South African Computer Journal
Number 13
April 1995

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 13
April 1995

# Computer Science and Information Systems

# 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*

**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

# Introduction

## WOFACS '94: The Second Workshop on Formal Aspects of Computer Science

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

Following the success of WOFACS '92 (for the *Proceedings* of which see SACJ 9 1993) another such event was held at the University of Cape Town, from 27 June to 8 July 1994.

These Workshops on Formal Aspects of Computer Science serve several purposes. First, they help to strengthen a culture of studying formal aspects and developing formal methods in Computer Science. Second, in doing so they provide an impetus towards collaboration and interdisciplinarity – in this case bringing together Logic, Mathematics and Computer Science. Third, they provide a vehicle for the inter-institutional training of postgraduate students. And fourth, they contribute to international collaboration by bringing a number of eminent scientists to South Africa.

WOFACS '94 was co-hosted by the Departments of Mathematics and Computer Science at the University of Cape Town, and was organised by the Laboratory for Formal Aspects and Complexity in Computer Science. It offered 2-week lecture courses as follows:

- Dr Falko Bause (Dortmund University, Germany), *Petri Nets*
- Prof Ed Brinksma (Technological University of Twente, Netherlands), *Formal Design of Distributed Systems*
- Prof Robert Goldblatt (Victoria University of Wellington), *Modal Logics of Programs*
- Prof Austin Melton (Michigan Technological University), *Denotational Semantics*
- Dr Carroll Morgan (Oxford University), *The Refinement Calculus*

About 80 participants attended WOFACS '94; of these roughly half were academic staff and half were graduate students from a number of Southern African Universities. Participation from neighbouring countries was very encouraging. The lecture courses were again made available to all attending students as credit-bearing courses at their home Universities (by prior arrangement with the Departments concerned). About 30 students were eventually evaluated and had their WOFACS results incorporated into their respective Honours programmes. Apart from travel and accommodation costs and a small administrative charge for registration WOFACS '94 was a free service to the community – no lecture fees of any kind were involved.

This volume of SACJ contains the *Proceedings* of WOFACS '94: papers relating to their lecture courses by Brinksma, Goldblatt, Melton and Morgan (and co-authors). Unfortunately the Bause contribution to WOFACS '94 was committed to a publisher before the event, and hence could not be included here.

For the success of WOFACS '94 I would like to express my grateful thanks to:

- our invited speakers, for their hard work and the quality of their presentations;
- the Foundation for Research Development, the participants of the Programme for Research Manpower in Computer Science, the UCT Research Committee and the Department of National Education, for direct and indirect sponsorship;
- my co-host, Professor Pieter Kritzinger;
- the indefatigable FACCS-Lab staff, particularly Diana Dixon, and
- all participants.

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

# Modal Logics for Programs

Robert Goldblatt
*Department of Mathematics, Victoria University of Wellington, P.O. Box 600, Wellington, New Zealand*
*Rob.Goldblatt@vuw.ac.nz*

## Abstract

*These lectures provide an introduction to modal logic and its use in formalising reasoning about the behaviour of computational processes. They begin with a general introduction to the syntax, semantics, and proof-theory of modal languages, and their historical origins. There then follows an exposition of some particular formalisms that are particularly relevant to computer science: dynamic logic, the temporal logic of concurrency, the Hennessy-Milner logic of processes, and the powerful Mu-Calculus that encompasses all of these systems. The third part explains technical methods (canonical models, filtrations) that have been developed to analyse particular logics, and finally these methods are applied to give a proof of the completeness theorem for linear temporal logic.*
**Keywords:** *Modal Logic, Dynamic Logic, Linear Temporal Logic, Branching time logics, Hennessy–Milner modal Logic, Modal Mu Calculus*
**Computing Review Categories:** *F.3, F.4*

## 1 Modal Languages and Logics

Modal logic is an ancient subject, devised by philosophers and modelled by mathematicians, with some of its most significant development and application occurring in the last two decades at the hands of computer scientists.

These lectures aim to provide

- an introduction to modal logic and its origins;
- a brief tour of some of the ways in which it has been used to formalise reasoning about the behaviour of computational processes;
- an indication of some of the technical methods that are employed in analysing different logics;
- some pointers to the literature for those who wish to pursue these topics in greater depth.

**Modalities:**
linguistic constructions that qualify assertions about the truth of statements, and express various *modes* of truth:

it is necessarily true that $A$.
it is possible/could be/might be that $A$ is true.
probably $A$.
it has always been true that $A$.
it will eventually be true that $A$.
it ought to be that $A$.
it is permissible that $A$.
it is known that $A$.
it is believed that $A$.

it is provable that $A$.
after the program terminates, $A$ will be true.
throughout the computation, $A$.
at the next state, $A$.
along all future paths, $A$.

**A Formal Language**
*"Syntax": the way in which words are put together to form phrases and sentences.*
Formulae (sentences) $A$ are constructed from a given set $\Phi$ of atomic formulae, and the constant "False" formula $\bot$:

| | |
|---|---|
| Atomic formulae: | $p \in \Phi$ |
| Formulae: | $A \in Fma(\Phi)$ |

$$A ::= p \mid \bot \mid A_1 \to A_2 \mid \Box A$$

**Other connectives**

These are introduced by the usual abbreviations.

| | | | |
|---|---|---|---|
| Negation (not): | $\neg A$ | is | $A \to \bot$ |
| "True": | $\top$ | is | $\neg\bot$ |
| Disjunction (or): | $A_1 \vee A_2$ | is | $(\neg A_1) \to A_2$ |
| Conjunction (and): | $A_1 \wedge A_2$ | is | $\neg(A_1 \to \neg A_2)$ |
| Equivalence (iff): | $A_1 \leftrightarrow A_2$ | is | $(A_1 \to A_2) \wedge (A_2 \to A_1)$ |
| "Diamond": | $\Diamond$ | is | $\neg\Box\neg A$ |

**Frames and Models**

*"Semantics": the study of meaning in language.*

A *frame* is a pair $\mathcal{F} = (S, R)$, where $S$ is a non-empty set, and $R$ a binary relation on $S$: in symbols, $R \subseteq S \times S$. Write $sRt$ to mean that the pair $(s,t)$ belongs to $R$.

A *$\Phi$-model* on a frame is a triple $\mathcal{M} = (S, R, V)$, with $V : \Phi \to 2^S$. Here $V$ is a *valuation* assigning to each atomic formula $p \in \Phi$ a subset $V(p)$ of $S$. Informally $V(p)$ is to be thought of as the set of points at which $p$ is "true". Generally we drop the prefix $\Phi$- in discussing models, provided the context is clear.

The relation *"$A$ is true (holds) at point $s$ in model $\mathcal{M}$"*, denoted

$$\mathcal{M} \models_s A,$$

is defined inductively on the formation of $A \in Fma(\Phi)$ as follows.

| | | |
|---|---|---|
| $\mathcal{M} \models_s p$ | iff | $s \in V(p)$ |
| $\mathcal{M} \not\models_s \bot$ | | (i.e. not $\mathcal{M} \models_s \bot$) |
| $\mathcal{M} \models_s (A_1 \to A_2)$ | iff | $\mathcal{M} \models_s A_1$ implies $\mathcal{M} \models_s A_2$ |
| $\mathcal{M} \models_s \Box A$ | iff | for all $t \in S$, $sRt$ implies $\mathcal{M} \models_t A$ |

hence

| | | |
|---|---|---|
| $\mathcal{M} \models_s \neg A$ | iff | not $\mathcal{M} \models_s A$ |
| $\mathcal{M} \models_s (A_1 \wedge A_2)$ | iff | $\mathcal{M} \models_s A_1$ and $\mathcal{M} \models_s A_2$ |
| $\mathcal{M} \models_s (A_1 \vee A_2)$ | iff | $\mathcal{M} \models_s A_1$ or $\mathcal{M} \models_s A_2$ |
| $\mathcal{M} \models_s \Diamond A$ | iff | for some $t \in S$, $sRt$ and $\mathcal{M} \models_t A$. |

