, for all of the properties encountered in the movitaving examples, except for one property of floor: $\iota_{\mathbf{X}} = \pi \circ \varepsilon$. Galois connections which satisfy that stronger property occur when one semantic domain is stronger than the other, which accounts for the name we give them.

Definition

A *Galois embedding*, $\mathbf{ge}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$, is a Galois connection $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ in which $\pi \circ \varepsilon = \iota_{\mathbf{X}}$.

The following result is readily proved using the same techniques (mainly sandwiching) as in previous results. It means that in a Galois embedding $\mathbf{X}$ can be considered a subspace of $\mathbf{Y}$.

Theorem(equivalence for embeddings). These are equivalent

$\mathbf{ge}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$ (i.e. $\pi \circ \varepsilon = \iota_{\mathbf{X}}$)
ε is injective
π is surjective.

Defining a Galois connection

Suppose we wish to define a Galois connection with $\mathbf{X}$ and $\mathbf{Y}$ complete lattices, $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$, and we already have a definition of an embedding function $\varepsilon : \mathbf{X} \to \mathbf{Y}$. To complete the Galois connection there are three possibilities.

(a) Define $\pi : \mathbf{Y} \to \mathbf{X}$ and prove the Galois equivalence (or an equivalent form). This approach is recommended if an obvious simple π suggests itself. However it might be difficult to think of the appropriate π; use of adjunction to do that provides the second approach.

(b) Show ε is universally $\sqcup$-junctive and define π by adjunction. The advantage of this approach is that the Galois equivalence need not be checked and that no inspiration is required in defining π. Its disadvantage is primarily that universal $\sqcup$-junc- tivity of ε needs to be checked (otherwise the defining formula may not be well defined); also, to a lesser extent, that some manipulation may be necessary to simplify the form of π. The third approach obviates the former disadvantage.

(c) Substitute the definition of ε into the Galois equivalence and, by manipulation, obtain a definition of π. That approach shares with the previous one the advantage that the resulting π automatically completes a Galois connection. Its disadvantage is mainly that the manipulations leading to a definition may be difficult to achieve; and to a lesser extent that care must be exercised to avoid undefined cases (which might not be evident from the manipulation, comparable to use of approach (b) without confirming universal $\sqcup$-junctivity of ε).

In these notes we shall free to use whichever approach seems easiest.

Examples

Now that we have completed the basic theory of Galois connections, we consider three examples: first that of predicates, considered in section 2, then the motivating examples from this section. We use them to compare the three approaches above and to exemplify the isomorphism theorem.

Predicates

The most useful predicate combinator for constructing designs is conjunction: $\mathbf{p} \wedge \mathbf{r}$ is a design satisfying the constraints $\mathbf{p}$ and $\mathbf{r}$. If $\mathbf{q}$ is a specification then $\mathbf{p} \wedge \mathbf{r} \leq \mathbf{q}$ is the proof obligation for establishing that the design meets specification $\mathbf{q}$. Let us consider the adjoint of that obligation.

For any set $\mathbf{X}$ and predicate $\mathbf{r} : \mathbf{PX}$, define

$$\varepsilon : \mathbf{PX} \to \mathbf{PX}$$
$$\varepsilon.\mathbf{p} \;\widehat{=}\; \mathbf{p} \wedge \mathbf{r}\,.$$

We wish to make embedding ε into a Galois connection on $\mathbf{PX}$. Following (b) above, we observe that ε is universally $\sqcup$-junctive (why?) so its adjoint equals the function satisfying, for each $\mathbf{q} : \mathbf{PX}$,

$\pi.\mathbf{q}$
$\equiv$ adjunction
$\sqcup\{\mathbf{p} : \mathbf{PX} \mid \varepsilon.\mathbf{p} \leq \mathbf{q}\}$
$\equiv$ definition of ε and of order
$\sqcup\{\mathbf{p} : \mathbf{PX} \mid \mathbf{p} \wedge \mathbf{r} \Rightarrow \mathbf{q}\}$
$\equiv$ propositional calculus
$\sqcup\{\mathbf{p} : \mathbf{PX} \mid \mathbf{p} \Rightarrow (\mathbf{r} \Rightarrow \mathbf{q})\}$
$\equiv$ lub attained
$(\mathbf{r} \Rightarrow \mathbf{q})\,.$

Thus $\pi.\mathbf{q} \;\widehat{=}\; \mathbf{r} \Rightarrow \mathbf{q}$ completes a Galois connection with ε.

Were (c) above to be used instead, we would reason

$\varepsilon.\mathbf{p} \leq \mathbf{q}$
$\equiv$ definition of ε and of order
$(\mathbf{p} \wedge \mathbf{r}) \Rrightarrow \mathbf{q}$
$\equiv$
$\mathbf{p} \Rrightarrow (\mathbf{r} \Rightarrow \mathbf{q})$
$\equiv$
$\mathbf{p} \leq (\mathbf{r} \Rightarrow \mathbf{q})\,;$

which leads us to define $\pi.\mathbf{q} \;\widehat{=}\; \mathbf{r} \Rightarrow \mathbf{q}$.

Now to identify isomorphism, we must identify the ranges of the two functions. First

$\mathbf{ran}.\varepsilon$

$$
\begin{array}{ll}
= & \text{definition of } \mathbf{ran} \\
\{\varepsilon.\mathbf{p} \mid \mathbf{p} \in \mathbf{PX}\} & \\
= & \text{definition of } \varepsilon \\
\{\mathbf{p} \wedge \mathbf{r} \mid \mathbf{p} \in \mathbf{PX}\} & \\
= & \text{logic} \\
\{\mathbf{q} : \mathbf{PX} \mid \mathbf{q} \Rrightarrow \mathbf{r}\}. &
\end{array}
$$

Thus the range of the embedding ε consists of all predicates at least as strong as $\mathbf{r}$. Next

$$
\begin{array}{ll}
\mathbf{ran}.\pi & \\
= & \text{definition of } \mathbf{ran} \\
\{\pi.\mathbf{q} \mid \mathbf{q} \in \mathbf{PX}\} & \\
= & \text{definition of } \varepsilon \\
\{\mathbf{r} \Rightarrow \mathbf{q} \mid \mathbf{q} \in \mathbf{PX}\} & \\
= & \text{logic} \\
\{\mathbf{p} : \mathbf{PX} \mid \neg\mathbf{r} \Rrightarrow \mathbf{p}\}. &
\end{array}
$$

Thus the range of projection π consists of all predicates at least as weak as $\neg\mathbf{r}$.

Isomorphism of $\mathbf{ran}.\varepsilon$ with $\mathbf{ran}.\pi$ is thus the isomorphism of the predicates at least as strong as $\mathbf{r}$ with those as least as weak as $\neg\mathbf{r}$. It is achieved by negation, from the contrapositive law.

The floor

Equivalence (3) establishes that the embedding from the integers to the reals (both with their standard order), together with the floor function as projection in the reverse direction, forms a Galois connection.

The equivalence theorem shows that properties (4) follow straight from the definition of a Galois connection, except for $\forall\, \mathbf{n} : \mathbf{Z} \bullet \lfloor \varepsilon.\mathbf{n} \rfloor = \mathbf{n}$, which states that the functions form a Galois embedding.

In the orders on $\mathbf{Z}$ and $\mathbf{R}$, $\mathbf{m} \sqcup \mathbf{n}$ is the maximum of $\mathbf{m}$ and $\mathbf{n}$. Thus the embedding is disjunctive and so adjunction constructs the floor function (in the guise of equation (2)) straight from definition (3), a possibility which was probably not clear originally, but which corresponds to either (b) or (c) above.

The range of the embedding is the integral real numbers, and the range of the floor function is the integers. Thus the isomorphism theorem is no more than the order isomorphism of the integers with the integral reals. Of course every integer is a fixed point of the embedding followed by the floor, so $\mathbf{Z}_0 = \mathbf{Z}$. The fixed points of the floor followed by the embedding are the integral real numbers, so $\mathbf{R}_0$ consists of the integral reals.

Predicate transformers

Earlier in section 3, approach (a) was used to show that, for any $\mathbf{r} : \mathbf{X} \leftrightarrow \mathbf{Y}$, $\mathbf{gc}(\mathbf{sp.r}, \mathbf{wp.r}; \mathbf{PX}, \mathbf{PY})$. Recall that $\mathbf{sp.r}$ maps precondition $\mathbf{p}$ to its image through relation $\mathbf{r}$, whilst $\mathbf{wp.r}$ maps postcondition $\mathbf{q}$ to the weakest precondition whose image through $\mathbf{r}$ is at least as strong as $\mathbf{q}$.

How does use of approach (c) compare? It generates the definition of $\mathbf{wp.r}$ from that of $\mathbf{sp.r}$ as follows

$$
\begin{array}{ll}
\mathbf{sp.r.p} \leq \mathbf{q} & \\
\equiv & \text{pointwise order} \\
\forall\, \mathbf{y} : \mathbf{Y} \bullet \mathbf{sp.r.p.y} \leq \mathbf{q.y} & \\
\equiv & \text{definitions of } \mathbf{sp} \text{ and order} \\
\forall\, \mathbf{y} : \mathbf{Y} \bullet \exists\, \mathbf{x} : \mathbf{X} \bullet \begin{pmatrix} \mathbf{x\,r\,y} \\ \mathbf{p.x} \end{pmatrix} \Rightarrow \mathbf{q.y} & \\
\equiv & \text{logic} \\
\forall\, \mathbf{x} : \mathbf{X} \bullet \mathbf{p.x} \Rightarrow \forall\, \mathbf{y} : \mathbf{Y} \bullet (\mathbf{x\,r\,y} \Rightarrow \mathbf{q.y}) & \\
\equiv & \text{defining } \mathbf{wp.r.q.x} \mathrel{\widehat{=}} \forall\, \mathbf{y} : \mathbf{Y} \bullet (\mathbf{x\,r\,y} \Rightarrow \mathbf{q.y}) \\
\forall\, \mathbf{x} : \mathbf{X} \bullet \mathbf{p.x} \leq \mathbf{wp.r.q.x} & \\
\equiv & \text{pointwise order} \\
\mathbf{p} \leq \mathbf{wp.r.q}. &
\end{array}
$$

Approach (b) is almost identical, though universal $\sqcup$-junctivity of $\mathbf{sp.r}$ must be checked too.

To identify the claim of the isomorphism theorem, we observe that

$$
\begin{array}{ll}
\mathbf{q} \in \mathbf{ran}.(\mathbf{sp.r}) & \\
\equiv & \text{definition of } \mathbf{ran} \\
\exists\, \mathbf{p} : \mathbf{PX} \bullet \mathbf{q} = \mathbf{sp.r.p} & \\
\equiv & \text{pointwise equality} \\
\exists\, \mathbf{p} : \mathbf{PX} \bullet \forall\, \mathbf{y} : \mathbf{X} \bullet \mathbf{q.y} = \mathbf{sp.r.p.y} & \\
\equiv & \text{definition of } \mathbf{sp.r} \\
\exists\, \mathbf{p} : \mathbf{PX} \bullet \forall\, \mathbf{y} : \mathbf{X} \bullet \mathbf{q.y} = \exists\, \mathbf{x} : \mathbf{X} \bullet \begin{pmatrix} \mathbf{x\,r\,y} \\ \mathbf{p.x} \end{pmatrix}. &
\end{array}
$$

Thus $\mathbf{q}$ is the image through $\mathbf{r}$ of a predicate on $\mathbf{r}$'s domain.

Analogously

$$
\begin{array}{ll}
\mathbf{p} \in \mathbf{ran}.(\mathbf{wp.r}) & \\
\equiv & \text{pointwise equality} \\
\exists\, \mathbf{q} : \mathbf{PX} \bullet \forall\, \mathbf{x} : \mathbf{X} \bullet \mathbf{p.x} = \mathbf{wp.r.q.x} & \\
\equiv & \text{definition of } \mathbf{wp.r} \\
\exists\, \mathbf{q} : \mathbf{PX} \bullet \forall\, \mathbf{x} : \mathbf{X} \bullet \mathbf{p.x} = \forall\, \mathbf{y} : \mathbf{X} \bullet (\mathbf{x\,r\,y} \Rightarrow \mathbf{q.y}) & \\
\equiv & \text{definition of characteristic function} \\
\exists\, \mathbf{q} : \mathbf{PX} \bullet \mathbf{p} \Rrightarrow \xi_{\mathbf{dom.r}}. &
\end{array}
$$

Thus $\mathbf{p}$ is at least as strong as $\xi_{\mathbf{dom.r}}$.

To identify the isomorphism of $\mathbf{ran}.(\mathbf{sp.r})$ with $\mathbf{ran}.(\mathbf{wp.r})$ achieved by the restriction of $\mathbf{wp.r}$, we observe that for $\mathbf{wp.r}$ evaluated at an arbitrary member of $\mathbf{ran}.(\mathbf{sp.r})$,

$$
\begin{array}{ll}
\mathbf{wp.r}.(\mathbf{sp.r.p}).\mathbf{x} & \\
\equiv & \text{definition of } \mathbf{wp.r} \\
\forall\, \mathbf{y} : \mathbf{X} \bullet \mathbf{x\,r\,y} \Rightarrow \mathbf{sp.r.p} & \\
\equiv & \text{definition of } \mathbf{sp.r} \\
\forall\, \mathbf{y} : \mathbf{X} \bullet \mathbf{x\,r\,y} \Rightarrow \exists\, \mathbf{z} : \mathbf{X} \bullet \begin{pmatrix} \mathbf{z\,r\,y} \\ \mathbf{p.z} \end{pmatrix}. &
\end{array}
$$

Informally that affects the "$\mathbf{r}$-closure" of $\mathbf{p}$ by assigning to $\mathbf{p}$ the strongest predicate weaker than it and invariant under "having the same $\mathbf{r}$ image". In those terms $\mathbf{p}$ is a fixed point of $(\mathbf{wp.r}) \circ (\mathbf{sp.r})$ iff $\mathbf{p}$ equals its own $\mathbf{r}$-closure.

Similar elementary reasoning demonstrates that conjugation transforms the strongest postcondition of $\mathbf{r}$ to the weakest precondition of the converse of $\mathbf{r}$ (and dually) $(\mathbf{sp.r})^* = \mathbf{wp.}(\mathbf{r}^\sim)$.

4 Relational semantics

Introduction

We now begin the study of program semantics by considering a relational model of the guarded-command language, and its extension to a relational model of specifications. We shall find that in spite of its intuitive appeal, the model is neither as straightforward nor as useful for calculation as might have been hoped.

The guarded-command language

A computation in the guarded-command language consists of a sequence of assignments, configured by the control structures of sequential composition, conditional choice and repetition (although in the next section we also allow invocation of procedures). The vector of variables used by a computation is called its *state*, whose type consists of the cartesian product, called the *state space*, of the type of each variable. Each *terminating* assignment transforms state, from the *state before* the assignment to a *state after* it.

Without loss of generality we shall assume a homogeneous state space. For firstly we can ignore input and output by incorporating the former in the state before and the latter in the state after, by suitable extensions of those states. And secondly if those extended kinds of state have different types we can form their (discriminated) union. The result is a homogeneous state space which we denote $\mathbf{X}$.

As the computation proceeds a succession of states is traced out in state space $\mathbf{X}$. This is the view of a process or computing agent as a *dynamical system*, more commonly referred to in Computation as that of a *transition system*.

We are interested in semantic support for the development of implementations from their specifications, and so in the description of computations at all levels of abstraction: from abstract, simple choice of $\mathbf{X}$ through successively more complicated choices, to the final concrete choice in the implementation. Typically, abstract behaviour is described in a state-based specification language, intermediate-level behaviour is expressed in a refinement calculus, and implementations are written in code.

But whatever the level of abstraction, we must decide which behaviours we deem equivalent. Here we take the view of *total correctness*, the simplest outlook for ensuring correct, termination of designs. (Alternatives include *partial correctness*, which fails to distinguish nontermination from termination and so is too weak for our purposes, and *generalised correctness*, which incorporates both total and partial correctness and so is unnecessarily extensive for us here.) However we shall settle on that view only after a brief excursion into partial correctness. Let us begin by considering the behaviour of implementations.

E. W. Dijkstra's *guarded-command language* provides a convenient notation for describing algorithms imperatively. We shall refer to it as *code*, even though it deliberately abstracts many features of imperative programming languages and incorporates nondeterminism. However it is close enough to several popular programming languages so that algorithms designed in it can be "downcoded" routinely without introducing design errors. Its atomic statements are

- **skip**, which terminates immediately, changing no variables
- **abort**, which models the (undesirable) behaviour of nontermination
- assignment, $\mathbf{x} := \mathbf{e}$ (where $\mathbf{e}$ is an expression whose evaluation terminates with a single value) which terminates with $\mathbf{x}$ having the value of $\mathbf{e}$.

The constructors of the guarded-command language are

- sequential composition, $\mathbf{P} \fatsemi \mathbf{Q}$, which first executes statement $\mathbf{P}$ and, if $\mathbf{P}$ terminates, then executes statement $\mathbf{Q}$
- conditional, $\mathbf{if}\ \Box_i\ \mathbf{gi} \rightarrow \mathbf{Pi}\ \mathbf{fi}$, which executes one of the statements $\mathbf{Pi}$ whose *guard* $\mathbf{gi}$, a predicate on state space, is true; if none of the guards is true it aborts; note that evaluation of a predicate is presumed to terminate at each state
- repetition, $\mathbf{do}\ \Box_i\ \mathbf{gi} \rightarrow \mathbf{Pi}\ \mathbf{od}$, which—as long as at least one of the guards holds—executes one of the statements $\mathbf{Pi}$ whose guard $\mathbf{gi}$ is true; otherwise it terminates.

In order to give a semantics to the guarded command language it is convenient to consider instead the language $\mathcal{L}$ in which conditional and repetition are replaced by

- binary conditional, $\mathbf{P} \lhd \mathbf{b} \rhd \mathbf{Q}$, read $\mathbf{P}$ if $\mathbf{b}$ else $\mathbf{Q}$, which executes $\mathbf{P}$ if predicate $\mathbf{b}$ holds and executes $\mathbf{Q}$ otherwise
- choice, $\mathbf{P} \Box \mathbf{Q}$, which executes $\mathbf{P}$ or $\mathbf{Q}$; the choice between them is nondeterministic
- recursion, $\mu\mathbf{F}$, which for monotone (indeed, in practice, continuous) function $\mathbf{F}$ on programs denotes the least fixed point of $\mathbf{F}$.

In executable programming languages the expressions which appear in assignments are defined by syntax over variables and 'built-in' constants, functions and operations. It is assumed that their evaluation terminates with a single result. Accordingly, in the guarded-command language, an expression is a (total) function on state space $\mathbf{X}$. Computability (in the sense of Recursion Theory) is ignored. In an executable implementation computability is not an issue. In an implementation expressed in the guarded-command language it is assured by the particular form chosen for expressions.

Such freedom from a specific syntax for expressions allows the guarded-command language to be used at various levels of abstraction. A program can be specified as an 'abstract' assignment over 'not necessarily executable' state space (for example doubling a natural number: $\mathbf{x} := 2\mathbf{x}$), but implemented using only executable expressions (for example $\mathbf{x}$ may be represented as an array of bits and the assignment achieved by a loop moving bits one place to the left and appending 0 in the least-significant place). Indeed the first program developments where performed in the guarded-command language.

The guarded-command language thus has deterministic assignments, and introduces nondeterminism through overlappping guards. In language $\mathcal{L}$ an alternative approach is taken: an expression is permitted not to terminate and to be nondeterministic. As a result $\mathcal{L}$ is even more useful in supporting program developments (though we shall extend it further).

Languages $\mathcal{L}$ and the guarded command language are endowed with the pre-order $\sqsubseteq$ of program refinement, defined by the law $\mathbf{P} \sqsubseteq \mathbf{Q} \Leftrightarrow \mathbf{P} \sqcap \mathbf{Q} = \mathbf{P}$. Though $\sqsubseteq$ is not antisymmetric we shall consider as equivalent computations which are "equally deterministic". Firstly that decision implies in particular that the pre-order is made into a partial order on equivalence classes of programs, just as we did for the partially-ordered space of predicates under $\Rightarrow$ modulo $\equiv$. But it also means that syntactically distinct programs dee -med to have the same behaviour will be considered equivalent. For example the three programs **abort**, **if** *false* $\rightarrow$ **skip fi**, and **do** *true* $\rightarrow$ **skip od** are deemed equivalent; evidently each aborts. Such equivalences reflect our commitment to total correctness. The implications of that will be discussed later; in summary, possible and certain nontermination are identified.

It is to language $\mathcal{L}$ with that partial order, $(\mathcal{L}, \sqsubseteq)$, that we shall give a relational semantics in this chapter and a transformer semantics later. In each semantic model considered we shall write $[\mathbf{P}]$ for the denotation of construct $\mathbf{P}$ in $\mathcal{L}$.

Partial functions

We start our semantic quest (ignoring, for the moment, total correctness and) by recalling the form of semantics used in Recursion Theory. It provides the traditional method for modelling deterministic computations which do not necessarily terminate. Thus it is too limited for our (nondeterministic) needs. However it provides a simple starting point by separating, as far as possible, the two issues of termination and nondeterminism. Our work will not be wasted since we shall find that its useful features can be lifted, by a Galois connection, to a more comprehensive model.

The model

Each program is assumed to be deterministic and so is represented by a partial function on homogeneous state space $\mathbf{X}$. In terms of the semantic denotation $[\mathbf{P}]$ of program $\mathbf{P}$, $[\mathbf{P}] : \mathbf{X} \nrightarrow \mathbf{X}$ with the domain of $[\mathbf{P}]$ equal to the set of states from which program $\mathbf{P}$ terminates. Semantic denotations thus lie in $\mathcal{P} \;\widehat{=}\; \mathbf{X} \nrightarrow \mathbf{X}$.

Program $\mathbf{P}$ is refined by program $\mathbf{Q}$ iff partial function $[\mathbf{Q}]$ is an extension of partial function $[\mathbf{P}]$. For then $\mathbf{Q}$ gives the same results as $\mathbf{P}$ when the latter terminates, though $\mathbf{Q}$ may terminate from states in which $\mathbf{P}$ does not. We thus define the order on $\mathcal{P}$ to be inclusion: $\mathbf{f} \sqsubseteq \mathbf{g} \;\widehat{=}\; \mathbf{f} \subseteq \mathbf{g}$.

In $(\mathcal{P}, \sqsubseteq)$ the least, or most unrefined, element is the empty partial function, $\{\}$, which denotes the nonterminating program **abort**. An element is maximal iff it can be extended no further: it is a function. And an element is compact iff it is a finite extension of the empty partial function: it is finite. Thus:

> **Theorem($\mathcal{P}$ a domain).** $(\mathcal{P}, \sqsubseteq)$ is a domain with least element $\{\}$, whose maximal elements are the total functions and whose compact elements are the finite partial functions: $\mathbf{K}(\mathcal{P}) = \{\mathbf{f} : \mathcal{P} \mid \#\mathbf{f} < \infty\}$.

Semantics

The approach of Recursion Theory provides, for the deterministic subset of language $\mathcal{L}$, the following semantics. Each denotation is motivated by the operational explanation of its corresponding construct given model $\mathcal{P}$.

$$
\begin{array}{rcl}
[\mathbf{skip}] & \widehat{=} & \iota_{\mathbf{X}} \\
[\mathbf{abort}] & \widehat{=} & \{\} \\
[\mathbf{x} := \mathbf{e}] & \widehat{=} & \{(\mathbf{x}, \mathbf{e}.\mathbf{x}) \mid \mathbf{x} \in \mathbf{X}\} \\
 & & \mathbf{e} \text{ total deterministic} \\
[\mathbf{P} \fatsemi \mathbf{Q}] & \widehat{=} & [\mathbf{P}] \fatsemi [\mathbf{Q}] \\
[\mathbf{P} \lhd \mathbf{b} \rhd \mathbf{Q}] & \widehat{=} & \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}[\mathbf{P}]\mathbf{y} \lhd \mathbf{b}.\mathbf{x} \rhd \mathbf{x}[\mathbf{Q}]\mathbf{y}\} \\
[\mathbf{P} \Box \mathbf{Q}] & \widehat{=} & [\mathbf{P}] \cup [\mathbf{Q}] \\
 & & [\mathbf{P}] \upharpoonright \mathbf{dom}.\mathbf{Q} = [\mathbf{Q}] \upharpoonright \mathbf{dom}.\mathbf{P} \\
[\mu \mathbf{F}] & \widehat{=} & \bigcup_{\kappa} \mathbf{F}^{\kappa}.\{\} \\
 & & \mathbf{F} \text{ monotone on } \mathcal{P} .
\end{array}
$$

The last union is over all ordinals κ . Fortunately if $\mathbf{F}$ is continuous then that reduces to a countable union (attained from a countable iteration of $\mathbf{F}$ applied to the least element, **abort**). The next result ensures that is the case whenever $\mathbf{F}$ is a composition of the three program combinators: sequential composition, binary conditional and nondeterministic choice.

> **Theorem(semantics of $\mathcal{P}$).** All denotations are well defined and the three combinators are continuous (hence monotone).

The restriction to determinism, imposed simply because we have chosen to start from the traditional approach of Recursion Theory, will be removed in the next section since it conflicts with one of the strengths of the guarded-

command language: its treatment of nondeterminism.

Nonetheless the semanatic denotations of programs in $\mathcal{L}$ (with assignments having terminating deterministic expressions) exhaust model $\mathcal{P}$; that is, $[\mathcal{L}] = \mathcal{P}$. For if $\mathbf{f} \in \mathcal{P}$ then choosing

$$\mathbf{P} \;\widehat{=}\; (\mathbf{x} := \mathbf{f.x}) \lhd \mathbf{x} \in \mathbf{dom.f} \rhd \mathbf{abort},$$

yields $\mathbf{P} \in \mathcal{L}$ and $[\mathbf{P}] = \mathbf{f}$.

Laws

The utility of that (or any) semantics must be confirmed by proving the soundness of the *laws of programming*, which reflect our view of program behaviour. For example the law $\mathbf{abort} ; \mathbf{P} = \mathbf{abort}$ expresses the inability of any program to follow nontermination; it holds because the composition of the empty partial function with any function is empty. The law $\mathbf{skip} ; \mathbf{P} = \mathbf{P}$ expresses the fact that **skip** terminates without effect, and thus that any program can follow termination; it holds because function $\iota_{\mathbf{X}}$ is the identity for composition. The law $\mathbf{abort} \Box \mathbf{P} = \mathbf{abort}$ expresses the fact that a program which might not terminate is as unreliable as one which is certain not to terminate; thus the behaviour of the two programs should be identified to avoid errors of nontermination. However it *fails* in this model because the union of the empty partial function with any function yields that function. Thus in the model $\mathcal{P}$, $\mathbf{abort} \Box \mathbf{P} = \mathbf{P}$. A result of that fallacious law would therefore be that program development based on equivalence between programs (or, even worse, merely on refinement) would ignore termination. In terms introduced earlier, $(\mathcal{P}, \sqsubseteq)$ is appropriate to partial correcntess and not total correctness. We return to this when studying predicate transformers.

There is a second problem. Were inclusion to be used for program development, a (partial) specification could have inconsistent refinements!

Thus even for deterministic programs, model $\mathcal{P}$ is unacceptable for reasoning about programs. It also does not contain development features like miracles. There is therefore little point in attempting a simple modification to incorporate nondeterminism. We must find an alternative model, which we begin to do by distinguishing termination in the usual way. That will leave us free to model nondeterminism as many-valuedness, and to exploit the relational calculus.

Total relations

In this section we provide a model for $\mathcal{L}$ which avoids the weaknesses of model $\mathcal{P}$.

Suppose that $\perp$ denotes an element not in (homogeneous) state space $\mathbf{X}$ and let $\mathbf{X}_\perp$ denote $\mathbf{X}$ suitably augmented $\mathbf{X}_\perp \;\widehat{=}\; \mathbf{X} \cup \{\perp\}$.

In the new model program $\mathbf{P}$ is represented by a relation $[\mathbf{P}]$ on $\mathbf{X}_\perp$. The condition "program $\mathbf{P}$ may not terminate from initial state $\mathbf{x}$" is now expressed $\mathbf{x}\,[\mathbf{P}]\,\perp$. The phrase "may not" is required since a nondeterministic $\mathbf{P}$ may both terminate and not terminate from a given initial state.

Introducing the model

The idea of the new model is that the refinement ordering on programs be represented by containment of the relations that represent them. Thus, for example, the least element in the model, $[\mathbf{abort}]$, must be the universal relation ω. However that means that for a program to be able to refine program $\mathbf{P}$ that "terminates less", $\mathbf{P}$ must behave like ω at those states from which it may not terminate; for only then is refinement represented as containment.

Thus if the ordering on our new model is to be containment of relations, the relations must satisfy a healthiness condition: if relation $\mathbf{d}$ represents a computation which may not terminate from state $\mathbf{x}$, then for each $\mathbf{y} : \mathbf{X}_\perp$, $\mathbf{x}\,\mathbf{d}\,\mathbf{y}$:

$$\mathbf{x}\,\mathbf{d}\,\perp \Rightarrow \forall \mathbf{y} : \mathbf{X}_\perp \bullet \mathbf{x}\,\mathbf{d}\,\mathbf{y}. \qquad (5)$$

In other words, for each $\mathbf{x} : \mathbf{X}_\perp$ the set $\mathbf{d}[\![\mathbf{x}]\!] \;\widehat{=}\; \{\mathbf{y} : \mathbf{X}_\perp \mid \mathbf{x}\,\mathbf{d}\,\mathbf{y}\}$ is up-closed in the flat order on $\mathbf{X}_\perp$ (in which the elements of $\mathbf{X}$ are incomparable but each dominates $\perp$).