**Motivations**

*Leibniz: a necessary truth is one that holds in all "possible worlds".*

- *Necessity.* $S$ may be thought of as a set of such worlds, with $sRt$ when $t$ is a conceivable alternative to $s$, i.e. a world in which all the necessary truths of $s$ are realised. $\Box A$ then means "A is necessarily true", while $\Diamond A$ means "$A$ is possible", i.e. true in some conceivable world.
  Notions of *logical* and *physical* necessity may be distinguished.
- *Temporal Logic.* Here the members of $S$ are taken to be moments of time. If $sRt$ means "$t$ is after (later than) $s$", then $\Box A$ means "*hence-forth* $A$", i.e. "at all future times $A$", while $\Diamond A$ means "*eventually* (at some future time) $A$". Dually, if $sRt$ means that $t$ is before $s$, then $\Box$ means "*hitherto*", and so on.
  Natural time frames $(S, R)$ for temporal logic are given by taking $S$ as one of the number sets $\omega$ (natural numbers), $\mathbb{Z}$ (integers), $\mathbb{Q}$ (rationals), or $\mathbb{R}$ (reals), and $R$ as one of the relations $<, \leq, >, \geq$.
- *Program states.* Regard $S$ as the set of possible *states* of a computation process, with $sRt$ meaning that there is an execution of the program that starts in state $s$ and terminates in state $t$. A non-deterministic program may admit more than one possible "outcome" $t$ when started in $s$. Then $\Box A$ means "every terminating execution of the program brings about $A$", while $\Diamond A$ means that the program *enables* $A$, i.e. "there is some execution that terminates with $A$ true".

**Truth and Validity**

Formula $A$ is *true in model* $\mathcal{M}$, denoted $\mathcal{M} \models A$, if it is true at all points in $\mathcal{M}$, i.e. if

$$\mathcal{M} \models_s A \text{ for all } s \in S.$$

$A$ is *valid in frame* $\mathcal{F} = (S, R)$, denoted $\mathcal{F} \models A$, if

$$\mathcal{M} \models A \text{ for all models } \mathcal{M} = (S, R, V) \text{ based on } \mathcal{F}.$$

If $\mathcal{C}$ is a class of models (respectively, frames), then $A$ is *true* (respectively, *valid*) in $\mathcal{C}$, $\mathcal{C} \models A$, if $A$ is true (respectively, valid) in all members of $\mathcal{C}$.

$A$ is *falsifiable* in model $\mathcal{M}$ if $\mathcal{M} \not\models A$, i.e. $\mathcal{M} \not\models_s A$ for some $s$, and is *satisfiable* in $\mathcal{M}$ if $\mathcal{M} \not\models \neg A$, i.e. $\mathcal{M} \models_s A$ for some $s$.

**Examples**

(1) The schema $\Box A \to A$ is valid in $\mathcal{F}$ iff $R$ is *reflexive*, i.e. $sRs$ for all $s \in S$.

(2) The schema $\Box A \to \Box\Box A$ is valid in $\mathcal{F}$ iff $R$ is *transitive*, i.e. if $sRt$ and $tRu$, then $sRu$.

(3) $\mathcal{F} \models A \to \Box\Diamond A$ iff $R$ is *symmetric*, i.e. $sRt$ implies $tRs$.

(4) $\mathcal{F} \models \Box A \to \Diamond A$ iff $R$ is *total*, i.e. $\forall s \exists t\,(sRt)$; while $\mathcal{F} \models \Diamond A \to \Box A$ iff $R$ is *functional*, i.e. $sRt$ and $sRu$ implies $t = u$.
Hence $\mathcal{F} \models \Diamond A \leftrightarrow \Box A$ iff $R$ is a total function on $S$.

(5) $\Box\Box A \to \Box A$ is valid on $(\mathbb{Q}, <)$ but not on $(\mathbb{Z}, <)$.

(6) $(\mathbb{Q}, <) \models A$ iff $(\mathbb{R}, <) \models A$, so there is no modal formula in the present language whose validity distinguishes $(\mathbb{Q}, <)$ and $(\mathbb{R}, <)$.

**Denotations**

A model $\mathcal{M} = (S, R, V)$ associates with each formula $A$ the "truth set"

$$V(A) = \{s \in S : \mathcal{M} \models_s A\},$$

which will be called the *denotation* of $A$ in $\mathcal{M}$. In this way the valuation $V : \Phi \to 2^S$ is extended to a function $V : Fma(\Phi) \to 2^S$ which satisfies such properties as

$$\begin{aligned}
V(\neg A) &= S - V(A)\\
V(A_1 \wedge A_2) &= V(A_1) \cap V(A_2)\\
V(A_1 \vee A_2) &= V(A_1) \cup V(A_2)\\
V(\Box A) &= \{s : \text{for all } t \in S,\ sRt \text{ implies } t \in V(A)\}\\
V(\Diamond A) &= \{s : \text{for some } t \in S,\ sRt \text{ and } t \in V(A)\}.
\end{aligned}$$

These equation could be used inductively to directly define the extension of a given valuation $V : \Phi \to 2^S$, and this offers an alternative approach to semantics that will be used in Section 5.

Observe that in terms of denotations, $A$ is valid in frame $(S, R)$ iff $V(A) = S$ for all valuations $V : \Phi \to 2^S$.

**Multimodal Syntax**

The theory presented so far adapts readily to languages with more than one modal connective. Given a set $\Phi$ of atomic formulae $p$, and a new collection of symbols $\{[i] : i \in I\}$, a set $Fma_I(\Phi)$ of formulae $A$ is generated by the definition

$$A ::= p \mid \bot \mid A_1 \to A_2 \mid [i]A,$$

so that there are now formulae $[i]A$ for each $A \in Fma_I(\Phi)$ and each $i \in I$. The connective $[i]$ is to be treated in the way $\Box$ was treated above. The dual connective $<i>$ is defined as $\neg[i]\neg$, and corresponds to $\Diamond$.

**Multimodal Semantics**

A *frame* for this new language is a structure

$$\mathcal{F} = (S, \{R_i : i \in I\}),$$

comprising a set $S$ with a collection of binary relations $R_i \subseteq S \times S$, one for each $i \in I$. A model $\mathcal{M} = (\mathcal{F}, V)$ on $\mathcal{F}$ is given by a function $V : \Phi \to 2^S$, just as before. The definition of the relation $\mathcal{M} \models_s A$ has the one new clause

$$\mathcal{M} \models_s [i]A \quad \text{iff} \quad \text{for all } t \in S,\ sR_it \text{ implies } \mathcal{M} \models_t A.$$

Hence

$$\mathcal{M} \models_s <i> A \quad \text{iff} \quad \text{for some } t \in S,\ sR_it \text{ and } \mathcal{M} \models_t A.$$

The definitions of truth in a model ($\mathcal{M} \models A$), and validity in a frame ($\mathcal{F} \models A$), are unchanged.

**Temporal Logic**

This is based on a propositional language with two modal connectives, [F] and [P], meaning, respectively, *henceforth* (at all future times), and *hitherto* (at all past times). Their dual connectives $<\mathrm{F}>$ and $<\mathrm{P}>$ mean *eventually* (at some future time) and *previously* (at some past time).

A frame for this language has the form $(S, R_{\mathrm{F}}, R_{\mathrm{P}})$, with the modelling

$$\mathcal{M} \models_s [\mathrm{F}]A \quad \text{iff} \quad sR_{\mathrm{F}}t \text{ implies } \mathcal{M} \models_t A,$$

$$\mathcal{M} \models_s [\mathrm{P}]A \quad \text{iff} \quad sR_{\mathrm{P}}t \text{ implies } \mathcal{M} \models_t A.$$

$sR_{\mathrm{F}}t$ is read as "$t$ is in the future of $s$" and $sR_{\mathrm{P}}t$ as "$t$ is in the past of $s$". But the intended interpretation is that [F] and [P] express properties of the same time-ordering, so that $t$ should be in the past of $s$ precisely when $s$ is in the future of $t$. This requires that

$$sR_{\mathrm{P}}t \quad \text{iff} \quad tR_{\mathrm{F}}s$$

(or, equivalently, that the relations $R_{\mathrm{P}}$ and $R_{\mathrm{F}}$ are each the inverse of the other), so that

$$\mathcal{M} \models_s [\mathrm{P}]A \quad \text{iff} \quad tR_{\mathrm{F}}s \text{ implies } \mathcal{M} \models_t A.$$

In this context, the formula $\Box A$ is taken as an abbreviation for

$$[\mathrm{P}]A \wedge A \wedge [\mathrm{F}]A,$$

which may be read "*always* $A$", i.e. at all times past present and future.