Now recursion is represented as the intersection of a *decreasing* family of relations, so some condition is required to ensure that it is nonempty. At this point it is convenient to consider the degree to which the guarded-command language, or indeed language $\mathcal{L}$, exhibits nondeterminism. In both it is explicitly assumed, for semantic purposes, that only bounded nondeterminism occurs. Thus the expression in an assignment assigns to each state at most a finite number of alternative values (if it terminates). In other words, the relational denotation of each program is finitary. The semantic reason behind that choice is precisely that it ensures that a recursion has a nonempty solution —and that it is again finitary. So finiteness is our final healthiness condition.

Those healthiness conditions ensure that the refinement ordering is as simple as possible. It is possible to simplify the healthiness conditions at the expense of complicating the ordering.

Those considerations are conveniently described using further relational notation. For $\mathbf{r} : \mathbf{X}_\perp \leftrightarrow \mathbf{X}_\perp$, $\mathbf{r}$ is *finitary* means that for each $\mathbf{x} : \mathbf{X}_\perp$, the set $\mathbf{r}[\![\mathbf{x}]\!]$ is either nonempty and finite, or equal to $\mathbf{X}_\perp$. Thus in particular a finitary relation is total.

Defining the model

$\mathcal{D}$ denotes the space of relations on $\mathbf{X}_\perp$ which are finitary and relate $\perp$ to every element of $\mathbf{X}_\perp$

$$\mathcal{D} \;\widehat{=}\; \{\mathbf{d} : \mathbf{X}_\perp \leftrightarrow \mathbf{X}_\perp \mid \begin{pmatrix} \mathbf{d} \text{ is finitary} \\ \mathbf{d}[\![\perp]\!] = \mathbf{X}_\perp \end{pmatrix}\}.$$

The space $(\mathcal{D}, \supseteq)$ is partially ordered; note that the ordering is containment rather than inclusion. Its least, or most unrefined, element is the universal relation ω on $\mathbf{X}_\perp$; its maximal elements are those which can be refined no fur-

ther: they are functions. Moreover:

Theorem($\mathcal{D}$ a domain). $(\mathcal{D}, \supseteq)$ is a domain with least element ω, whose maximal elements are (total) functions and whose compact elements are those relations relating all but finitely many members of $\mathbf{X}_\perp$ to each member of $\mathbf{X}_\perp$.

Galois connection

We wish to have a Galois connection from $(\mathcal{P}, \sqsubseteq)$ to $(\mathcal{D}, \sqsubseteq)$ which converts abortion in $\mathcal{P}$ (where it is represented as partiality) to abortion in $\mathcal{D}$ (where it is represented as arbitrary behaviour). Thus we must define $\varepsilon : \mathcal{P} \to \mathcal{D}$ by (recalling that $\overline{\mathbf{r}}$ denotes the complement of relation $\mathbf{r}$), $\varepsilon.\mathbf{f} \;\widehat{=}\; \mathbf{f} \cup \mathbf{f} \fatsemi \overline{\omega}$. On the domain of $\mathbf{f}$, $\varepsilon.\mathbf{f}$ equals $\mathbf{f}$; but outside it $\varepsilon.\mathbf{f}$ behaves like the universal relation. That captures the intuition behind our model, expressed in condition (5). We refer informally to $\varepsilon.\mathbf{f}$ as the *fluffing up* of $\mathbf{f}$!

Approach (c) yields the following definition of $\pi : \mathcal{D} \to \mathcal{P}$ forming a Galois connection:

$$\pi.\mathbf{d} \;\widehat{=}\; \{(\mathbf{x}, \mathbf{y}) : \mathbf{d} \mid \left(\begin{array}{l} \mathbf{y} \neq \perp \\ \mathbf{x}\,\mathbf{d}\,\mathbf{y}' \Rightarrow \mathbf{y}' = \mathbf{y} \end{array} \right)\},$$

$\pi.\mathbf{d}$ denotes the largest partial function in $\mathbf{d}$ which "accounts for all of $\mathbf{d}$'s results at its arguments". It may be thought of as the largest partial function which approximates, in $\mathcal{P}$, total relation $\mathbf{d}$; indeed that is the form for π given by adjunction.

Theorem(connecting $\mathcal{P}$ to $\mathcal{D}$). Functions ε and π form a Galois connection from $(\mathcal{P}, \subseteq)$ to $(\mathcal{D}, \supseteq)$: $\mathbf{gc}(\varepsilon, \pi; \mathcal{P}, \mathcal{D})$.

Semantics

The Galois connection can now be used to make easier the task of describing the semantics of $\mathcal{L}$ in $\mathcal{D}$. Indeed it enables us simply to lift, via ε, the semantics from $\mathcal{P}$.

Most of the denotations follow from those for $\mathcal{P}$. For the semantics of assignment we could either assume expression $\mathbf{e}$ is deterministic (as in the $\mathcal{P}$ case), using nondeterministic choice to generate nondeterministic assignments; or we could permit $\mathbf{e}$ to be nondeterministic from the start. We adopt the latter approach, which requires $\mathbf{e}$ to be finitary as a relation in order for $\varepsilon.\mathbf{e}$ to be well defined.

$$\begin{array}{rcl}
[\mathbf{skip}] & \widehat{=} & \varepsilon.\iota_{\mathbf{X}} \quad (= \iota_{\mathbf{X}} \cup \omega \restriction \perp) \\
[\mathbf{abort}] & \widehat{=} & \varepsilon.\{\} \quad (= \omega) \\
[\mathbf{x} := \mathbf{e}] & \widehat{=} & \varepsilon.\{(\mathbf{x}, \mathbf{e}.\mathbf{x}) \mid \mathbf{e}.\mathbf{x} \text{ terminates}\} \\
 & & \mathbf{e} \text{ finitary} \\
[\mathbf{P} \fatsemi \mathbf{Q}] & \widehat{=} & [\mathbf{P}] \fatsemi [\mathbf{Q}] \\
[\mathbf{P} \lhd \mathbf{b} \rhd \mathbf{Q}] & \widehat{=} & \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}[\mathbf{P}]\mathbf{y} \lhd \mathbf{b}.\mathbf{x} \rhd \\
 & & \quad \mathbf{x}[\mathbf{Q}]\mathbf{y}\} \\
[\mathbf{P} \Box \mathbf{Q}] & \widehat{=} & [\mathbf{P}] \cup [\mathbf{Q}] \\
[\mu\,\mathbf{F}] & \widehat{=} & \bigcap_\kappa \mathbf{F}^\kappa.\omega \\
 & & \mathbf{F} \text{ monotone on } \mathcal{D} .
\end{array}$$

We wish to confirm, as we did for $\mathcal{P}$, that each denotation is well defined and that recursion can be given a countable unfolding because each combinator is continuous.

Theorem(semantics of $\mathcal{D}$). Each denotation in $[\mathcal{L}]$ is well defined, and the three combinators are continuous.

By virtue of the more general nature of its expressions, language $\mathcal{L}$ is more expressive than the guarded-command language. We have seen that, with total functions for expressions, $[\mathcal{L}] = \mathcal{P}$. Evidently with arbitrary finitely-nondeterministic expressions, $[\mathcal{L}] = \mathcal{D}$.

Specification space

We now consider how the relational model can be used to support program development. Model $\mathcal{D}$ captures exactly the semantics of code, as we have just seen. So first we must extend it to incorporate development constructs.

The "specification space" we consider consists of partial relations. Miracles are represented by non-total relations; and examples of combinators not in $\mathcal{D}$ include complement and intersection (with some given relation). Though not code, they are extremely useful specification constructs.

Definition

The *specification space of relations* means the space $(\mathcal{R}, \supseteq)$ where $\mathcal{R} \;\widehat{=}\; \mathbf{X}_\perp \leftrightarrow \mathbf{X}_\perp$.

Since the specification space $(\mathcal{R}, \supseteq)$ actually *contains* the space $(\mathcal{D}, \supseteq)$ denoting "code", no Galois connection is required (in fact none exists).

Development

A development consists of a sequence of designs in $\mathcal{R}$, starting from the specification, each design refining its predecessor, and ending with the desired implementation (in $\mathcal{D}$). Laws are thus required, amongst others, for introducing the various constructs of language $\mathcal{L}$. Perhaps the most important are laws for introducing sequential composition. We consider those now.

In developing sequential programs we must be able to solve for $\mathbf{S}$ in $\mathbf{Q} \sqsubseteq \mathbf{S} \fatsemi \mathbf{P}$. For only then can we implement

specification $\mathbf{Q}$ by calculation, given the final component $\mathbf{P}$ in a sequential composition. Similarly we must be able to solve for $\mathbf{S}$ in $\mathbf{Q} \sqsubseteq \mathbf{P} ; \mathbf{S}$, since that corresponds to being given the first component in the sequential composition.

Weakest pre- and postspecification

The weakest such $\mathbf{S}$ for which $\mathbf{Q} \sqsubseteq \mathbf{S} ; \mathbf{P}$, is called the *weakest prespecification of* $\mathbf{P}$ *in* $\mathbf{Q}$, and is written $\mathbf{Q}/\mathbf{P}$. Thus $\mathbf{Q}/\mathbf{P} \sqsubseteq \mathbf{S} \;\widehat{=}\; \mathbf{Q} \sqsubseteq \mathbf{S} ; \mathbf{P}$. In that form, as a Galois connection, we see that $\mathbf{Q}/\mathbf{P}$ exists by disjunctivity of $;$.

Similarly the weakest $\mathbf{S}$ for which $\mathbf{Q} \sqsubseteq \mathbf{P} ; \mathbf{S}$, is called the *weakest postspecification of* $\mathbf{P}$ *in* $\mathbf{Q}$, defined $\mathbf{P} \setminus \mathbf{Q} \sqsubseteq \mathbf{S} \;\widehat{=}\; \mathbf{Q} \sqsubseteq \mathbf{P} ; \mathbf{S}$.

We shall need both forms of weakest specification in these notes. Here are the properties of weakest prespecification; exploration of analogues for weakest postspecification is left as an exercise. Proofs are mostly routine application of the Galois equivalence.

Theorem(basic properties).

1. $(\mathbf{Q}/\mathbf{P}) ; \mathbf{P} \subseteq \mathbf{Q}$
2. $\mathbf{Q} \subseteq (\mathbf{Q} ; \mathbf{P})/\mathbf{P}$
3. $\iota \subseteq \mathbf{Q}/\mathbf{Q}$
4. $\mathbf{Q} = \mathbf{Q}/\iota$
5. $\mathbf{Q}/(\mathbf{P} ; \mathbf{R}) = (\mathbf{Q}/\mathbf{R})/\mathbf{P}$
6. $(\mathbf{Q} \cap \mathbf{P})/\mathbf{R} = (\mathbf{Q}/\mathbf{R}) \cap (\mathbf{P}/\mathbf{R})$
7. $\mathbf{Q}/(\mathbf{P} \cup \mathbf{R}) = (\mathbf{Q}/\mathbf{P}) \cap (\mathbf{Q}/\mathbf{R})$.

Theorem(further properties).

1. $\mathbf{Q}/\mathbf{P} = \bigcup\{\mathbf{R} : \mathcal{R} \mid \mathbf{R} ; \mathbf{P} \subseteq \mathbf{Q}\}$
2. $\mathbf{x}_0(\mathbf{Q}/\mathbf{P})\mathbf{x} \equiv \forall \mathbf{y} : \mathbf{X} \bullet (\mathbf{x}\mathbf{P}\mathbf{y} \Rightarrow \mathbf{x}_0\mathbf{Q}\mathbf{y})$
3. $\mathbf{Q}/\mathbf{P} = \overline{\overline{\mathbf{Q}} ; \mathbf{P}^{\sim}}$
4. For each total $\mathbf{F} : \mathcal{P}$, $\mathbf{Q}/\mathbf{F} = \mathbf{Q} ; (\iota/\mathbf{F}) = \mathbf{Q} ; \mathbf{F}^{\sim}$
5. For each $\mathbf{f} : \mathcal{P}$, $\mathbf{Q}/\varepsilon.\mathbf{f} = \mathbf{Q}/\mathbf{f} \cap (\overline{\{\}/\mathbf{f}} \cup \mathbf{Q}/\omega)$
6. $\mathbf{Q}/(\mu\mathbf{F}) = \bigcup\{\mathbf{Q}/(\mathbf{F}^{\mathbf{n}}.\omega) \mid \mathbf{n} : \mathbf{N}\}$.

The results for weakest pre and post -specification may be used for program development. But it is to be observed that they are not particularly simple to apply.

5 Relational data refinement

Introduction

So far we have considered only computations (programs or specifications) over the same state space. However a vital technique in program development permits a computation to be specified (clearly but at the expense of being unimplementable) using abstract state variables but implemented (efficiently but at the expense of being complicated) using so called *concrete* state variables.

Such use of local variables is controlled through use of *datatypes*, and the technique for developing correct concrete computations from their more abstract specifications is called *data refinement*. In this chapter we introduce those concepts and analyse them in the relational model. We consider two simulation rules which are sound methods of data refinement. By itself neither rule is compete, yet together they are complete. This relects essential properties of the relational model, and is to be compared with the completeness of just one of those rules in the predicate-transformer model (established in section 7).

Datatypes

A *datatype*, or module, controls access to its local variables through invocation of its named operations and by communication using their input and output parameters.

Example

The type *Mean* contains a bag of real numbers, which is initially empty. Numbers can be added to the bag, by invoking operation **add** with input parameter the number to be added; and the mean of a nonempty bag can be obtained by invoking operation **mean** whose output parameter gives that mean. We write bag comprehension like set comprehension but with $[\![\ldots]\!]$ replacing $\{ \ldots \}$.

Type *Mean*

state	$\mathbf{b} : \mathbf{bag\ R}$
initially	$\mathbf{b} := [\![ ]\!]$
add$(\mathbf{x}? : \mathbf{R})$	$\mathbf{b} := \mathbf{b} + [\![\mathbf{x}]\!]$
mean$(\mathbf{y}! : \mathbf{R})$	$(\mathbf{y} := (\Sigma\, \mathbf{b})/\#\mathbf{b}$
	$\lhd\, \mathbf{b} \neq [\![ ]\!] \,\rhd$
	abort$)$

end

Definition

A *datatype* for use by programs having global state space $\mathbf{G}$, consists of a state space $\mathbf{L}$ (for its local variables), an initialisation operation *in* from $\mathbf{G}$ to $\mathbf{L}$, a finalisation operation *fin* from $\mathbf{L}$ to $\mathbf{G}$ (which is **skip** for type *Mean*, and is hence ommitted), and a family $\mathcal{O}$ of (named) operations on $\mathbf{L}$ having input and output. It is represented $\mathbf{T} = (\mathbf{in}, \mathcal{O}, \mathbf{fin})$, with $\mathbf{L}$ left implicit to emphasise that it cannot be directly accessed by the programs on $\mathbf{G}$ which use $\mathbf{T}$.