**Exercises**

(1) $R_{\mathrm{P}}$ and $R_{\mathrm{F}}$ are mutually inverse iff the schemata

$$A \to [\mathrm{P}] <\mathrm{F}> A$$

$$A \to [\mathrm{F}] <\mathrm{P}> A$$

are valid in $\mathcal{F}$.

(2) The schema

$$\Box([\mathrm{P}]A \to <\mathrm{F}> [\mathrm{P}]A) \to ([\mathrm{P}]A \to [\mathrm{F}]A)$$

is valid in $(\mathbb{R}, <, >)$, but not in $(\mathbb{Q}, <, >)$.

**Truth-Valuations and Tautologies**

Given a multimodal $\Phi$-model $\mathcal{M}$, and a fixed $s \in S$, define

$$V_s(p) = \begin{cases} \mathit{true} & \text{if } s \in V(p); \\ \mathit{false} & \text{otherwise.} \end{cases}$$

The function $V_s : \Phi \to \{\mathit{true}, \ \mathit{false}\}$ is a ***truth-valuation*** of the atomic formulae, a notion familiar from propositional logic. Using the standard truth-tables for propositional connectives, $V_s$ is extended to assign a truth-value to any formula not containing any modal symbol $[i]$.

A formula $A$ is *quasi-atomic* if either it is atomic ($A \in \Phi$), or else it begins with a $[i]$, i.e. $A = [i]B$ for some $B$. If $\Phi^q$ is the set of all quasi-atomic formulae, then any formula $A$ is constructible from members of $\Phi^q \cup \{\bot\}$ using the connective $\to$. Hence by using the truth-table for $\to$, any truth-valuation

$$V : \Phi^q \to \{\mathit{true}, \ \mathit{false}\}$$

of the quasi-atomic formulae extends uniquely to a truth-valuation

$$V : \mathit{Fma}(\Phi) \to \{\mathit{true}, \ \mathit{false}\}$$

of all formulae.

A formula $A$ is a *tautology* if $V(A) = \mathit{true}$ for every truth-valuation $V$ of its quasi-atomic subformulae. Any tautology is a substitution instance of a tautology of propositional logic (i.e. a $[i]$-free tautology). Tautologies are always valid.

## Tautological Consequence

A formula $A$ is a *tautological consequence* of formulae $A_1, \dots, A_n$ if $A$ is assigned *true* by every valuation that assigns *true* to all of $A_1, \dots, A_n$. In particular, a tautological consequence of the empty set of formulae is the same thing as a tautology.

## Logics

*Problem: Characterise* $\{A : \mathcal{M} \models A\}$ *in a syntactic, or proof-theoretic way.*

Given a language based on a set $\Phi$ of atomic formulae, a *logic* is defined to be any set $\Lambda \subseteq Fma_I(\Phi)$ such that

- $\Lambda$ includes all tautologies, and
- $\Lambda$ is closed under the rule of *Detachment*, i.e.,
  if $A$, $A \to B \in \Lambda$ then $B \in \Lambda$.

Such a set is closed under tautological consequence, i.e. if $A_1, \dots, A_n \in \Lambda$, then any tautological consequence of $A_1, \dots, A_n$ belongs to $\Lambda$.

The members of a logic are called its *theorems*, and the notation $\vdash_\Lambda A$ means that $A$ is a $\Lambda$-theorem, i.e.,

$$\vdash_\Lambda A \quad \text{iff} \quad A \in \Lambda.$$

## Examples of Logics

- $PL = \{A \in Fma_I(\Phi) : A \text{ is a tautology}\}$.
- For any class $\mathcal{C}$ of models, or of frames (including the cases $\mathcal{C} = \{\mathcal{M}\}$ and $\mathcal{C} = \{\mathcal{F}\}$),

$$\Lambda_{\mathcal{C}} = \{A : \mathcal{C} \models A\}$$

  is a logic.
- $Fma_I(\Phi)$ itself is a logic, the only one to contain $\perp$. For any logic $\Lambda$,

$$PL \subseteq \Lambda \subseteq Fma_I(\Phi).$$

- If $\{\Lambda_x : x \in X\}$ is a collection of logics, then their intersection

$$\bigcap \{\Lambda_x : x \in X\}$$

  is a logic. Thus for any $\Gamma \subseteq Fma_I(\Phi)$ there is a *smallest* logic $\Lambda_\Gamma$ containing $\Gamma$, namely

$$\Lambda_\Gamma = \bigcap \{\Lambda : \Lambda \text{ is a logic and } \Gamma \subseteq \Lambda\}.$$

Instead of defining a logic $\Lambda$ to include all tautologies, it would suffice to include a set of schemata from which all tautologies can be derived by Detachment, e.g.

$$\begin{aligned}
&A \to (B \to A) \\
&A \to (B \to D) \to ((A \to B) \to (A \to D)) \\
&\neg\neg A \to A.
\end{aligned}$$

$PL$ is the smallest set of formulae that includes these three schemata and is closed under Detachment.

## Soundness and Completeness

Let $\mathcal{C}$ be a class of frames, or of models. Then logic $\Lambda$ is *sound with respect to* $\mathcal{C}$ if for all formulae $A$,

$$\vdash_\Lambda A \quad \text{implies} \quad \mathcal{C} \models A.$$

$\Lambda$ is *complete with respect to* $\mathcal{C}$, if, for any $A$,

$$\mathcal{C} \models A \quad \text{implies} \quad \vdash_\Lambda A.$$

$\Lambda$ is *determined by* $\mathcal{C}$ if it is both sound and complete with respect to $\mathcal{C}$.

**Normal Logics**

A logic $\Lambda$ is *normal* if it contains the schema

$$K_i : [i](A \to B) \to ([i]A \to [i]B),$$

and satisfies the *Necessitation* rule

$$\vdash_\Lambda A \quad \text{implies} \quad \vdash_\Lambda [i]A,$$

for every $i \in I$. The smallest normal logic will be denoted $K_I$.

In a normal logic,

$$\begin{array}{ll} \text{if} & \vdash_\Lambda A_1 \wedge \ldots \wedge A_n \to B, \\ \text{then} & \vdash_\Lambda [i]A_1 \wedge \ldots \wedge [i]A_n \to [i]B. \end{array}$$

**Examples of Normal Logics**

- For any class $\mathcal{C}$ of models, or of frames,
$$\Lambda_{\mathcal{C}} = \{A : \mathcal{C} \models A\}$$
is a normal logic. In fact (almost) all the logics we will be dealing with are normal.
- If $\{\Lambda_x : x \in X\}$ is a collection of normal logics, then
$$\bigcap\{\Lambda_x : x \in X\}$$
is normal. In particular,
$$K_I = \bigcap\{\Lambda : \Lambda \text{ is a normal logic}\}$$
is the *smallest* normal logic. The use of the letter $K$ here is in honour of Kripke.

The $K_I$-theorems are valid in all frames, hence true in all models (Soundness). The first basic completeness theorem in modal logic is that the converse is true, so altogether $K_I$ is determined by the class of all frames, as well as by the class of all models.

**Axioms for New Logics**

The notation

$$K_I\Sigma_1 \ldots \Sigma_n$$

denotes the smallest normal logic containing the schemata, or axioms, $\Sigma_1, \ldots, \Sigma_n$. Set-theoretically this logic is defined as

$$\bigcap\{\Lambda : \Lambda \text{ is normal and } \Sigma_1 \cup \ldots \cup \Sigma_n \subseteq \Lambda\}.$$

Proof-theoretically, the logic is characterised by the following result.

*Formula $A$ is a theorem of $K_I\Sigma_1 \ldots \Sigma_n$ iff there is a finite "proof sequence"*

$$A_0, \ldots, A_m = A$$

*such that for all $k \leq m$, $A_k$ is either*

(i) *a tautology ; or*
(ii) *an instance of a schema $K_i$ ; or*
(iii) *an instance of some $\Sigma_j$ ; or*
(iv) *deducible by Detachment from two previous members of the sequence, i.e. $A_r = (A_s \to A_k)$ for some $r, s < k$ ; or*
(v) *deducible by Necessitation from some previous member of the sequence, i.e. $A_k = [i]A_r$ for some $r < k$ and some $i \in I$.*

To prove this, let $T$ be the set of formulae $A$ for which there exists such a proof sequence. Then

- Any member of $T$ belongs to $K_I\Sigma_1 \ldots \Sigma_n$ (by induction along proof sequences),
- $T$ is a normal logic containing $\Sigma_1 \cup \ldots \cup \Sigma_n$, hence containing $K_I\Sigma_1 \ldots \Sigma_n$,

so $T = K_I\Sigma_1 \ldots \Sigma_n$.

Deleting clause (iii) above leaves us with a proof-theoretic characterisation of the logic $K_I$ determined by the class of all models.

**Finite Model Property**

This concerns the issue of whether a logic $\Lambda$ is determined by its finite models, in the sense that

if $\nvdash_\Lambda A$, then there is a finite model $\mathcal{M}$ with $\mathcal{M} \models \Lambda$ and $\mathcal{M} \not\models A$.

In fact a strong version of this is needed, in which the size of the counter-model can be calculated. A logic $\Lambda$ has the *finite model property* if there is a computable function $f$ such that

if $\nvdash_\Lambda A$, then there is a finite $\Lambda$-model that falsifies $A$ and has at most $f(n)$ elements, where $n = |A|$ is the "length" of $A$.

Now for a given $n$ there are only finitely many models $\mathcal{M}$ of size at most $f(n)$. It is effectively decidable whether a given formula $A$ is true in a given finite model $\mathcal{M}$. Moreover, if $\Lambda$ is *finitely axiomatisable*, meaning that $\Lambda = K\Sigma_1 \ldots \Sigma_n$ for some finite number of schemata $\Sigma_j$, then the property "$\mathcal{M} \models \Lambda$" is decidable: it suffices to determine whether each $\Sigma_j$ is true in $\mathcal{M}$. This yields an algorithm for determining, for each formula $A$, whether or not $\vdash_\Lambda A$ : examine all models of size at most $f(|A|)$ and check whether $A$ is falsifiable (or $\neg A$ is satisfiable) in any of those that are $\Lambda$-models. Hence

- *a finitely axiomatisable logic with the finite model property is decidable.*

Actually the existence of the computable function $f$ is not essential here: decidability can be shown assuming only that $\Lambda$ is determined by its finite models and is effectively enumerable. But most proofs of the finite model property do involve showing that such an $f$ exists (often it is an exponential function) and this is releveant to determining the complexity of the decision problem for $\Lambda$-theoremhood.

**Some Points in History**

- Aristotle investigated logical relationships between the necessary, the impossible, the possible, and the permitted.
- The Stoic logician Diodorus described the possible in temporal terms as being that which *either is or will be*, the necessary as that which *being true, will not be false*, etc.
- C. I. Lewis (*A Survey of Symbolic Logic*, 1918), initiated the modern symbolic analysis of modality, defining "$A$ strictly implies $B$" as $\mathrm{I}(A \wedge \neg B)$, where I is an *impossibility* operator, and developing the logical systems S1 – S5 based directly on axioms for strict implication.
- Kurt Gödel 1932 put forward a modal logic based on "the ordinary propositional calculus", with the schemata $K$, $\Box A \to A$, and $\Box A \to \Box\Box A$ and the rule of Necessitation. He proposed that $\Box A$ be read as "$A$ is provable", stated that his logic was equivalent to Lewis' S4, and described a translation into it from intuitionistic logic.
- J. C. C. McKinsey 1941 used algebraic methods ( Boolean algebras with operators) to prove the decidability of Lewis' S2 and S4.
- Saul Kripke 1959, 1963 developed the relational semantics for modal logics described here.
- Vaughan Pratt 1976 introduced dynamic logic.
- Amir Pnueli 1977 proposed the use of temporal logic to formalise the behaviour of continually operating concurrent programs.

**References**

Robert Goldblatt, *Logics of Time and Computation*, CSLI Lecture Notes No. 7, Centre for the Study of Language and Information, Stanford University, Second Edition, 1992, (distributed by University of Chicago Press).

Robert Goldblatt, *Mathematics of Modality*, CSLI Lecture Notes No. 43, Centre for the Study of Language and Information, Stanford University, 1993, (distributed by University of Chicago Press).

G. E. Hughes and M. J. Cresswell, *An Introduction to Modal Logic*, Methuen, 1968.

E. J. Lemmon (with Dana Scott), *An Introduction to Modal Logic*, American Philosophical Quarterly Monograph Series, no. 11 (ed. by Krister Segerberg), Basil Blackwell, Oxford, 1977.

## 2 Propositional Dynamic Logic

*PDL* is an *exogenous* logic of programs: its programs occur explicitly as part of its syntax, and appear within formulae. They are interpreted as input/output relations on states. Each program $\alpha$ is associated with a modal connective $[\alpha]$, the formula $[\alpha]A$ being read "after $\alpha$ terminates, $A$", i.e. "after every terminating execution of $\alpha$, $A$ is true" (allowing that a non-deterministic $\alpha$ may be executed in more than one way). The dual formula $<\alpha>A$ then means "there is an execution of $\alpha$ that terminates with $A$ true".

**Syntax**

| | |
|---|---|
| Atomic formulae: | $p \in \Phi$ |
| Atomic programs: | $\pi \in \Pi$ |
| Formulae: | $A \in Fma(\Phi, \Pi)$ |
| Programs: | $\alpha \in Prog(\Phi, \Pi)$ |

$$A ::= p \mid \bot \mid A_1 \to A_2 \mid [\alpha]A$$

$$\alpha ::= \pi \mid \alpha_1; \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha^* \mid A?$$

Intended meanings are:

| | |
|---|---|
| $[\alpha]A$ | after $\alpha$, $A$, |
| $\alpha_1; \alpha_2$ | do $\alpha_1$ and then $\alpha_2$ (*composition*), |
| $\alpha_1 \cup \alpha_2$ | do either $\alpha_1$ or $\alpha_2$ non-deterministically (*alternation*), |
| $\alpha^*$ | repeat $\alpha$ some finite number ($\geq 0$) of times (*iteration*), |
| $A?$ | *test* $A$: continue if $A$ is true, otherwise "fail". |

Further constructs are introduced by definitional abbreviation:

| | | |
|---|---|---|
| $<\alpha>A$ | is | $\neg[\alpha]\neg A$, |
| **if** $A$ **then** $\alpha$ **else** $\beta$ | is | $(A?; \alpha) \cup (\neg A?; \beta)$ |
| **while** $A$ **do** $\alpha$ | is | $(A?; \alpha)^*; \neg A?$ |
| **repeat** $\alpha$ **until** $A$ | is | $\alpha; (\neg A?; \alpha)^*$ |
| **skip** | is | $\top?$ |
| **abort** | is | $\bot?$ |
| $\alpha^0$ | is | **skip** |
| $\alpha^{n+1}$ | is | $(\alpha; \alpha^n)$ |

**Standard Models**

A model for the language just described is a structure of the form

$$\mathcal{M} = (S, \{R_\alpha : \alpha \in Prog(\Phi, \Pi)\}, V),$$

with $R_\alpha$ a binary relation on $S$ for each program $\alpha$, and

$$\mathcal{M} \models_s [\alpha]A \quad \text{iff} \quad sR_\alpha t \text{ implies } \mathcal{M} \models_t A.$$

The binary relations $R_\alpha$ should reflect the intended meanings of programs $\alpha$. Thus a model $\mathcal{M}$ will be defined to be *standard* if it satisfies the following conditions:

$$\begin{aligned} R_{\alpha;\beta} &= R_\alpha \circ R_\beta = \{(s,t) : \exists u(sR_\alpha u \ \&\ uR_\beta t)\}; \\ R_{\alpha\cup\beta} &= R_\alpha \cup R_\beta; \\ R_{\alpha^*} &= R_\alpha^* = \text{ ancestral, or reflexive-transitive closure, of } R_\alpha; \\ R_{A?} &= \{(s,s) : \mathcal{M} \models_s A\}. \end{aligned}$$

Hence $sR_{\alpha^*}t$ iff for some $n \geq 0$ there is a sequence

$$s_0 R_\alpha s_1 R_\alpha \cdots R_\alpha s_n$$

with $s = s_0$ and $s_n = t$.