Use of datatype $\mathbf{T}$ by a program $\mathbf{P}$ on $\mathbf{G}$ consists of a sequence $\mathbf{in} ; \mathbf{P} ; \mathbf{fin}$ in which $\mathbf{P}$ invokes the operations in $\mathcal{O}$.

Refinement

The two operations of *Mean* can be executed efficiently, without altering observable behaviour, by replacing the local variable, bag $\mathbf{b}$, by its sum and cardinality. Although that loses information (like which numbers have been added) that information is not discernible from the operations of the type.

Type *Mean'*

state	$\mathbf{s} : \mathbf{R};\ \mathbf{n} : \mathbf{N}$
initially	$\mathbf{s}, \mathbf{n} := 0, 0$
add'$(\mathbf{x}? : \mathbf{R})$	$\mathbf{s}, \mathbf{n} := \mathbf{s} + \mathbf{x}, \mathbf{n} + 1$
mean'$(\mathbf{y}! : \mathbf{R})$	$(\mathbf{y} := \mathbf{s}/\mathbf{n}$
	$\lhd\, \mathbf{n} \neq 0 \,\rhd$
	abort$)$

end

Each use of *Mean'* by a program of language $\mathcal{L}$ (achieved by invoking operations of the type) could have been achieved by using type *Mean* instead. It is that which we wish to formalise in defining refinement between datatypes.

Conformality

Consider two datatypes which are *conformal* in the sense of having the same global state space and with their operations paired, so that for each $\mathbf{O} : \mathcal{O}$, operation $\mathbf{O}$ of one type has input and output parameters of the same type as operation $\mathbf{O}'$ of the other:

$$\begin{array}{lcl} \mathbf{T} & \widehat{=} & (\mathbf{in}, \mathcal{O}, \mathbf{fin}) \\ \mathbf{T}' & \widehat{=} & (\mathbf{in}', \mathcal{O}', \mathbf{fin}') \,. \end{array}$$

Extend language $\mathcal{L}$ to include named-procedure calls. Thus invocations of operations $\mathbf{O}$ and $\mathbf{O}'$ now become program statements. Suppose that $\mathbf{P} \in \mathcal{L}$ includes procedure calls from just type $\mathbf{T}$ and let $\mathbf{P}'$ denote the program which is like $\mathbf{P}$ except that it uses $\mathbf{O}'$ where $\mathbf{P}$ uses $\mathbf{O}$. $\mathbf{P}'$ thus includes procedure calls from just type $\mathbf{T}'$.

Definition

Type $\mathbf{T}'$ *refines* type $\mathbf{T}$, written $\mathbf{T} \sqsubseteq \mathbf{T}'$, means

$$\forall \mathbf{P} : \mathcal{L} \bullet \mathbf{in} \fatsemi \mathbf{P} \fatsemi \mathbf{fin} \sqsubseteq \mathbf{in}' \fatsemi \mathbf{P}' \fatsemi \mathbf{fin}' .$$

In terms of the relational model,

$$\forall \mathbf{P} : \mathcal{L} \bullet [\mathbf{in} \fatsemi \mathbf{P} \fatsemi \mathbf{fin}] \supseteq [\mathbf{in}' \fatsemi \mathbf{P}' \fatsemi \mathbf{fin}'] .$$

Observe that proof of refinement between data types thus requires proof of that program refinement for all $\mathbf{P} \in \mathcal{L}$. Our next task is to find a sufficient condition for that which is simple to establish but which has wide applicability.

Simulations

We require a condition sufficient for data refinement but which does not require reasoning over all $\mathbf{P} \in \mathcal{L}$. Such a condition can be found by exploiting the states of the datatypes.

Example

Consider the conformal datatypes *Mean* and *Mean'*. The intuition that the former is refined by the latter can be based on the observation that the state $\mathbf{b} : \mathbf{bag}\ \mathbf{R}$ of the former is represented by the state $\mathbf{s} : \mathbf{R};\ \mathbf{n} : \mathbf{N}$ of the latter, iff

$$\left(\begin{array}{l} \mathbf{s} = \sum \mathbf{b} \\ \mathbf{n} = \#\mathbf{b} \end{array} \right) .$$

Now the intuition is completed by using that correspondence to pair uses of the type.

Definition

Given conformal datatypes $\mathbf{T}$ and $\mathbf{T}'$ as above with local states $\mathbf{l} : \mathbf{L}$ and $\mathbf{l}' : \mathbf{L}'$ respectively, relation $\mathbf{d} : \mathbf{L} \leftrightarrow \mathbf{L}'$ is a *downwards simulation*, or forwards simulation, or *dsim* for short, from $\mathbf{T}$ to $\mathbf{T}'$ means

$$\begin{array}{ll} \mathbf{in} & \mathbf{in} \fatsemi \mathbf{d} \sqsubseteq \mathbf{in}' \\ \mathbf{op} & \forall \mathbf{O} : \mathcal{O} \bullet \mathbf{O} \fatsemi \mathbf{d} \sqsubseteq \mathbf{d} \fatsemi \mathbf{O}' \\ \mathbf{fin} & \mathbf{fin} \sqsubseteq \mathbf{d} \fatsemi \mathbf{fin}' . \end{array}$$

Definition

With $\mathbf{T}$ and $\mathbf{T}'$ as above, relation $\mathbf{u} : \mathbf{L}' \leftrightarrow \mathbf{L}$ is an *upwards simulation*, or backwards simulation, or *usim* for short, from $\mathbf{T}'$ to $\mathbf{T}$ means

$$\begin{array}{ll} & \mathbf{u} \text{ is finitary} \\ \mathbf{in} & \mathbf{in} \sqsubseteq \mathbf{in}' \fatsemi \mathbf{u} \\ \mathbf{op} & \forall \mathbf{O} : \mathcal{O} \bullet \mathbf{u} \fatsemi \mathbf{O} \sqsubseteq \mathbf{O}' \fatsemi \mathbf{u} \\ \mathbf{fin} & \mathbf{u} \fatsemi \mathbf{fin} \sqsubseteq \mathbf{fin}' . \end{array}$$

Of course it is vital that those rules, intended as sufficient conditions for data refinement, are correct in the following sense.

Theorem(Soundness of simulations). If there is a *dsim* from $\mathbf{T}$ to $\mathbf{T}'$ or a *usim* from $\mathbf{T}'$ to $\mathbf{T}$ then $\mathbf{T}$ is refined by $\mathbf{T}'$: $\mathbf{T} \sqsubseteq \mathbf{T}'$.

Utility

Simulations are used in documenting developments since, like loop invariants, they clarify otherwise complex code; in this case, they clarify the relationship that is supposed to exist between an abstract datatype and its implementation.

Perhaps more importantly simulations can be used in deriving a datatype from its more abstract specification. In terms of weakest postspecification (see section 4), the concrete operations $\mathbf{in}'$, $\mathbf{O}'$ and $\mathbf{fin}'$ can be *specified*:

$$\begin{array}{lcl} \mathbf{in}' & \widehat{=} & \mathbf{in} \fatsemi \mathbf{d} \\ \mathbf{O}' & \widehat{=} & \mathbf{d} \setminus (\mathbf{O} \fatsemi \mathbf{d}) \\ \mathbf{fin}' & \widehat{=} & \mathbf{d} \setminus \mathbf{fin} . \end{array}$$

Completeness of simulations

We now consider the breadth of applicability of the simulation rules for ensuring data refinement.

First we observe that by itself neither rule is complete for establishing refinement.

Incompleteness of dsim

The abstract datatype, $\mathbf{T}$, has five states and a single operation, $\mathbf{O}$, which after initialisation outputs 1 in one state and 0 in the other. It is defined using $\mathbf{x}_0$ for the initial state,

$\mathbf{x}$ for the final state and $\mathbf{y}$ for output:

$$\begin{array}{ll}\textbf{Type } T & \\ \quad \textbf{state} & \mathbf{x} : \{0,1,2,3,4\} \\ \quad \textbf{initially} & \mathbf{x} := 0 \\ \\ \quad \mathbf{O}(\mathbf{y}! : \{0,1\}) & \left(\begin{array}{l} \mathbf{x}_0 = 0 \wedge \mathbf{x} = 1 \wedge \mathbf{y} = 0 \\ \vee \\ \mathbf{x}_0 = 0 \wedge \mathbf{x} = 2 \wedge \mathbf{y} = 0 \\ \vee \\ \mathbf{x}_0 = 1 \wedge \mathbf{x} = 3 \wedge \mathbf{y} = 1 \\ \vee \\ \mathbf{x}_0 = 2 \wedge \mathbf{x} = 4 \wedge \mathbf{y} = 0 \end{array}\right) \\ \\ \quad \textbf{finally} & \textbf{skip} \\ \textbf{end}. & \end{array}$$

In the concrete type, $\mathbf{T}'$, there are four states and operation $\mathbf{O}'$ is nondeterministic.

$$\begin{array}{ll}\textbf{Type } T' & \\ \quad \textbf{state}' & \mathbf{x} : \{0,1,2,3\} \\ \quad \textbf{initially}' & \mathbf{x} := 0 \\ \\ \quad \mathbf{O}'(\mathbf{y}! : \{0,1\}) & \left(\begin{array}{l} \mathbf{x}_0 = 0 \wedge \mathbf{x} = 1 \wedge \mathbf{y} = 0 \\ \vee \\ \mathbf{x}_0 = 1 \wedge \mathbf{x} = 2 \wedge \mathbf{y} = 1 \\ \vee \\ \mathbf{x}_0 = 1 \wedge \mathbf{x} = 3 \wedge \mathbf{y} = 0 \end{array}\right) \\ \\ \quad \textbf{finally}' & \textbf{skip} \\ \textbf{end}. & \end{array}$$

Evidently there is no *dsim* from $\mathbf{T}$ to $\mathbf{T}'$ (although there *is* a *usim* from $\mathbf{T}'$ to $\mathbf{T}$).

Dually, by an analogous example, *usim* is incomplete.

Completeness

However jointly the two simulation rules are complete for data refinement.

Theorem(Completeness theorem). If $\mathbf{T} \sqsubseteq \mathbf{T}'$ then there is a *usim* $\mathbf{u}$ and a *dsim* $\mathbf{d}$ for which $\mathbf{d} \circ \mathbf{u}$ is a relation from $\mathbf{L}$ to $\mathbf{L}'$.

Corollary(of the method). If the operations of $\mathbf{T}$ are functions and if $\mathbf{T} \sqsubseteq \mathbf{T}'$ then there is a *dsim* from $\mathbf{T}$ to $\mathbf{T}'$.

Thus the simulation rule *dsim* is complete for data types whose operations are total and deterministic.

6 Predicate-transformer semantics

Introduction

We now consider a more expressive model for computation which contains not only $\mathcal{R}$ but also other important computational entities. Its lattice structure ensures that a single simulation rule is complete for data refinement, and it provides reasonable calculational facility.

Predicate transformers

Definition

For state space $\mathbf{X}$, we continue to identify $\mathbf{PX}$ with the space of Boolean-valued functions (that is, predicates) on $\mathbf{X}$. The set, $\mathcal{T}$, of *predicate transformers* on $\mathbf{X}$ is defined $\mathcal{T} \mathrel{\widehat{=}} \mathbf{PX} \to \mathbf{PX}$.

A predicate, $\mathbf{q}$, on $\mathbf{X}$ which is thought of as being a predicate on final states (that is, after a computation) is called a *postcondition*. One, $\mathbf{p}$, which is thought of as being a predicate on initial states (that is, before a computation) is called a *precondition*. Predicate transformer $\mathbf{t} : \mathcal{T}$ represents a computation by mapping postconditions to preconditions as follows. For each postcondition $\mathbf{q} : \mathbf{PX}$, precondition $\mathbf{t.q}$ holds at just those initial states from which computation $\mathbf{t}$ is guaranteed to terminate in a state satisfying $\mathbf{q}$. (The phrase "is guaranteed to terminate" is included to emphasise application of the definition to nondeterministic computations.)

For each $\mathbf{t} : \mathcal{T}$ and $\mathbf{q} : \mathbf{PX}$, the predicate $\mathbf{t.q}$ is called the *weakest precondition* of computation $\mathbf{t}$ at postcondition $\mathbf{q}$.

Predicate transformers are ordered with the partial order lifted pointwise from implication on $\mathbf{PX}$

$$\mathbf{t} \sqsubseteq \mathbf{u} \mathrel{\widehat{=}} \forall \mathbf{q} : \mathbf{PX} \bullet \mathbf{t.q} \leq \mathbf{u.q} .$$

Thus $\mathbf{t}$ is refined by $\mathbf{u}$ iff for each postcondition $\mathbf{q}$, the weakest precondition of $\mathbf{t}$ at $\mathbf{q}$ is at least as strong as the weakest precondition of $\mathbf{u}$ at $\mathbf{q}$. Thus whenever $\mathbf{t}$ achieves $\mathbf{q}$ so too does $\mathbf{u}$.

The least transformer is the function returning *false* everywhere (representing **abort**); the greatest transformer is the function returning *true* everywhere (representing **magic**).

The Galois connection with relations

We now define two functions

$$\mathbf{wp} : \mathcal{R} \to \mathcal{T} \quad \text{the embedding } \varepsilon$$

$$\mathbf{rp} : \mathcal{T} \to \mathcal{R} \quad \text{the projection } \pi$$

between the lattices $(\mathcal{R}, \supseteq)$ and $(\mathcal{T}, \sqsubseteq)$.

The embedding $\mathbf{wp}$

For a relational computation $\mathbf{r} : \mathcal{R}$ and postcondition $\mathbf{q} : \mathbf{PX}$, define

$$\mathbf{wp.r.q.x} \mathrel{\widehat{=}} \forall \mathbf{y} : \mathbf{X}_\perp \bullet \mathbf{x\,r\,y} \Rightarrow \mathbf{q.y} .$$

The projection $\mathbf{rp}$

For a predicate transformer $\mathbf{t} : \mathcal{T}$, an intial state $\mathbf{x} : \mathbf{X}$ and a possibly improper final state $\mathbf{y} : \mathbf{X}_\perp$, define

$$\mathbf{x\,(rp.t)\,y} \mathrel{\widehat{=}} \forall \mathbf{q} : \mathbf{PX} \bullet \mathbf{t.q.x} \Rightarrow \mathbf{q.y} .$$

Galois properties

For comparison with the definition of $\mathbf{gc}(\varepsilon, \pi; \mathbf{X}, \mathbf{Y})$, note the correspondence:

$$\begin{array}{ccccccc} \mathbf{x} & \leq & \pi.\mathbf{y} & \equiv & \varepsilon.\mathbf{x} & \leq & \mathbf{y} \\ & \updownarrow & & & & \updownarrow & \\ \mathbf{r} & \subseteq & \mathbf{rp.t} & \equiv & \mathbf{wp.r} & \sqsupseteq & \mathbf{t}. \end{array}$$

Of course both formalise "the transformer is refined by the relation".