There are no constraints on the $R_\pi$'s. This means that given a structure

$$(S, \{R_\pi : \pi \in \Pi\}, V)$$

which assigns a binary relation to each atomic program, a uniquely determined standard model is obtained by using the above standard model conditions to inductively *define* $R_\alpha$ for non-atomic programs $\alpha$.

*PDL* is defined formally to be the logic determined by the standard models, i.e.

$$PDL = \{A \in Fma(\Phi, \Pi) \colon A \text{ is true in all standard models}\}.$$

**Execution Sequences**

In a standard model, any particular execution of a program consists of a finite sequence of atomic programs and/or tests. To substantiate this, associate with each $\alpha$ a set $\tau(\alpha)$ of programs of this form, by defining

$$\begin{aligned}\tau(\pi) &= \{\pi\}\\ \tau(A?) &= \{A?\}\\ \tau(\alpha_1;\alpha_2) &= \{\alpha;\beta\colon \alpha \in \tau(\alpha_1) \text{ and } \beta \in \tau(\alpha_2)\}\\ \tau(\alpha_1 \cup \alpha_2) &= \tau(\alpha_1) \cup \tau(\alpha_2)\\ \tau(\alpha^*) &= \textstyle\bigcup_{n\geq 0}\tau(\alpha^n).\end{aligned}$$

Then in a standard model, for any program $\alpha$,

$$sR_\alpha t \quad \text{iff} \quad sR_\beta t \text{ for some } \beta \in \tau(\alpha).$$

**Partial Correctness Assertions**

These are used to specify program behaviour, and their derivation is part of the methodology of *verification* of programs. They typically take the form

*if $A$ is true before initiation of $\alpha$, then $B$ will be true when $\alpha$ terminates,*

which is rendered symbolically as

$$A\{\alpha\}B,$$

in a notation introduced by Hoare in 1969. Such a statement can be expressed in $PDL$-language as

$$A \to [\alpha]B.$$

For instance, the *Iteration Rule* of Hoare logic takes the form

$$\frac{A \wedge B \to [\alpha]B}{B \to [\textbf{while } A \textbf{ do } \alpha](B \wedge \neg A)}$$

which is sound with respect to truth in standard models.

**Small Model Property**

By the filtration (quotient model) method, a formula $A$ that is satisfiable in a standard model can be shown to be satisfiable in one whose size is at most $2^{|A|}$, exponential in $|A|$. Hence $PDL$ has the finite model property, and is decidable. It has a decision procedure that runs in deterministic exponential time.

**Axiomatisation**

$PDL$ is the smallest normal logic in $Fma(\Phi, \Pi)$ that contains the schemata

$$\begin{array}{ll} \textit{Test}: & [A?]B \leftrightarrow (A \to B),\\ \textit{Comp}: & [\alpha;\beta]A \leftrightarrow [\alpha][\beta]A,\\ \textit{Alt}: & [\alpha\cup\beta]A \leftrightarrow [\alpha]A \wedge [\beta]A,\\ \textit{Mix}: & [\alpha^*]A \to A \wedge [\alpha][\alpha^*]A,\\ \textit{Ind}: & [\alpha^*](A \to [\alpha]A) \to (A \to [\alpha^*]A). \end{array}$$

The last of these is the *Induction* axiom, expressing that if truth of $A$ is preserved by program $\alpha$, and $A$ is true now, then it will remain true after any number of iterations of $\alpha$. This axiom can be replaced by the inference rule

$$\textit{if} \vdash A \to [\alpha]A \quad \textit{then} \quad \vdash A \to [\alpha^*]A.$$

**Determinism**

A program is deterministic if its outcome is uniquely determined by its initial state. The alternation $(\alpha\cup\beta)$ and iteration $(\alpha^*)$ constructs are inherently non-deterministic, but non-determinism also arises in $PDL$ by allowing atomic programs to be modelled by relations, rather than functions, between states.

The logic *DPDL* is determined by those standard *PDL*-models in which the relations $R_\pi$ are *functional* for all atomic programs $\pi$. This logic is axiomatised by adding to the *PDL* axioms the schema

$$<\pi>A \rightarrow [\pi]A.$$

*DPDL* is decidable in deterministic exponential time, and has the finite model property with a bound on the size of a model for satisfiable $A$ of $|A|^2 \cdot 4^{|A|}$.

*Strict Deterministic Propositional Dynamic Logic* (*SDPDL*) is the sublogic of *DPDL* obtained by syntactically restricting the program operators $\cup$ and $^*$ to appear only in programs of the form

$$\textbf{if } A \textbf{ then } \alpha \textbf{ else } \beta \quad \text{and} \quad \textbf{while } A \textbf{ do } \alpha.$$

*SDPDL* is a strictly weaker logic than *DPDL*. It is finitely axiomatisable and decidable, with a decision procedure that runs in polynomial space.

## Termination

The formula $<\alpha>\top$ expresses that $\alpha$ has a halting execution, but does not imply that $\alpha$ ***must*** halt, since it may also have non-halting executions if it is non-deterministic. Failure to terminate may arise because some execution of (part of) $\alpha$ might "run forever" or "loop", or because $\alpha$ can be repeatedly executed. The latter case is expressed by a formula

$$\textbf{repeat}(\alpha),$$

with the semantics

$$\mathcal{M} \models_s \textbf{repeat}(\alpha) \quad \text{iff} \quad \text{there exists an infinite sequence } s = s_0, s_1, s_3, \ldots \text{ such that } s_n R_\alpha s_{n+1} \text{ for all } n \geq 0.$$

Extending the syntax of *PDL* to allow formation of formulae **repeat**($\alpha$) for all programs $\alpha$, the system *RPDL* is defined to be that determined by the standard *PDL*-models with this extended semantics.

The other notion of non-termination is given by formulae

$$\textbf{loop}(\alpha),$$

with the inductively defined modelling

$$\begin{array}{lll}
\mathcal{M} \not\models_s \textbf{loop}(\pi) & & \\
\mathcal{M} \not\models_s \textbf{loop}(A?) & & \\
\mathcal{M} \models_s \textbf{loop}(\alpha;\beta) & \text{iff} & \mathcal{M} \models_s \textbf{loop}(\alpha) \text{ or } \mathcal{M} \models_s <\alpha>\textbf{loop}(\beta) \\
\mathcal{M} \models_s \textbf{loop}(\alpha \cup \beta) & \text{iff} & \mathcal{M} \models_s \textbf{loop}(\alpha) \text{ or } \mathcal{M} \models_s \textbf{loop}(\beta) \\
\mathcal{M} \models_s \textbf{loop}(\alpha^*) & \text{iff} & \mathcal{M} \models_s <\alpha^*>\textbf{loop}(\alpha) \text{ or there exist } s = s_0, s_1, s_3, \ldots \\
& & \text{such that } s_n R_\alpha s_{n+1} \text{ for all } n \geq 0.
\end{array}$$

This defines a logic *LPDL* which is stronger than *PDL* but weaker than *RPDL*. In fact **loop**($\alpha$) can be syntactically defined in *RPDL* by inductively putting

$$\begin{aligned}
\textbf{loop}(\pi) &= \textbf{loop}(A?) = \bot \\
\textbf{loop}(\alpha;\beta) &= \textbf{loop}(\alpha) \vee <\alpha>\textbf{loop}(\beta) \\
\textbf{loop}(\alpha \cup \beta) &= \textbf{loop}(\alpha) \vee \textbf{loop}(\beta) \\
\textbf{loop}(\alpha^*) &= <\alpha^*>\textbf{loop}(\alpha) \vee \textbf{repeat}(\alpha).
\end{aligned}$$

The formula **halt**($\alpha$), defined as the negation

$$\neg\textbf{loop}(\alpha),$$

then expresses "$\alpha$ always halts", i.e. "every execution of $\alpha$ terminates".

*LPDL* and *RPDL* both have the finite model property and are decidable. The proof uses the theory of automata on infinite trees.

### Total Correctness and Weakest Preconditions

The statement

*if $A$ is true before initiation of $\alpha$, then execution of $\alpha$ must terminate, with $B$ true afterwards,*

asserts the *total correctness* of $\alpha$ with respect to pre- and post-conditions $A$ and $B$. If $\alpha$ is deterministic, this can be symbolised as

$$A \to <\alpha>B,$$

but in general the appropriate formulation is

$$A \to <\alpha>\top \wedge \mathbf{halt}(\alpha) \wedge [\alpha]B.$$

The consequent

$$<\alpha>\top \wedge \mathbf{halt}(\alpha) \wedge [\alpha]B$$

would appear to be an accurate representation of Dijkstra's *weakest precondition* $\mathbf{wp}(\alpha, B)$ for program $\alpha$ corresponding to post-condition $B$. This was defined by Dijkstra as

" the condition that characterises the set of all initial states such that activation will certainly result in a properly terminating happening leaving the system in a final state satisfying a given post-condition."

### Concurrent PDL

This extends the syntax of programs by the construct $\alpha \cap \beta$, interpreted as "$\alpha$ and $\beta$ executed in parallel", the idea being that $\alpha$ and $\beta$ are processes acting independently *at the same time.*

The result of an execution started in state $s$ will not now be a single terminal state $t$, but rather a set $T$ of states representing the terminal situations of all the parallel processes involved. Thus the relation $R_\alpha$ interpreting command $\alpha$ is no longer a set of pairs $(s, t)$, but rather a set of pairs $(s, T)$, with $s$ a member of the state-set $S$, and $T \subseteq S$.

The meaning of $<\alpha>A$ as "there is an execution of $\alpha$ that terminates with $A$ true", is modelled by putting

$$\mathcal{M} \models_s <\alpha>A \quad \text{iff} \quad \text{there exists } T \subseteq S \text{ with } sR_\alpha T \text{ and } T \subseteq V(A),$$

where $V(A)$ is the denotation (page 16) given by

$$V(A) = \{t \in S : \mathcal{M} \models_t A\}.$$

The modelling of $[\alpha]A$ as "after every terminating execution of $\alpha$, $A$ is true", becomes

$$\mathcal{M} \models_s [\alpha]A \quad \text{iff} \quad sR_\alpha T \quad \text{implies} \quad T \subseteq V(A),$$

making $[\alpha]$ and $<\alpha>$ no longer interdefinable via $\neg$.

The standard model condition for the new operation $\alpha \cap \beta$ is

$$R_{\alpha\cap\beta} = \{(s, T \cup W) : sR_\alpha T \text{ and } sR_\beta W\},$$

and the conditions for all the other program operations require reformulation.

Details of this logic, with proofs of decidability and the finite model property, can be found in Chapter 10 of [Goldblatt, 1992].

### Quantificational Dynamic Logic

Although our concern is with propositional logic, let us briefly consider an example that demonstrates the strength of the program modalities in the presence of quantifiers. The formula

$$\forall y <x := 0; \mathbf{while}\ x \neq y\ \mathbf{do}\ x := x + 1>\top,$$

in the language of the arithmetic of natural numbers, asserts that for all $y$ the displayed program has a terminating execution, i.e. that any $y$ can be obtained by starting at 0 and repeatedly applying the successor operation $x \mapsto x + 1$. In other words: any set of numbers that contains 0 and is closed under successors must contain everything. But this is a version of the Peano Induction Postulate, a postulate which cannot be expressed in the first-order language of the structure $(\omega, x \mapsto x + 1, 0)$. There is a single formula of dynamic logic which characterises this structure up to isomorphism, and from this it follows, by standard arguments, that the set of valid dynamic formulae is not effectively enumerable, unlike the case of non-modal quantification logic.

**Further Study**

There are many other topics in *PDL*, including retrictions of test programs, converse programs, programs as automata, recursively enumerable sets of execution sequences, undecidable logics, and logics without the finite model property. Then there is the extensive study of quantificational dynamic logic. The following surveys give information about these, and provide references to an extensive literature.

David Harel, *Dynamic Logic*, in Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic, D. Gabbay and F. Guenthner (eds.), D. Reidel, 1984, pp. 497–604.

Dexter Kozen & Jerzy Tiuryn, *Logics of Programs*, in Handbook of Theoretical Computer Science, vol. B, J. van Leeuwen (ed.), Elsevier/The MIT Press, 1990, pp. 789–840.

## 3 Temporal Logic of Concurrency

The language to be considered now is *endogenous*: there are no programs within the syntax, and formulae describe the computational situation, over time, of a single implicit program. This is regarded as particularly appropriate to the specification and analysis of *reactive* programs, like operating systems and systems for airline reservation or process control, that maintain an interaction with their environment and are not expected to terminate.

**Shared Memory Parallelism**

Consider the following description of a "concurrent" program. There are $n$ different processes acting in parallel, using a shared memory environment, so that each can alter the values of variables used by the others. For illustrative purposes, the processes may be thought of as disjoint flowcharts, with labelled nodes. A typical node of the $i$-th process is denoted $m^i$. Each process has an entry node $m^i_0$. If the program variables are $v_1, \ldots, v_k$, then a *state* may be defined as a vector

$$s = (m^1, \ldots, m^n, a_1, \ldots, a_k),$$

specifying a label for each process (denoting the point that the process is currently at), and a current value $a_i$ for each variable $v_i$. Predicates $at_i$ of labels will be used, with the semantics

$$\models_s at_i(m) \quad \text{iff} \quad m = m^i.$$

Each successive state is to be obtained from its predecessor by exactly one process being chosen to execute one transition in its flow chart. Thus from an initial state

$$s_0 = (m^1_0, \ldots, m^n_0, a_1, \ldots, a_k),$$

many different execution sequences $s_0, s_1, \ldots\ldots$ may be generated, depending on which process gets chosen to act at each step. Some interesting properties of such sequences can be formulated by reading the connective $\Box$ as "at all states from now on". Thus we use the temporal modality [F], and its dual <F>, interpreting them as the *state quantifiers* "at all future states" and "at some future state", respectively.

- **Mutual Exclusion**

$$[\mathrm{F}]\neg(at_i(m) \wedge at_j(m'))$$

asserts that the program can never simultaneously access $m$ and $m'$.

- **Accessibility**

$$[\mathrm{F}](at_i(m) \rightarrow <\mathrm{F}> at_j(m'))$$

expresses that if the program ever reaches $m$ it will eventually proceed from there to $m'$.

- **Deadlock Freedom**

Deadlock occurs when no processor can act. The requirement that deadlock does not occur at $(m^1, \ldots, m^n)$ can be expressed by

$$[\mathrm{F}](at_1(m^1) \wedge \cdots \wedge at_n(m^n) \rightarrow E_1 \vee \cdots \vee E_n),$$

where $E_i$ is the *exit condition* for node $m^i$ consisting of the disjunction of the propositions labelling edges out of $m^i$ (the truth of such a proposition being the requirement for the process to be able to proceed along that edge).

• **Fairness**

The formula $[\mathrm{F}] <\mathrm{F}> A$ expresses that at all future states there will be a later state at which $A$ is true, i.e. $A$ will be true infinitely often. Now if a particular process is infinitely often able to act, then in fair scheduling of processes it should not have to wait forever. If $A_i$ expresses that the $i$-th process is enabled, and $B_i$ that it is selected for execution, then one way to express this fairness requirement is by the formula

$$[\mathrm{F}] <\mathrm{F}> A_i \to [\mathrm{F}] <\mathrm{F}> B_i.$$

• **Correctness**

A *partial correctness* assertion about a program states that if the program works as was intended, then a certain post-condition $B$ must be true after termination, given that some pre-condition $A$ was true at the start. Illustrating with a program having a single entry label $m_0$, and exit $m_e$, this can be formalised as

$$at(m_0) \wedge A \to [\mathrm{F}](at(m_e) \to B).$$

*Total* correctness includes the assertion that the program will halt:

$$at(m_0) \wedge A \to <\mathrm{F}> (at(m_e) \wedge B).$$

• **Responsiveness**

An operating system may receive requests $(r_i)$ from various agents, to whom it will signal $(q_i)$ when it grants the request. The formula

$$[\mathrm{F}](r_i \to <\mathrm{F}> q_i)$$

expresses that a request is always eventually honoured.