Consequences

The first two properties follow directly from the Galois connection; proof of the last is an exercise.

1. For all $\mathbf{r} : \mathcal{R}$, $\mathbf{r} \subseteq \mathbf{rp.(wp.r)}$.
2. For all $\mathbf{t} : \mathcal{T}$, $\mathbf{t} \sqsubseteq \mathbf{wp.(rp.r)}$.
3. The embedding is injective: $\mathbf{rp.(wp.r)} = \mathbf{r}$. Thus the pair of functions forms a Galois embedding

$$\mathbf{ge(wp, rp}; \mathcal{R}, \mathcal{T}).$$

Healthiness conditions

We now recall Dijkstra's healthiness conditions for language $\mathcal{L}$, in the predicate-transformer model. For $\mathbf{r} : \mathcal{D}$ we have

1. *The excluded miracle*: $\mathbf{wp.r.}\mathit{false} = \mathit{false}$.
2. *Monotonicity*: if $\mathbf{q} \Rightarrow \mathbf{q'}$ then $\mathbf{wp.r.q} \Rightarrow \mathbf{wp.r.q'}$.
3. *Conjunctivity*: $\mathbf{wp.r.(q} \wedge \mathbf{q'}) = \mathbf{wp.r.q} \wedge \mathbf{wp.r.q'}$.
4. *Disjunctivity*: if $\mathbf{r}$ is predeterministic (deterministic unless aborting) then

$$\mathbf{wp.r.(q} \vee \mathbf{q'}) = \mathbf{wp.r.q} \vee \mathbf{wp.r.q'}.$$

5. *Continuity*: if $\mathbf{Q}$ is a directed set of predicates in the space $(\mathbf{PX}, \leq)$ then

$$\mathbf{wp.r.} \sqcup \mathbf{Q} = \sqcup \mathbf{wp.r.Q}.$$

The $\mathbf{wp}$-image of $\mathcal{R}$ is characterised by *positive* $\sqcup$-*junctivity*: for any nonempty set $\mathbf{Q}$ of postconditions,

$$\mathbf{wp.r.} \sqcup \mathbf{Q} \;=\; \sqcup\{\mathbf{wp.r.q} \mid \mathbf{q} : \mathbf{Q}\}.$$

Note that (positive) $\sqcup$-junctivity implies monotonicity.

Other healthiness-style facts include

1. If $\mathbf{r}$ is finitary then $\mathbf{wp.r}$ is $\vee$-continuous.
2. If $\mathbf{r}$ is total then $\mathbf{wp.r}$ is strict (no miracles).
3. If $\mathbf{t}$ is $\vee$-continuous then positive conjunctivity is equivalent to binary conjunctivity.
4. $\mathbf{wp.r}$ is disjunctive iff $\mathbf{r}$ is predeterministic.

wp is a functor

Since both $\mathcal{R}$ and $\mathcal{T}$ contain identities and are closed under composition, it is meaningful to seek their preservation under $\mathbf{wp}$. We have

1. $\mathbf{wp}.\iota_{\mathcal{R}} = \iota_{\mathcal{T}}$ and
2. $\mathbf{wp}.(\mathbf{r} \fatsemi \mathbf{s}) = \mathbf{wp.r} \fatsemi \mathbf{wp.s}$,

which together mean that $\mathbf{wp}$ is a (covariant?) functor.

The structure on transformers

Self-duality: conjugates

For transformer $\mathbf{t} : \mathcal{T}$, recall that its *conjugate* $\mathbf{t}^*$ is defined (see section 2) $\mathbf{t}^*.\mathbf{q} \;\widehat{=}\; \neg\mathbf{t}.\neg\mathbf{q}$.

The dual lattice $(\mathcal{T}^*, \sqsupseteq)$ is isomorphic to the original $(\mathcal{T}, \sqsubseteq)$ with, for example,

1. $\mathbf{false}^* = \mathbf{true}$, and $\mathbf{true}^* = \mathbf{false}$;
2. $\mathbf{t}^{**} = \mathbf{t}$;
3. $\mathbf{t}$ monotonic iff $\mathbf{t}^*$ monotonic;
4. $(\mathbf{t} \fatsemi \mathbf{u})^* = \mathbf{t}^* \fatsemi \mathbf{u}^*$;
5. $\mathbf{t} \sqsubseteq \mathbf{u}$ iff $\mathbf{t}^* \sqsupseteq \mathbf{u}^*$; and
6. $(\mathbf{t} \sqcap \mathbf{u})^* = \mathbf{t}^* \sqcup \mathbf{u}^*$.

The operator $\sqcup$, created in the last item above by duality, is *angelic choice*: $(\mathbf{t} \sqcup \mathbf{u}).\mathbf{q} \;\widehat{=}\; \mathbf{t.q} \vee \mathbf{u.q}$.

Non-relational transformers

Transformers $\mathbf{t}$ for which $\mathbf{t}.\mathit{false} \neq \mathit{false}$ are *infeasible*, or *miraculous*, and correspond (if conjunctive) to relations that are not total: for some initial states there is "no" final state, and the computation "cannot be started".

Transformers $\mathbf{t}$ for which $\mathbf{t}.(\sqcup\mathbf{Q}) \neq \sqcup\{\mathbf{t.q} \mid \mathbf{q} : \mathbf{Q}\}$ are *not continuous*, and correspond (if conjunctive) to relations that are not image finite; for some initial states there are infinitely many possible final states.

Transformers $\mathbf{t}$ for which $\mathbf{t}.(\mathbf{q} \wedge \mathbf{q'}) \neq \mathbf{t.q} \wedge \mathbf{t.q'}$ are *not conjunctive*; they do not correspond to relations at all. Angelic choice does not preserve conjunctivity.

We continue to concentrate on transformers that are monotonic.

Angelic nondeterminism

Let $\mathbf{X}$ be the integers, and let $\mathbf{t}^+, \mathbf{t}^-$ be the predicate transformers corresponding to programs that set the final state demonically to some non-negative, non-positive value.

The program $\mathbf{t}^0 \;\widehat{=}\; \mathbf{t}^+ \sqcup \mathbf{t}^-$ is not conjunctive, and so does not correspond to any relation; in particular it does not represent the (deterministic) program "set the final state to 0" (which is the intersection of the two relations $\mathbf{rp.t}^+$ and $\mathbf{rp.t}^-$).

Execution can however be arranged by backtracking; but that requires a more elaborate model than the relational.

Characterisation of the monotonic transformers

The following theorem shows that there are no more unusual elements lurking in the monotonic transformer space than those already discovered:

> **Theorem(transformer factorisation).** Any monotonic predicate transformer can be written as the angelic choice of conjunctive (that is, terminating relational) transformers.

The theorem means that execution of a (general) monotonic predicate transformer can be interpreted as a game of moves alternating between a player (who moves angelically) and an adversary (who moves demonically).

Strongest postconditions

A second embedding

Define two functions

$$\mathbf{sp} : \mathcal{R} \to \mathcal{T} \quad \text{the embedding } \varepsilon$$

$$\mathbf{qp} : \mathcal{T} \to \mathcal{R} \quad \text{the projection } \pi$$

between the lattices $(\mathcal{R}, \supseteq)$ and $(\mathcal{T}, \sqsubseteq)$, with

$$\mathbf{sp.r.q} \ni \mathbf{x} \;\widehat{=}\; \exists \mathbf{y} : \mathbf{X} \bullet \begin{pmatrix} \mathbf{q.y} \\ \mathbf{y\,r\,x} \end{pmatrix}.$$

The name **sp** stands for *strongest postcondition* (and "$\ni$" is pronounced "epsiloff").

It is possible to define (indeed, to calculate) a definition for **qp** that makes **(sp, qp)** a Galois connection.

wp *versus* **sp**

For each relation $\mathbf{r} : \mathcal{R}$ the two transformers **wp.r** and **sp.r** are adjoint (but see section 7).

Weakest liberal preconditions

Purpose of **wlp**

The weakest *liberal* precondition makes behavioural distinctions that **wp** cannot detect, and *vice versa*. For example

$$\mathbf{skip} =_{\mathbf{wlp}} \mathbf{skip} \sqcap \mathbf{abort} =_{\mathbf{wp}} \mathbf{abort}.$$

The distinction is based on *not* assuming that **abort** "can do anything" (as well as fail to terminate). For example, in that interpretation the program

$$\mathbf{x} := 0 \sqcap \mathbf{abort}$$

might fail to set variable **x** to 0, but it can't set **x** to 1.

The relational model is as before, except that we no longer require $\mathbf{x\,r\,y}$ whenever $\mathbf{x\,r} \perp$.

Definition of **wlp**

We define two functions

$$\mathbf{wlp}: \quad \mathcal{R} \to \mathcal{T} \quad \text{the embedding } \varepsilon$$

$$\mathbf{rlp}: \quad \mathcal{R} \to \mathcal{T} \quad \text{the projection } \pi$$

between the lattices $(\mathcal{R}, \supseteq)$ and $(\mathcal{T}, \sqsubseteq)$, with

$$\mathbf{wlp.r.q.x} \;\widehat{=}\; \forall \mathbf{y} : \mathbf{X}_{\perp} \bullet \mathbf{x\,r\,y} \Rightarrow (\mathbf{y} = \perp \vee \mathbf{q.x}).$$

Thus **wlp.r.q** holds at just those initial states from which execution of **r** either diverges or terminates in a state satisfying **q**.

The Galois property can be used to calculate a corresponding definition for **rlp**.

The **wp** *and* **wlp** *orders*

On the (new) relational space, the **wp** order is derived from the *Smyth* preorder on subsets of **X**:

$$\mathbf{L} \sqsubseteq_{\mathbf{S}} \mathbf{H} \;\widehat{=}\; \forall \mathbf{h} : \mathbf{H} \bullet \exists \mathbf{l} : \mathbf{L} \bullet \mathbf{l} \sqsubseteq \mathbf{h}.$$

We have

$$\mathbf{r} \sqsubseteq_{\mathbf{wp}} \mathbf{s} \;\widehat{=}\; \forall \mathbf{x} : \mathbf{X} \bullet \mathbf{x}.\langle \mathbf{r} \rangle \sqsubseteq_{\mathbf{S}} \mathbf{x}.\langle \mathbf{s} \rangle,$$

where as before $\mathbf{x}.\langle \mathbf{r} \rangle$ denotes the image of $\{\mathbf{x}\}$ through relation **r**.

"Fluffing up" converts the preorder to an ordinary partial order, and $\sqsubseteq_{\mathbf{S}}$ becomes simple $\supseteq$.

The **wlp** order is derived from the *Hoare* pre-order on subsets of **X**:

$$\mathbf{L} \sqsubseteq_{\mathbf{H}} \mathbf{H} \;\widehat{=}\; \forall \mathbf{l} : \mathbf{L} \bullet \exists \mathbf{h} : \mathbf{H} \bullet \mathbf{l} \sqsubseteq \mathbf{h}.$$

We have $\mathbf{r} \sqsubseteq_{\mathbf{wlp}} \mathbf{s} \;\widehat{=}\; \forall \mathbf{x} : \mathbf{X} \bullet \mathbf{x}.\langle \mathbf{r} \rangle \sqsupseteq_{\mathbf{H}} \mathbf{x}.\langle \mathbf{s} \rangle$; note the reversal of the order. "Fluffing down" converts the preorder to an ordinary partial order, and $\sqsubseteq_{\mathbf{H}}$ becomes simple $\subseteq$.

7 Predicate-transformer data refinement

In this chapter we return to *data refinement*—last considered relationally—in the setting of predicate transformers. The result is more than a simple reformulation; with the extra structure of predicate transformers we have a single complete rule for data refinement.

Data refinement

Recall that a *datatype* is defined by its initialisation **in**, its family of operations $\mathcal{O}$ (an indexed set) and its finalisation **fin**.

A *use* of a datatype is a composition $\mathbf{in} ; \mathbf{P}(\mathcal{O}) ; \mathbf{fin}$ for some program scheme **P** referring by index to individual operations in $\mathcal{O}$.

One datatype $(\mathbf{in}, \mathcal{O}, \mathbf{fin})$ is *refined by* another data type $(\mathbf{in}', \mathcal{O}', \mathbf{fin}')$ if for all uses **P** we have

$$\mathbf{in} ; \mathbf{P}(\mathcal{O}); \mathbf{fin} \sqsubseteq \mathbf{in}' ; \mathbf{P}'(\mathcal{O}') ; \mathbf{fin}',$$

with $\sqsubseteq$ denoting ordinary refinement between programs.

Informally, $(\mathbf{in}, \mathcal{O}, \mathbf{fin})$ is refined by $(\mathbf{in}', \mathcal{O}', \mathbf{fin}')$ if no program **P** can show that $(\mathbf{in}', \mathcal{O}', \mathbf{fin}')$ behaves in a way that $(\mathbf{in}, \mathcal{O}, \mathbf{fin})$ could not.

Simulation

Simulation is a relationship established component-wise between two datatypes, as below, in order to show data refinement between the datatypes. It avoids having to consider all their possible uses **P**.

Recall that there are two kinds of simulation. A *dsim* is an operation **rep** satisfying

in: $\mathbf{in} ; \mathbf{rep} \sqsubseteq \mathbf{in}'$,

op: for all corresponding $\mathbf{O}, \mathbf{O}'$ in $\mathcal{O}, \mathcal{O}'$ respectively

$$\mathbf{O} \fatsemi \mathbf{rep} \sqsubseteq \mathbf{rep} \fatsemi \mathbf{O}' \text{ and}$$

fin: $\mathbf{fin} \sqsubseteq \mathbf{rep} \fatsemi \mathbf{fin}'$.

A *usim* is an operation **rep**(which in the presence of recursion **rep** must also be continuous and strict) satisfying

in: $\mathbf{in} \sqsubseteq \mathbf{rep} \fatsemi \mathbf{in}'$,

op: for all corresponding $\mathbf{O}, \mathbf{O}'$ in $\mathcal{O}, \mathcal{O}'$ respectively

$$\mathbf{rep} \fatsemi \mathbf{O} \sqsubseteq \mathbf{O}' \fatsemi \mathbf{rep}, \text{ and}$$

fin: $\mathbf{fin} \fatsemi \mathbf{rep} \sqsubseteq \mathbf{fin}'$.

We have seen that in the relational setting, those simulations are sound and jointly complete.

Weak inverses

Any conjunctive *and terminating* predicate transformer $\mathbf{P}$ corresponds to some relation and so can be written $\mathbf{wp.r}$ for that relation $\mathbf{r}$.[1] It therefore has an adjoint $\mathbf{sp.r}$, which we write $\widehat{\mathbf{P}}$, satisfying

$$\begin{array}{rcl} \iota & \sqsubseteq & \mathbf{P} \fatsemi \widehat{\mathbf{P}} \\ \widehat{\mathbf{P}} \fatsemi \mathbf{P} & \sqsubseteq & \iota. \end{array}$$

The requirement that $\mathbf{P}$ be terminating is necessary, corresponding in lattice terms to $\mathbf{P}.\mathit{true} = \mathit{true}$. To that extent section 6 was not telling the whole truth.