• **Absence of Unsolicited Response**

This uses the connective $A\mathcal{U}B$ ("it will be $A$ until $B$") to express the requirement that if a response is to occur, it will not do so until a request has been received:

$$<\mathrm{F}> q_i \to (\neg q_i)\mathcal{U}r_i.$$

**Syntax**

Given a set $\Phi$ of atomic formulae $p$ as usual, define a set of formulae $A \in Fma(\Phi)$ by the specification

$$A ::= p \mid \perp \mid A_1 \to A_2 \mid [\mathrm{X}]A \mid A_1\mathcal{U}A_2$$

$[\mathrm{X}]$ means "next" (i.e. at the next state).
$\mathcal{U}$ means "until".
$[\mathrm{F}]A$ is defined as the formula $\neg(\top\mathcal{U}\neg A)$, and means "henceforth" (i.e. from now on, including the present).
$<\mathrm{F}> A$ ("eventually $A$") is defined as $\top\mathcal{U}A$, so $[\mathrm{F}]A = \neg <\mathrm{F}> \neg A$.

**Semantics**

A *state sequence* is a pair $\mathcal{F} = (S, \sigma)$, where $\sigma$ is a surjective function $\omega \to S$ enumerating $S$ as a sequence

$$\sigma_0, \sigma_1, \ldots, \sigma_j, \ldots\ldots$$

(possibly with repetition, for example when $S$ is finite).

A *model* $\mathcal{M} = (S, \sigma, V)$ on a state sequence is defined in the usual way, and the relation

$$\mathcal{M} \models_j A,$$

meaning "$A$ is true at the $j$-th state $\sigma_j$ in $\mathcal{M}$", is given by

$$\begin{array}{lll} \mathcal{M} \models_j p & \text{iff} & \sigma_j \in V(p) \\ \mathcal{M} \not\models_j \perp & & \\ \mathcal{M} \models_j A \to B & \text{iff} & \mathcal{M} \models_j A \text{ implies } \mathcal{M} \models_j B \\ \mathcal{M} \models_j [\mathrm{X}]A & \text{iff} & \mathcal{M} \models_{j+1} A \\ \mathcal{M} \models_j A\mathcal{U}B & \text{iff} & \text{for some } k \geq j,\ \mathcal{M} \models_k B \text{ and} \\ & & \text{for every } i \text{ such that } j \leq i < k, \\ & & \mathcal{M} \models_i A. \end{array}$$

The definitions of the relations $\mathcal{M} \models A$ and $\mathcal{F} \models A$ are as usual.

The semantics gives

$$\begin{array}{lll} \mathcal{M} \models_j [\mathrm{F}]A & \text{iff} & \text{for all } k \geq j,\ \mathcal{M} \models_k A \\ \mathcal{M} \models_j <\mathrm{F}> A & \text{iff} & \text{for some } k \geq j,\ \mathcal{M} \models_k A. \end{array}$$

Intuitively, this amounts to interpreting $[\mathrm{F}]$ and $<\mathrm{F}>$ by the relation $\leq$, and $[\mathrm{X}]$ by the relation $R$, where $jRk$ iff $k = j + 1$. $R$ is functional, and the connection between the two relations is that $\leq$ is the *ancestral* (reflexive transitive closure) of $R$. Thus $[\mathrm{X}]$ is indeed a "Box-type" modality.

## Propositional Linear Temporal Logic

*PLTL* is the logic determined by the class of all (models on) state sequences. It has a polynomial space decision algorithm, and is finitely axiomatisable, being the smallest [X]-normal logic in the present language that contains the schemata

| | |
|---|---|
| *Fun*: | $[X]\neg A \leftrightarrow \neg[X]A$ |
| $\mathcal{U}1$: | $A\mathcal{U}B \rightarrow <F> B$ |
| $\mathcal{U}2$: | $A\mathcal{U}B \leftrightarrow B \vee (A \wedge [X](A\mathcal{U}B))$ |

and is closed under the $<F>$*-Rule*

$$\text{if } \vdash A \rightarrow B \text{ then } \vdash <F> A \rightarrow <F> B,$$

and the *Induction Rule*

$$\text{if } \vdash A \rightarrow [X]A \text{ then } \vdash A \rightarrow [F]A.$$

A proof that this axiomatisation is complete for the *PLTL* semantics will be presented in Section 7.

If the rule of Necessitation for [F] was included here, then the Induction Rule could be replaced by the schema

$$[F](A \rightarrow [X]A) \rightarrow (A \rightarrow [F]A),$$

which is the analogue of the *PDL*-axiom *Ind*. As will be seen, it is the Induction Rule that lies at the heart of the completeness proof, and Necessitation for [F] is not needed.

The analogue

$$[F]A \rightarrow A \wedge [X][F]A$$

of the *PDL* axiom *Mix* is derivable from *Fun* and $\mathcal{U}2$.

## Branching Time

*Linear* Temporal Logic is concerned with logical properties of a *single* execution sequence $s_0, s_1, \ldots\ldots$ generated by processes acting in parallel. As mentioned at the outset, each state will have several possible successor states, and so there will be many different sequences that have a given starting state $s_0$. Thus any particular sequence will be but one "branch" of the "tree" of all possible future states. If we consider this tree as a whole, there a number of interesting new modal connectives that can be used to formalise reasoning about future behaviour:

| | |
|---|---|
| $\forall<F>A$: | along any future branch there is a state at which $A$ is true, i.e. $A$ is *inevitable*. |
| $\exists<F>A$: | along some branch there is a state at which $A$ is true, i.e. $A$ is *potentially true*. |
| $\forall[F]A$: | along all branches, $A$ holds at all states, i.e. $A$ is true at all possible future states. |
| $\exists[F]A$: | along some branch, $A$ holds at all states. |
| $\forall[X]A$: | along every branch, $A$ holds at the next state, i.e. $A$ holds at all possible successor states. |
| $\exists[X]A$: | $A$ holds at some successor state. |
| $\forall(A\mathcal{U}B)$: | along every branch, it will be $A$ until $B$. |
| $\exists(A\mathcal{U}B)$: | along some branch, it will be $A$ until $B$. |

A logical system embodying these notions, known as Computational Tree Logic, will now be set out.

## Syntax of *CTL*

This is specified by

$$A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid \forall[X]A \mid \forall(A_1\mathcal{U}A_2) \mid \exists(A_1\mathcal{U}A_2)$$

The other connectives mentioned above are given by the following abbreviations.

| | | |
|---|---|---|
| $\forall<F>A$ | is | $\forall(\top\mathcal{U}A)$ |
| $\exists<F>A$ | is | $\exists(\top\mathcal{U}A)$ |
| $\forall[F]A$ | is | $\neg\exists(\top\mathcal{U}\neg A)$ |
| $\exists[F]A$ | is | $\neg\forall(\top\mathcal{U}\neg A)$ |
| $\exists[X]A$ | is | $\neg\forall[X]\neg A$ |

**Semantics**

To define *CTL*-models, consider a frame $\mathcal{F} = (S, R)$ in which $R$ is total, i.e. $\forall s \exists t(sRt)$. Here $sRt$ will be interpreted to mean that $t$ is a *possible immediate successor* to $s$. An *R-branch starting at s* in $\mathcal{F}$ is an infinite sequence $s_0, \ldots, s_n, \ldots$ with $s = s_0$ and $s_n R s_{n+1}$ for all $n$. An *R-path* is a finite version of a branch, i.e. a sequence $s_0, \ldots, s_k$ with $s_n R s_{n+1}$ for all $n < k$. By totality of $R$, any path extends to a branch.