Construction of the data refinement

Given two datatypes with

$$(\mathbf{in}, \mathcal{O}, \mathbf{fin}) \sqsubseteq (\mathbf{in}', \mathcal{O}', \mathbf{fin}')$$

in proving the completeness theorem in section 5 we constructed a simulation whose form we mimick now. Set

$$\mathbf{rep} \;\widehat{=}\; \sqcup\{(\widehat{\mathbf{in}' \fatsemi \mathbf{S}'}) \fatsemi \mathbf{in} \fatsemi \mathbf{S} \mid \mathbf{S} \in \mathcal{O}^*\},$$

where $\mathcal{O}^*$ denotes the closure of $\mathcal{O}$ under sequential composition.

The proofs of the three properties are then direct, if not completely obvious.

Nontermination

When $\mathbf{P}$ does not necessarily terminate, it may be that the adjoint $\widehat{\mathbf{P}}$ does not exist. Luckily it is possible to generalise that definition, by using assumptions, to

$$\begin{array}{rcl} \{\mathbf{wp.P}.\mathit{true}\} & \sqsubseteq & \mathbf{P} \fatsemi \widehat{\mathbf{P}} \\ \widehat{\mathbf{P}} \fatsemi \mathbf{P} & \sqsubseteq & \iota, \end{array}$$

where $\{\mathbf{wp.P}.\mathit{true}\}$ is a program that aborts when $\mathbf{P}$ would abort, and behaves like **skip** otherwise (when $\mathbf{P}$ is guaranteed to terminate).

[1] We write $\mathbf{P}$ rather than $\mathbf{t}$ for transformers here, to be consistent with the convention for datatypes.

The proof of completeness then goes through with the modified definition, but for *soundness* in the presence of possible failure to terminate we require that **rep** be strict and $\sqcup$-continuous. And the **rep** generated by the completeness construction is guaranteed to be so only if the abstract program itself is continuous—it must contain no unbounded nondeterminism.

8 Process semantics

Introduction

We now turn to another programming paradigm: that of process algebra. Settling upon the notation CSP we study its hierarchy of three progressively finer semantic models which are usually linked by the natural embeddings up the hierarchy. In order to obtain Galois connections we replace those embeddings by alternatives, based on a version of "fluffing up" appropriate to interactive processes, and identify their adjoints.

The result is that the standard hierarchy of models for CSP now becomes founded on embeddings that are capable of straightforward extension when it is desired to incorporate the results of further observations in the semantics. Such extensions are to be found, for example, in timed models and probabilistic models.

Processes

A *process*, $\mathbf{P}$, has a finite *alphabet*, $\alpha\mathbf{P}$, of *events* in which it may engage. It is expressed algebraically, according to which events it may next perform. We consider a slightly restricted subset of standard CSP, ignoring termination, sequential composition and interleaving using only the following combinators.

- *General choice*, $(\mathbf{x} : \mathbf{X} \to \mathbf{P}(\mathbf{x}))$, offers its environment any of the events $\mathbf{x} : \mathbf{X}$; after performing $\mathbf{x}$ it behaves like $\mathbf{P}(\mathbf{x})$. Special cases are
  - $\mathbf{X} = \{\}$, called *deadlock* and written **stop**
  - $\mathbf{X} = \{\mathbf{x}\}$, called *prefix* and written $\mathbf{x} \to \mathbf{P}$
  - $\mathbf{X} = \{\mathbf{x}, \mathbf{y}\}$, called *choice* and written

$$(\mathbf{x} \to \mathbf{P}(\mathbf{x}) \mid \mathbf{y} \to \mathbf{P}(\mathbf{y})).$$

- *Recursion*, $\mu\,\mathbf{Z} : \mathbf{A} \bullet \mathbf{F}.\mathbf{Z}$, is the least fixed point of continuous function $\mathbf{F}$ on processes having alphabet $\mathbf{A}$. For example

$$\mu\,\mathbf{Z} : \{0, 1\} \bullet 0 \to 1 \to \mathbf{Z}$$

  alternately performs 0 then 1 forever.
- *Parallel composition*, $\mathbf{P} \parallel \mathbf{Q}$, synchronises on events common to $\mathbf{P}$ and $\mathbf{Q}$ and interleaves others.
- *External choice*, $\mathbf{P} \Box \mathbf{Q}$, offers its environment the choice between $\mathbf{P}$ and $\mathbf{Q}$ on the first interaction.
- *Abstraction*, $\mathbf{P}$ ***hide*** $\mathbf{E}$, for $\mathbf{E} \subseteq \alpha\mathbf{P}$, behaves like $\mathbf{P}$ but with the events in $\mathbf{E}$ concealed, or internalised.
- *Nondeterministic choice*, $\mathbf{P} \sqcap \mathbf{Q}$, arises from abstraction. It behaves like either $\mathbf{P}$ or $\mathbf{Q}$, but the environment

has no influence over which.

- *Abort*, **chaos**, exhibits arbitrary behaviour.

Refinement between processes, $\mathbf{P} \sqsubseteq \mathbf{Q}$, holds iff $\mathbf{P}$ and $\mathbf{Q}$ have the same alphabet, $\alpha\mathbf{P} = \alpha\mathbf{Q}$, and in every environment $\mathbf{P}$ can be replaced by $\mathbf{Q}$. As usual equality between processes, $\mathbf{P} = \mathbf{Q}$, means refinement of each side by the other. Laws to formalise the concept of refinement include

$$\mathbf{x} \rightarrow (\mathbf{P} \parallel \mathbf{Q}) = (\mathbf{x} \rightarrow \mathbf{P}) \parallel (\mathbf{x} \rightarrow \mathbf{Q})$$

$$(\mathbf{x} \rightarrow \mathbf{P} \Box \mathbf{x} \rightarrow \mathbf{Q})\ \mathit{hide}\ \{\mathbf{x}\}$$
$$=$$
$$(\mathbf{P}\ \mathit{hide}\ \{\mathbf{x}\}) \sqcap (\mathbf{Q}\ \mathit{hide}\ \{\mathbf{x}\})$$

$$\mathbf{P} \sqcap \mathbf{Q} \sqsubseteq \mathbf{P}\,.$$

Models

There are three standard models of CSP: traces; failures; and divergences. They capture increasingly more detailed behaviour of a process: safety; liveness; and divergence. We now summarise them, reminding the reader of the healthiness conditions for each.

Traces

The *traces model* of CSP captures safety properties and is complete for deterministic processes. In it, each process is denoted by the set of finite sequences of events in which it can engage, called its *traces* (defined by structural induction). One process refines another if it can perform all the traces of that other.

Thus for finite universe $\mathbf{A}$ of events, the *traces model* over $\mathbf{A}$ is defined to be the space $(\mathcal{T}, \sqsubseteq_{\mathcal{T}})$ where

$$\mathcal{T}$$
$$\widehat{=}$$
$$\{\mathbf{T} \subseteq \mathbf{seq}\ \mathbf{A} \mid \mathbf{T} \text{ is nonempty and prefix closed}\}$$

and where $\mathbf{T} \sqsubseteq_{\mathcal{T}} \mathbf{U} \ \widehat{=}\ \mathbf{T} \subseteq \mathbf{U}$.

The least process is **stop** which performs no events at all. So recursion is given, for continuous $\mathbf{F}$, by

$$[\mu\,\mathbf{Z} \bullet \mathbf{F}.\mathbf{Z}] = \bigcup_{\mathbf{n}:\mathbf{N}} [\mathbf{F}^{\mathbf{n}}.\mathbf{stop}]\,.$$

However traces refinement is unable to distinguish $\mathbf{P} \sqcap \mathbf{stop}$ from $\mathbf{P}$, since their traces are the same. Thus development based on it could not ensure absence of deadlock! It is also unable to distinguish internal and external choice $[\mathbf{x} \rightarrow \mathbf{P} \Box \mathbf{y} \rightarrow \mathbf{Q}] = [\mathbf{x} \rightarrow \mathbf{P} \sqcap \mathbf{y} \rightarrow \mathbf{Q}]$. That distinction lies with the failures model!

Failures

The *failures model* of CSP captures liveness properties and is complete for nondivergent processes. In it, each process is denoted by a relation between the set of traces of the process and the set of all subsets of its alphabet, relating each trace to the sets of events which may be refused (that is, may lead to deadlock if its environment offers them) by the process immediately after it has engaged in that trace. Such failures information is sufficient to express nondeterminism, with the result that a process which is less deterministic than another contains the failures of that other.

The failures of CSP are defined by structural induction. In the failures model one process refines another if its behaviour is at least as deterministic.

Thus for universe $\mathbf{A}$ of events, the *failures model* of CSP is defined to be the space $(\mathcal{F}, \sqsubseteq_{\mathcal{F}})$ where $\mathcal{F}$ denotes the set of relations $\mathbf{F}$ from $\mathbf{seq}\ \mathbf{A}$ to subsets of $\mathbf{A}$ satisfying:

$$\left(\begin{array}{l} \mathbf{dom}\ \mathbf{F} \in \mathcal{T} \\ \left(\begin{array}{l} \mathbf{t}\ \mathbf{F}\ \mathbf{E} \\ \mathbf{D} \subseteq \mathbf{E} \end{array} \right) \Rightarrow \mathbf{t}\ \mathbf{F}\ \mathbf{D} \\ \left(\begin{array}{l} \mathbf{t}\ \mathbf{F}\ \mathbf{E} \\ \mathbf{x} \in \mathbf{A} \end{array} \right) \Rightarrow \begin{array}{l} \mathbf{t}\ \mathbf{F}\ (\mathbf{E} \cup \{\mathbf{x}\}) \\ \vee \\ \mathbf{t}\langle \mathbf{x} \rangle\ \mathbf{F}\ \{\} \end{array} \end{array} \right),$$

and where $\mathbf{F} \sqsubseteq_{\mathcal{F}} \mathbf{G} \ \widehat{=}\ \mathbf{F} \supseteq \mathbf{G}$.

The least process is **havoc** which may perform or refuse any element in $\mathbf{A}$ at any interaction (but never diverges). So recursion is given, for continuous $\mathbf{F}$, by

$$[\mu\,\mathbf{Z} \bullet \mathbf{F}.\mathbf{Z}] = \bigcap_{\mathbf{n}:\mathbf{N}} [\mathbf{F}^{\mathbf{n}}.\mathbf{havoc}]\,.$$

In the failures model nondeterminism can be distinguished; in particular $\mathbf{P}$ is distinguished from $\mathbf{P} \sqcap \mathbf{stop}$ (which it cannot in the traces model). Indeed a process is *nondeterministic* means at some interaction it can both perform and refuse an event. Thus $\mathbf{P} : \mathcal{F}$ is *deterministic* means

$$\mathbf{t}\langle \mathbf{a} \rangle \in \mathbf{dom}\ \mathbf{P} \Rightarrow \neg \mathbf{t}\ \mathbf{P}\ \{\mathbf{a}\}\,,$$

and otherwise $\mathbf{P}$ is nondeterministic. For example, $\mathbf{x} \rightarrow \mathbf{P} \Box \mathbf{y} \rightarrow \mathbf{Q}$ is determinitic but $\mathbf{x} \rightarrow \mathbf{P} \sqcap \mathbf{y} \rightarrow \mathbf{Q}$ is not.

However the failures model is unable to distinguish arbitrary refusal behaviour from *livelock* (or divergence): $[\mathbf{havoc}] = [\mathbf{chaos}]$. For that we must turn to the divergences model.

Divergences

The *divergences model* of CSP captures divergence information and is complete for CSP. In it, each process is denoted by a pair $(\mathbf{F}, \mathbf{D})$ where $\mathbf{F}$ is the failures, as above, and $\mathbf{D}$ is the set of traces after which the process may diverge. For consistency with $\mathcal{F}$, if $\mathbf{P}$ diverges at trace $\mathbf{t}$ then after having engaged in $\mathbf{t}$, $\mathbf{P}$ can also subsequently perform or refuse any event. One process refines another if it is at least as deterministic and at least as nondivergent.

Thus for universe $\mathbf{A}$, the *divergences model* of CSP is defined to be the space $(\mathcal{C}, \sqsubseteq_{\mathcal{C}})$ where $\mathcal{C}$ denotes the set of pairs $(\mathbf{F}, \mathbf{D})$ with $\mathbf{F} \in \mathcal{F}$ and $\mathbf{D} \subseteq \mathbf{dom}.\mathbf{F}$ satisfying

$$\left(\begin{array}{l} \left(\begin{array}{l} \mathbf{t} \in \mathbf{D} \\ \mathbf{u} \in \mathbf{seq}\ \mathbf{A} \end{array} \right) \Rightarrow \mathbf{tu} \in \mathbf{D} \\ \left(\begin{array}{l} \mathbf{t} \in \mathbf{D} \\ \mathbf{E} \subseteq \mathbf{A} \end{array} \right) \Rightarrow \mathbf{t}\ \mathbf{F}\ \mathbf{E} \end{array} \right),$$

and where

$$(\mathbf{F},\mathbf{D}) \sqsubseteq_{\mathcal{C}} (\mathbf{G},\mathbf{E}) \;\widehat{=}\; \begin{pmatrix} \mathbf{F} \supseteq \mathbf{G} \\ \mathbf{D} \supseteq \mathbf{E} \end{pmatrix}.$$

The least process is **chaos** which may diverge, or perform, or refuse any element in $\mathbf{A}$ at any interaction.

In the divergences model divergence is distinguish- ed from arbitrary refusal behaviour; in particular **havoc** is distinct from **chaos** since the former has no divergences whilst the latter has every possible sequence as a divergence.

Summary

As summary of those models we have:

> **Theorem(CSP domains).** Each of the spaces $(\mathcal{T},\sqsubseteq_{\mathcal{T}})$, $(\mathcal{F},\sqsubseteq_{\mathcal{F}})$ and $(\mathcal{C},\sqsubseteq_{\mathcal{C}})$ is a domain and the operations of CSP provide continuous functions on each of them.

Embedding $\mathcal{T}$ in $\mathcal{F}$

In this section we replace the standard embedding of $\mathcal{T}$ in $\mathcal{F}$ by one which forms a Galois embedding.

The standard embedding

The *deterministic embedding* $\hat{}\; : \mathcal{T} \to \mathcal{F}$ assigns to traces process $\mathbf{D}$ the failures process $\mathbf{D}\hat{}$ which performs all events that $\mathbf{D}$ performs and refuses all others:

$$\mathbf{t}\,\mathbf{D}\hat{}\,\mathbf{E} \;\widehat{=}\; \begin{pmatrix} \mathbf{t} \in \mathbf{D} \\ \forall \mathbf{a} : \mathbf{A} \bullet \mathbf{t}\langle \mathbf{a} \rangle \in \mathbf{D} \Rightarrow \mathbf{a} \notin \mathbf{E} \end{pmatrix}.$$

Since it is unable to refuse any event it performs, $\mathbf{D}\hat{}$ is deterministic. Each trace it performs is a trace of $\mathbf{D}$ and conversely. Thus:

> **Lemma(Deterministic embedding).** $\mathbf{D}\hat{}$ is deterministic with the same traces as $\mathbf{D}$.

However in order to be the embedding of a Galois connection, $\hat{}$ would have to be monotone (by the equivalence theorem of section 3). Yet it is not; for in $\mathcal{T}$, $\mathbf{stop} \sqsubseteq_{\mathcal{T}} \mathbf{a} \to \mathbf{stop}$ since the latter can perform every trace the former can. But in $\mathcal{F}$ that refinement fails since

$$\begin{pmatrix} \neg\langle \mathbf{a} \rangle\, \mathbf{stop}\,\{\,\} \\ \langle \mathbf{a} \rangle\, (\mathbf{a} \to \mathbf{stop})\,\{\,\} \end{pmatrix}.$$

We must look a little further.

Recall that in section 4 we embedded the space $\mathcal{P}$ of partial functions in the space $\mathcal{D}$ of total relations by an embedding ε which "fluffed up" its argument: outside the domain of partial function $\mathbf{f}$, $\varepsilon.\mathbf{f}$ was as unconstrained as possible, whilst on $\mathbf{dom.f}$ it equalled $\mathbf{f}$. Being as unconstrained as possible meant behaving like the least element, **abort**, the universal relation.

Now we are in an identical situation. In order to find a Galois connection from $\mathcal{T}$ to $\mathcal{F}$ we must "fluff up" a traces process $\mathbf{D}$ so that it behaves in as unconstrained a manner as possible unless it behaves like $\mathbf{D}$. Thus $\varepsilon.\mathbf{D}$ behaves like the least element **chaos** unless $\mathbf{D}$ perfroms an event in which case $\varepsilon.\mathbf{D}$ cannot refuse that event. This leads us to the following definition.

The liberal embedding

The *liberal embedding* $^{+} : \mathcal{T} \to \mathcal{F}$ assigns to $\mathbf{D} : \mathcal{T}$ the process $\mathbf{D}^{+} : \mathcal{F}$ whose only constraint is that it does not refuse an event performed by $\mathbf{D}$:

$$\mathbf{t}\,\mathbf{D}^{+}\,\mathbf{E} \;\widehat{=}\; \forall \mathbf{a} : \mathbf{A} \bullet \mathbf{t}\langle \mathbf{a} \rangle \in \mathbf{D} \Rightarrow \mathbf{a} \notin \mathbf{E}.$$

Since $\mathbf{D}^{+}$ has arbitrary traces it is, in general, nondeterministic (unlike $\mathbf{D}\hat{}$). The following result establishes that claim together with two results about refinement of $\mathbf{D}\hat{}$ in $\mathcal{F}$.

> **Lemma(Liberal embedding).** $\mathbf{D}^{+}$ is refined in $\mathcal{F}$ by $\mathbf{D}\hat{}$. Every sequence from its alphabet is a trace of $\mathbf{D}^{+}$, and if $\mathbf{D}^{+}$ is refined in $\mathcal{F}$ by $\mathbf{P}$ then every trace of $\mathbf{P}$ is a trace of $\mathbf{D}$.

Calculating the adjoint

To identify the adjoint of $^{+}$ we have (approach (c) works well) $^{-} : \mathcal{F} \to \mathcal{T}$ defined

$$\mathbf{P}^{-} \;\widehat{=}\; \{\mathbf{r} : \mathbf{dom.P} \mid \mathbf{u}\langle \mathbf{a} \rangle \leq \mathbf{r} \;\Rightarrow\; \neg \mathbf{u}\,\mathbf{P}\,\{\mathbf{a}\}\}.$$

Function $^{-}$ thus retains deterministic behaviour by discarding events that are both performed and refused.

A Galois connection

The manipulation above provides us with the Galois connection we seek between $\mathcal{T}$ and $\mathcal{F}$. In fact the embedding and projection form a Galois embedding (and moreover the projection is also $\sqcup$-junctive:

> **Theorem(Liberal connection).** The liberal embedding $^{+}$ and its adjoint $^{-}$ form a Galois embedding: $\mathbf{ge}(^{+},^{-}; \mathcal{T}, \mathcal{F})$.

Since the Galois connection is a Galois embedding, the isomorphism theorem of section 3 simply yields the isomorphism between $\mathcal{T}$ and the image of the embedding in $\mathcal{F}$. Every element of $\mathcal{T}$ is a fixed point of the embedding followed by the projection, whilst the fixed points of the projection followed by the embedding are those processes which at each interaction and for each event either perform the event or behave chaotically by both performing and refusing it.

Embedding $\mathcal{F}$ in $\mathcal{C}$

In this section we replace the standard embedding of $\mathcal{F}$ in $\mathcal{C}$ by one which forms the basis of a Galois embedding. Combining this with the result of the previous section we will obtain a Galois embedding of $\mathcal{T}$ in $\mathcal{C}$.

The standard embedding

The *divergence-free embedding* $^{\diamond} : \mathcal{R} \to \mathcal{C}$ assigns to failures process $\mathbf{F}$ the divergences process having the same failures as $\mathbf{F}$ but having no divergences:

$$\mathbf{F}^{\diamond} \;\widehat{=}\; (\mathbf{F}, \{\,\}).$$

Now $^\diamond$ is monotone and continuous, but no process is mapped by it to **chaos**. Hence (by the junctivity theorem of section 3) it cannot be the embedding in a Galois connection.

The divergent embedding

Were we to apply the reasoning from earlier in the section, we would expect to embed $\mathcal{F}$ in $\mathcal{C}$ by mapping $\mathbf{F} : \mathcal{F}$ to a process in $\mathcal{C}$ which is as unrefined as possible; it certainly cannot in general have empty divergences.

A natural first attempt is to map $\mathbf{F}$ to $(\mathbf{F}, \mathbf{seq.A})$, which has all possible divergences. Unfortunately that fails the (first and third) healthiness conditions for $\mathcal{C}$. But it *does* satisfy them when $\mathbf{F}$ behaves like **havoc**, which suggests that instead $\mathbf{F}$ be mapped to a process which diverges only when $\mathbf{F}$ behaves like **havoc**; for then the result does lie in $\mathcal{C}$. That leads us to the following definition.

The *divergent embedding* $^\oplus : \mathcal{F} \to \mathcal{C}$ ensures that $\mathbf{F}^\oplus$ diverges at $\mathbf{t}$ iff $\mathbf{F}\ \mathbf{after}\ \mathbf{t} = \mathbf{havoc}$:

$$\mathbf{F}^\oplus \mathrel{\widehat{=}} (\mathbf{F}, \{\mathbf{t} : \mathbf{seq.A} \mid \forall \mathbf{s} : \mathbf{seq.A} \bullet \forall \mathbf{E} \subseteq \mathbf{A} \bullet \mathbf{st}\,\mathbf{F}\,\mathbf{E}\}) .$$

Calculating the adjoint

The adjoint of $^\oplus$ is readily calculated by either approach (b) or (c) to be $^\ominus$ where

$$(\mathbf{G}, \mathbf{D})^\ominus \mathrel{\widehat{=}} \mathbf{G} .$$

A Galois connection

Theorem(Divergent embedding). The divergent embedding and its adjoint $^\ominus$ form a Galois embedding:

$$\mathbf{ge}(^\oplus, ^\ominus ; \mathcal{F}, \mathcal{C}) .$$

Furthermore $^\ominus$ is $\sqcup$-junctive.

Embedding $\mathcal{T}$ in $\mathcal{C}$

Combining the results of the previous sections we obtain a Galois embedding from $\mathcal{T}$ to $\mathcal{C}$.

Theorem(CSP embedding). The liberal embedding followed by the divergent embedding and their inverses composed in the reverse order, form a Galois embedding

$$\mathbf{ge}(^\oplus \circ ^\frown, ^- \circ ^\ominus ; \mathcal{T}, \mathcal{C}) .$$

Furthermore the composition of the projections is $\sqcup$-junctive.

9 Probabilistic semantics

Introduction

We have considered two standard programming paradigms (the imperative language $\mathcal{L}$ and the communicating process language CSP) and studied several semantic models for each. In this chapter we turn to a more novel topic, that of probabilistic imperative programs, and show how the same techniques can be used to provide a semantics for it.

The language $\mathcal{L}_\oplus$ consists of $\mathcal{L}$ extended with a combinator $_\mathbf{p}\oplus$, for each $\mathbf{p} : [0, 1]$, which chooses its left-hand argument with probability $\mathbf{p}$ and its right-hand argument with probability $1 - \mathbf{p}$. Such a language is sufficient to express the probabilistic algorithms championed by Rabin and others (for a survey of which, see R. Motwani and P. Raghavan, *Randomized Algorithms*, CUP, 1995).

For language $\mathcal{L}_\oplus$ we consider: both a relational semantics and an extended-predicate-transformer semantics; a Galois connection between them; healthiness conditions; and some programming techniques for probabilistic algorithms (namely for reasoning about loops). Subtleties of the theory are due to the interplay between probability and nondeterminism.

Programming-language syntax

Our probabilistic programming language $\mathcal{L}_\oplus$ is the language $\mathcal{L}$ extended with probabilistic choice $_\mathbf{p}\oplus$, for $\mathbf{p} : [0, 1]$.

Let $\mathbf{P}$ range over programs, $\mathbf{b}$ over Boolean expressions, and $\mathbf{p}$ over real number expressions between 0 and 1 inclusive; assume that $\mathbf{x}$ stands for a list of distinct variables, and $\mathbf{E}$ for a list of expressions; and let the program *scheme* $\mathcal{C}$ be a program in which the program *name* $\mathbf{Z}$ can appear.

$$\begin{aligned} \mathbf{P} \mathrel{\widehat{=}}\ & \mathbf{abort} \mid \mathbf{Skip} \mid \mathbf{x} := \mathbf{E} \mid \mathbf{P} \fatsemi \mathbf{P} \mid \\ & \mathbf{P} \lhd \mathbf{b} \rhd \mathbf{P} \mid \mathbf{P}\ {}_\mathbf{p}\oplus\ \mathbf{P} \mid \mathbf{P} \sqcap \mathbf{P} \mid \\ & (\mu\, \mathbf{Z} \bullet \mathcal{C}) \end{aligned}$$

Relational semantics

The result of a *deterministic* probabilistic program is a probability distribution over its state space. For a *nondeterministic* probabilistic program the result is a set of distributions, one for each resolution of the nondeterminism. A probabilistic program is therefore modelled relationally by a function from its initial state to a set of distributions over final state.

Distributions

We formalise that by introducing the following notation for distributions. As usual, $\mathbf{X}$ denotes the underlying state space. $\Pi.\mathbf{X}$ denotes the set of all subsets of $\mathbf{X}$.

A (discrete probability) *distribution* means a function $\mathbf{F} : \Pi.\mathbf{X} \to [0, 1]$ satisfying, for all $\mathbf{U}, \mathbf{V} \in \Pi.\mathbf{X}$,

$$\begin{aligned} & \mathbf{U} \subseteq \mathbf{V} \Rightarrow \mathbf{F.U} \leq \mathbf{F.V} \\ & \mathbf{F.}(\mathbf{U} \cup \mathbf{V}) \geq \mathbf{F.U} + \mathbf{F.V} - \mathbf{F.}(\mathbf{U} \cap \mathbf{V}) \\ & \mathbf{F.}\{\,\} = 0 . \end{aligned}$$

Note that distributions do not necessarily satisfy $\mathbf{F.X} = 1$. Often we shall write $\mathbf{F.x}$ for $\mathbf{F.}\{\mathbf{x}\}$.

We write $(\widetilde{\mathbf{X}}, \sqsubseteq)$ for the space of distributions over $\mathbf{X}$,

with order defined

$$\mathbf{F} \sqsubseteq \mathbf{G} \;\widehat{=}\; \forall \mathbf{x} : \mathbf{X} \bullet \mathbf{F}.\mathbf{x} \leq \mathbf{G}.\mathbf{x} .$$

Note that $\mathbf{X}$ is embedded in $\widetilde{\mathbf{X}}$ via the point-mass function: for $\mathbf{x} : \mathbf{X}$ its point mass $\widetilde{\mathbf{x}} : \widetilde{\mathbf{X}}$ assigns 1 to any set containing $\mathbf{x}$ and 0 to any other set

$$\widetilde{\mathbf{x}}.\mathbf{U} \;=\; (\mathbf{x} \in \mathbf{U}) .$$

The relational semantic space for $\mathcal{L}_\oplus$ is thus a subset of $\mathbf{X} \to \Pi.\widetilde{\mathbf{X}}$. That leads us to consider the partially-ordered space $(\mathcal{H}, \sqsubseteq)$, where

$$\mathcal{H} \;\widehat{=}\; \mathbf{X} \to \Pi.\widetilde{\mathbf{X}} ,$$

and $\sqsubseteq$ is containment; for $\mathbf{P}, \mathbf{P}' : \mathbf{X} \to \Pi.\widetilde{\mathbf{X}}$, it is defined

$$\mathbf{P} \sqsubseteq \mathbf{P}' \;\widehat{=}\; \forall \mathbf{x} \in \mathbf{X} \bullet \mathbf{P}.\mathbf{x} \supseteq \mathbf{P}'.\mathbf{x} .$$

Relational semantics

The semantics of language $\mathcal{L}_\oplus$ is given by the function

$$[\,] : \mathcal{L}_\oplus \to \mathcal{H}$$

defined as follows.

$$[\mathbf{Skip}].\mathbf{x} \;\widehat{=}\; \{\widetilde{\mathbf{x}}\}$$

$$[\mathbf{abort}].\mathbf{x} \;\widehat{=}\; \widetilde{\mathbf{X}}$$

$$[\mathbf{x} := \mathbf{E}].\mathbf{s} \;\widehat{=}\; \mathbf{s}[\widetilde{\mathbf{x} := \mathbf{E}}]$$

$$[\mathbf{P} \sqcap \mathbf{P}'].\mathbf{x} \;\widehat{=}\; \cup\{(\mathbf{P}\,{}_{\mathbf{p}}\!\oplus \mathbf{P}').\mathbf{x} \mid \mathbf{p} \in [0,1]\}$$

$$[\mathbf{P} \lhd \mathbf{b} \rhd \mathbf{P}'].\mathbf{x} \;\widehat{=}\; \mathbf{P}.\mathbf{x} \lhd \mathbf{b}.\mathbf{x} \rhd \mathbf{P}'.\mathbf{x}$$

$$[\mathbf{P}\,{}_{\mathbf{p}}\!\oplus \mathbf{P}'].\mathbf{x} \;\widehat{=}\; \{\mathbf{p}\mathbf{E} + (1-\mathbf{p})\mathbf{F} \mid \begin{pmatrix} \mathbf{E} \in \mathbf{P}.\mathbf{x} \\ \mathbf{F} \in \mathbf{P}'.\mathbf{x} \end{pmatrix}\}$$

$$[\mathbf{P} \fatsemi \mathbf{P}'].\mathbf{x} \;\widehat{=}\; \textstyle\sum\{\mathbf{F}.\mathbf{x}' \times \mathbf{G}_{\mathbf{x}'} \mid \begin{pmatrix} \mathbf{x}' \in \mathbf{X} \\ \mathbf{F} \in \mathbf{P}.\mathbf{x} \\ \mathbf{G}_{\mathbf{x}'} \in \mathbf{P}'.\mathbf{x}' \end{pmatrix}\}$$

$$[\mu \mathbf{Z} \bullet \mathcal{C}] \;\widehat{=}\; \text{least fixed point of } \mathbf{cntx}: \mathcal{H} \to \mathcal{H}, \text{ defined } \mathbf{wp}.\mathcal{C} = \mathbf{cntx}.(\mathbf{wp}.\mathbf{Z}) .$$

Healthiness

The set of distributions of a probabilistic program satisfy three conditions: they are convex-closed, Cauchy-closed and up-closed.

1. A set $\mathcal{F}$ of distributions is **convex** means for every $\mathbf{E}, \mathbf{F} \in \mathcal{F}$ and $\mathbf{p} \in [0,1]$, $\mathbf{pE} + (1-\mathbf{p})\mathbf{F} \in \mathcal{F}$ also.
2. A set $\mathcal{F}$ of distributions is *Cauchy-closed* means it is a closed subset of $\mathbf{R}^{\#\mathbf{X}}$ in the usual Euclidean sense.
3. A set $\mathcal{F}$ of distributions is *up-closed* means it is closed under refinement of its elements: for all $\mathbf{E}, \mathbf{F} \in \widetilde{\mathbf{X}}$,

$$\begin{pmatrix} \mathbf{E} \in \mathcal{F} \\ \mathbf{E} \sqsubseteq \mathbf{F} \end{pmatrix} \Rightarrow \mathbf{F} \in \mathcal{F} .$$

Up-closure means that program refinement corresponds to reverse inclusion of resulting-distribution sets. Convex-closure means that a nondeterministic choice between two programs is refined by any probabilistic choice. Cauchy-closure makes the space of programs into a complete partial order.

Predicate-transformer semantics

We now consider an alternative semantics, based on predicate transformers, for language $\mathcal{L}_\oplus$. Again, we require a little notation first.

Probabilistic predicates

A *probabilistic predicate* is a non-negative-real-valued predicate on state space; the set of all such is defined $\mathcal{Q} \;\widehat{=}\; \mathbf{X} \to \mathbf{R}^{\geq}$, where $\mathbf{R}^{\geq}$ denotes the set of non-negative reals.

Standard predicates are embedded in $\mathcal{Q}$ by identifying *true* with 1 and *false* with 0. For standard predicate $\mathbf{q}$ we write $[\mathbf{q}]$ for its embedding.

The logical operations, and their arithmetic equivalents, on probabilistic predicates are summarised as follows.

logic	*arithmetic*
$[\neg \mathbf{q}]$	$1 - [\mathbf{q}]$
$[\mathbf{q} \vee \mathbf{q}']$	$[\mathbf{q}] \sqcup [\mathbf{q}']$, or $[\mathbf{q}] + [\mathbf{q}']$ if $\mathbf{q}, \mathbf{q}'$ disjoint
$[\mathbf{q} \wedge \mathbf{q}']$	$[\mathbf{q}] \sqcap [\mathbf{q}']$, or $[\mathbf{q}] \times [\mathbf{q}']$
$\mathbf{q} \Rrightarrow \mathbf{q}'$	$[\mathbf{q}] \leq [\mathbf{q}']$ on $\mathbf{X}$
$\mathbf{q} \equiv \mathbf{q}'$	$[\mathbf{q}] = [\mathbf{q}']$ on $\mathbf{X}$
$\mathbf{q} \Lleftarrow \mathbf{q}'$	$[\mathbf{q}] \geq [\mathbf{q}']$ on $\mathbf{X}$

The *entailment* relation $\Rrightarrow$ between standard predicates becomes pointwise $\leq$ over their probabilistic counterparts.

For distribution $\mathbf{F} : \widetilde{\mathbf{X}}$ and probabilistic predicate $\mathbf{q} : \mathcal{Q}$, the integral over $\mathbf{X}$ of $\mathbf{q}$ against $\mathbf{F}$ with the discrete, or counting, measure is denoted

$$\textstyle\int_{\mathbf{F}} \mathbf{q} \;\widehat{=}\; \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{q}.\mathbf{x} \times \mathbf{F}.\mathbf{x} .$$

Semantics

Let $(\mathcal{J}, \sqsubseteq)$ denote the space of probabilistic-predicate transformers ordered

$$\mathbf{j} \sqsubseteq \mathbf{k} \;\widehat{=}\; \forall \mathbf{q} : \mathcal{Q} \bullet \mathbf{j}.\mathbf{q} \Rrightarrow \mathbf{k}.\mathbf{q} .$$

We can now interpret a program in $\mathcal{L}_\oplus$ to be a transformer from probabilistic postconditions to probabilistic precon-

ditions as follows.

$$\mathbf{wp.abort.q} \;\widehat{=}\; 0$$

$$\mathbf{wp.Skip.q} \;\widehat{=}\; \mathbf{q}$$

$$\mathbf{wp.(x := E).q} \;\widehat{=}\; \mathbf{q[x:=E]}$$

$$\mathbf{wp.(P \fatsemi P').q} \;\widehat{=}\; \mathbf{wp.P.(wp.P'.q)}$$

$$\mathbf{wp.(P \lhd b \rhd P').q} \;\widehat{=}\; \mathbf{[b] \times wp.P.q + [\neg b] \times wp.P'.q}$$

$$\mathbf{wp.(P \sqcap P').q} \;\widehat{=}\; \mathbf{wp.P.q \sqcap wp.P'.q}$$

$$\mathbf{wp.(P}\ {}_{\mathbf{p}}\!\oplus \mathbf{P').q} \;\widehat{=}\; \mathbf{p \times wp.P.q + (1-p) \times wp.P'.q}$$

$$\mathbf{wp.}(\mu \mathbf{Z} \bullet \mathcal{C}) \;\widehat{=}\; \text{least fixed point of } \mathbf{cntx}: \mathcal{J} \to \mathcal{J}, \text{ defined } \mathbf{wp.}\mathcal{C} = \mathbf{cntx.(wp.Z)}\,.$$

There, where appropriate, $\sqcap$ means pointwise minimum between functions.

Galois connection

For program $\mathbf{P}$ and probabilistic postcondition $\mathbf{q}$,

$$\mathbf{wp.P.q.x} = \min\{\textstyle\int_{\mathbf{F}} \mathbf{q} \mid \mathbf{F} \in \mathbf{P.x}\}\,.$$

We shall define two functions $\mathbf{wp} : \mathcal{H} \to \mathcal{J}$ and $\mathbf{rp} : \mathcal{J} \to \mathcal{H}$ between the lattices $(\mathcal{H}, \supseteq)$ and $(\mathcal{J}, \sqsubseteq)$.

The embedding **wp**

For probabilistic relation $\mathbf{h} : \mathcal{H}$, probabilistic predicate $\mathbf{q} : \mathcal{Q}$, and state $\mathbf{x} : \mathbf{X}$,

$$\mathbf{wp.h.q.x} \;\widehat{=}\; \sqcap\{\textstyle\int_{\mathbf{F}} \mathbf{q} \mid \mathbf{F} \in \mathbf{h.x}\}\,.$$

The projection **rp**

For probabilistic predicate transformer $\mathbf{j} : \mathcal{J}$ and state $\mathbf{x} : \mathbf{X}$,

$$\mathbf{rp.j.x} \;\widehat{=}\; \{\mathbf{F} : \widetilde{\mathbf{X}} \mid \forall \mathbf{q} : \mathcal{Q} \bullet \mathbf{j.q.x} \le \textstyle\int_{\mathbf{F}} \mathbf{q}\}\,.$$

Note that **rp** is not always defined: recall that relations in $\mathcal{H}$ are non-empty. However the application of **rp** to a **wp** image of a relation is always defined.

Partial weak inverse

When **rp** is defined then **wp** is its weak inverse. For $\mathbf{h} : \mathcal{H}$ and $\mathbf{j} : \mathcal{J}$,

$$\mathbf{rp.wp.h} = \mathbf{h} \qquad \mathbf{j} \sqsubseteq \mathbf{wp.rp.j}\,.$$

Healthiness conditions

The **wp** images of relations within $\mathcal{J}$ no longer satisfy distribution of $\sqcap$. Instead they are characterised by continuity and sub-linearity.

Transformer $\mathbf{j} : \mathcal{J}$ is *sub-linear* if for all $\mathbf{q}, \mathbf{q}' : \mathcal{Q}$ and all $\mathbf{a}, \mathbf{b}, \mathbf{c} : \mathbf{R}^{\geq}$

$$\mathbf{aj.q + bj.q' - c} \Rrightarrow \mathbf{j.(aq + bq' - c)}\,.$$

Sub-linearity implies monotonicity, feasibility and scaling. In fact when the state space is finite it implies continuity as well.

Operator & is defined on probabilistic predicates:

$$\mathbf{(A \,\&\, B).s} \;\widehat{=}\; \mathbf{(A.s + B.s - 1) \sqcup 0}\,.$$

Sub-linearity implies &-distribution and when specialised to standard programs &-distribution is equivalent to positive conjunctivity.

Loop rules

One of the strengths of the standard predicate-trans- former calculus is its facility for proof—in particular variant/invariant -based rules for correctness of iterations. This section deals with the probabilistic counterparts to the standard loop rules where, here too, the predicate-transformer calculus pays off.

We shall use laws to prove termination and correctness about loops of the form

$$\mathbf{loop} \;\widehat{=}\; \mathbf{do\, G \to body\, od}$$

where the loop guard $\mathbf{G}$ is a standard (non-probabilis- tic) predicate and **body** is a relational (thus sub-linear) predicate transformer. A program, $\mathbf{h}$, terminates from an initial state $\mathbf{x}$ if $\mathbf{wp.h.[true].x} = 1$.

Let $\mathbf{T} = \mathbf{wp.loop.[true]}$ be the termination condition for **loop**. We will assume (although it is not necessary to do so) that **body** always terminates. (The rules are actually derived from a probabilistic version of **wlp** which agrees with **wp** if termination is assumed.)

Standard rules for reasoning about probabilistic loops separate correctness from termination. An invar iant-based argument is usually sufficient to establish partial correctness, whereas a variant argument is used to show termination.

A predicate, $\mathbf{I}$, is an invariant for the loop if

$$\mathbf{(G \sqcap I)} \Rrightarrow \mathbf{wp.body.I}\,.$$

Combining partial and total correctness

Let $\mathbf{I}$ be an invariant for **loop**. Then

$$\textbf{Rule 1}: \quad \mathbf{I \,\&\, T} \Rrightarrow \mathbf{wp.loop.}(\widetilde{\mathbf{G}} \sqcap \mathbf{I})\,.$$

Notice that reduces to the standard rule when specialised to standard programs.

In many cases & gives too pessimistic a bound. We can do better, though. If $\mathbf{I} \Rrightarrow \mathbf{T}$ then

$$\textbf{Rule 2}: \quad \mathbf{I} \Rrightarrow \mathbf{wp.loop.}(\widetilde{\mathbf{G}} \sqcap \mathbf{I})\,.$$

Variant-based arguments

The 0-1 law

The probabilistic variant arguments (over a finite state space) we use are based on a so-called 0-1 law.

Informally, the idea is as follows. Let $\mathbf{Y}$ be a fixed subset of state space $\mathbf{X}$ and suppose that from every state in $\mathbf{Y}$ there is a non-zero probability of the loop escaping from $\mathbf{Y}$. Then in fact we can deduce that the probability of eventual escape from $\mathbf{Y}$ is 1.

The formal rule is

$$\textbf{0-1 Law}: \quad (\exists\, \mathbf{p} > 0 \bullet \mathbf{pI} \Rrightarrow \mathbf{T}) \Rightarrow \mathbf{I} \Rrightarrow \mathbf{T}\,.$$

From that we can derive Rule 3.

Rule 3: Let $\mathbf{v}$ be an integer-valued expression on $\mathbf{X}$ defined at least over some subset $\mathbf{I}$ of the state space $\mathbf{X}$. Suppose further that for iteration **loop**

1. there are fixed integer constants $\mathbf{L}$ (low) and $\mathbf{H}$ (high) such that
$$\mathbf{G} \sqcap \mathbf{I} \Rrightarrow [\mathbf{L} \leq \mathbf{V} < \mathbf{H}]\,,$$
2. the subset $\mathbf{I}$, as a (standard) predicate, is a **wp**-invariant for **loop**
3. for some fixed probability $\mathbf{p} \neq 0$ and for all $\mathbf{M} : \mathbf{N}$, $\mathbf{p}(\mathbf{G} \sqcap \mathbf{I} \sqcap [\mathbf{V} = \mathbf{M}]) \Rrightarrow \mathbf{wp.body}.[\mathbf{V} < \mathbf{M}]$.

Then termination is certain from any state in which $\mathbf{I}$ holds: we have $\mathbf{I} \Rrightarrow \mathbf{T}$.

Finitary completeness of variants

Rule 3 is sound and complete for termination over a finite state space.

Self-stabilisation

We now give an example of a probabilistic algorithm. It is based on an algorithm due to Ted Hermann. The problem is how to achieve, efficiently and symmetrically, a leader from a ring of processes.

Consider $\mathbf{M}$ identical processors connected clockwise in a ring. A single processor—a leader—is chosen from them as follows.

Initially each processor is given a single token: the leader will be the first processor to obtain all $\mathbf{M}$ tokens. Fix some probability $\mathbf{p} : [0, 1]$. On each step all processors synchronously perform the following actions:

1. Make a local probabilistic decision either to **pass** (with probability $\mathbf{p}$) or to **keep** (with probability $1 - \mathbf{p}$) all its tokens.
2. If **pass**, then send **all** its tokens to its clockwise successor; if **keep**, skip.
3. Receive any tokens passed from its clockwise predecessor.

Termination of this algorithm follows directly from Rule 3 taking as variant the largest arc of the ring containing all tokens.

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarverenging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof Lucas Introna
Department of Informatics
University of Pretoria
Hatfield 0083
lintrona@econ.up.ac.za

Production Editor
Dr Riël Smit
Mosaic Software (Pty) Ltd
P.O.Box 23906
Claremont 7735
gds@mosaic.co.za

World-Wide Web: http://www.mosaic.co.za/sacj/

Subscriptions

	Annual	Single copy
Southern Africa:	R50,00	R25,00
Elsewhere:	$30,00	$15,00

An additional $15 per year is charged for airmail outside Southern Africa

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276

South African Computer Journal

Number 19, February 1997
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 19, Februarie 1997
ISSN 1015-7999

Contents