Given the usual notion of a model $\mathcal{M} = (S, R, V)$ on such a frame, satisfaction of *CTL*-formulae is given by

| | | |
|---|---|---|
| $\mathcal{M} \models_s \forall[\mathrm{X}]A$ | iff | for all $t \in S$, $sRt$ implies $\mathcal{M} \models_t A$. |
| $\mathcal{M} \models_s \forall(A\mathcal{U}B)$ | iff | for all $R$-branches $s = s_0 R s_1 R \cdots$ there exists $k$ with $\mathcal{M} \models_{s_k} B$ and $\mathcal{M} \models_{s_i} A$ whenever $0 \leq i < k$. |
| $\mathcal{M} \models_s \exists(A\mathcal{U}B)$ | iff | for some $R$-branch $s = s_0 R s_1 R \cdots$ there exists $k$ with $\mathcal{M} \models_{s_k} B$ and $\mathcal{M} \models_{s_i} A$ whenever $0 \leq i < k$. |

**Axioms**

*CTL* is the logic determined by the models just described. It is the smallest logic that contains the schemata

$$\begin{array}{ll} K_{\mathrm{X}}: & \forall[\mathrm{X}](A \to B) \to (\forall[\mathrm{X}]A \to \forall[\mathrm{X}]B) \\ D_{\mathrm{X}}: & \exists[\mathrm{X}]\top \\ \exists\mathcal{U}: & \exists(A\mathcal{U}B) \leftrightarrow (B \vee (A \wedge \exists[\mathrm{X}]\exists(A\mathcal{U}B))) \\ \forall\mathcal{U}: & \forall(A\mathcal{U}B) \leftrightarrow (B \vee (A \wedge \forall[\mathrm{X}]\forall(A\mathcal{U}B))) \end{array}$$

and is closed under Necessitation for $\forall[\mathrm{X}]$ i.e.,

$$\vdash A \quad \text{implies} \quad \vdash \forall[\mathrm{X}]A,$$

and under the following two rules:

$$\begin{array}{llll} \exists\text{-}Ind: & \vdash B \vee (A \wedge \exists[\mathrm{X}]C) \to C & \text{implies} & \vdash \exists(A\mathcal{U}B) \to C \\ \forall\text{-}Ind: & \vdash B \vee (A \wedge \forall[\mathrm{X}]C) \to C & \text{implies} & \vdash \forall(A\mathcal{U}B) \to C. \end{array}$$

*CTL* is decidable in deterministic exponential time.

**CTL***

In *CTL*, the *branch quantifiers* $\forall$ and $\exists$ are always tied to a single linear-time state quantifier (modality), as in the forms $\forall\langle\mathrm{F}\rangle$, $\exists[\mathrm{F}]$ etc. They cannot be applied to combination of modalities, as in a formula like $\forall[\mathrm{F}] < \mathrm{F} > A$, which would express "along every branch, $A$ is true infinitely often", of relevance to fairness conditions on branches.

The logic *CTL** allows path quantifiers to apply all formulae, whose syntax is

$$A ::= p \mid \bot \mid A_1 \to A_2 \mid [\mathrm{X}]A \mid (A_1\mathcal{U}A_2) \mid \forall A$$

(some versions distinguish between *branch-formulae* (or path-formulae) and *state-formulae*, but this version will suffice for illustration).

The semantics of formulae is now given by a satisfaction relation

$$\mathcal{M}, \sigma \models_j A,$$

similar to that for *PLTL*, but which names the $R$-branch (state-sequence) $\sigma$ explicitly. The truth conditions are

| | | |
|---|---|---|
| $\mathcal{M}, \sigma \models_j p$ | iff | $\sigma_j \in V(p)$ |
| $\mathcal{M}, \sigma \not\models_j \bot$ | | |
| $\mathcal{M}, \sigma \models_j A \to B$ | iff | $\mathcal{M}, \sigma \models_j A$ implies $\mathcal{M}, \sigma \models_j B$ |
| $\mathcal{M}, \sigma \models_j [\mathrm{X}]A$ | iff | $\mathcal{M}, \sigma \models_{j+1} A$ |
| $\mathcal{M}, \sigma \models_j A\mathcal{U}B$ | iff | for some $k \geq j$, $\mathcal{M}, \sigma \models_k B$ and for every $i$ such that $j \leq i < k$, $\mathcal{M}, \sigma \models_i A$. |
| $\mathcal{M}, \sigma \models_j \forall A$ | iff | for all $R$-branches $\sigma'$, and all $k$, if $\sigma_j = \sigma'_k$, then $\mathcal{M}, \sigma' \models_k A$. |

This defines a logic that is substantially stronger than *CTL*, with a decision problem that is complete for deterministic *double* exponential time (proven by the theory of automata on infinite trees). The existence of a (finite) axiomatisation of *CTL** is an open question.

### Applications: Model Checking and Synthesis

The *model checking problem* is that of deciding whether a given finite structure is a model of a particular formula. The complexity of the problem has been determined for various temporal logics, and it has practical applications, for instance to the automatic verification of finite-state concurrent systems whose specifications are expressed by temporal formulae.

Another application of logics like $LTL$ and $CTL$ is to the synthesis (construction) of the *synchronisation skeleton* of a concurrent program, this being an abstraction of the actual program containing only details relevant to synchronisation. The method is to test the satisfiability of a formula specifying the required synchronisation properties and, if satisfiable, to use the finite model property to construct a finite model of it. The skeleton can then be read from this model.

### Further Study

The article by Emerson in the following list of references provides the most extensive survey and bibliography of the logics discussed here and their applications.

J. W. de Bakker et. al. (eds.), *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency,* Lecture Notes in Computer Science, vol. 354, Springer-Verlag, 1989.

E. Allen Emerson, *Temporal and Modal Logic,* in Handbook of Theoretical Computer Science, vol. B, J. van Leeuwen (ed.), Elsevier/The MIT Press, 1990, pp. 995–1072.

Antony Galton,, *Temporal Logics and their Applications,* Academic Press, 1987.

Brent T. Hailpern, *Verifying Concurrent Processes Using Temporal Logic,* Lecture Notes in Computer Science vol. 129, Springer-Verlag, 1982.

Ben Moszkowski, *Executing Temporal Logic Programs,* Cambridge University Press, 1986.

Colin Stirling, *Modal and Temporal Logics,* in Handbook of Logic in Computer Science, vol. 2, S. Ambramsky et. al. (eds.), Clarendon Press, Oxford, 1992, pp. 447–563.

## 4 Hennessy-Milner Logic of Processes

Milner's *Calculus of Communicating Systems* ($CCS$) is an algebraic theory that provides operations for constructing new processes from given ones, and uses a modal language to express assertions about transitions between processes. Equivalence of processes can then be characterised in terms of this language: under certain conditions two processes prove to be equivalent just when they satisfy the same modal properties.

We will now develop enough of this conceptual framework to be able to explain this application of modal logic to process algebra.

### Processes=States

*An agent is a process that interacts with its environment by performing actions which cause it to change its state.*

Indeed in process algebra the words "agent" and "process" are synonymous, and processes are identified with their state. The notation for transitions is

$$P@ > \alpha >> Q,$$

which means that process $P$ can become $Q$ by performing, or participating in, the action $\alpha$. This could instead be written

$$P\, R_\alpha\, Q,$$

indicating already where modal logic comes into the picture!

### Prefixing

This operation takes a process $P$ and an action $\alpha$ and constructs a process $\alpha.P$ which can itself become $P$ by performing $\alpha$. This yields the transition relation

$$\alpha.P@ > \alpha >> P$$

### Choice

$P + Q$ is a process that can act by doing either $P$ or $Q$. Transitions involving this construct are obtain by the derivation rules

$$\frac{P@ > \alpha >> P'}{P + Q@ > \alpha >> P'} \qquad \frac{Q@ > \alpha >> Q'}{P + Q@ > \alpha >> Q'}$$

**Composition**

$P|Q$ is a process that performs $P$ and $Q$ concurrently. Its basic transition rules are

$$\frac{P@>\alpha>>P'}{P|Q@>\alpha>>P'|Q} \qquad \frac{Q@>\alpha>>Q'}{P|Q@>\alpha>>P|Q'}$$

**Concurrent Interaction**

Processes $P$ and $Q$ may communicate along a channel or at a port, provided one of them can perform an action $\alpha$ (e.g. "send") that has an associated co-action $\bar{\alpha}$ ("receive") that the other process can perform. This interaction is regarded as an internal unobservable action, or silent step, performed by the composite process $P|Q$, and denoted by $\tau$. It transition rule is

$$\frac{P@>\alpha>>P' \qquad Q@>\bar{\alpha}>>Q'}{P|Q@>\tau>>P'|Q'}$$

**(Recursive) Definitions**

Processes can be defined by setting a process name equal to some process expression, which may itself already contain that name, as in

$$Cl := tick.tock.Cl,$$

which defines a "clock" that ticks and then tocks forever. The rule for this is

$$\frac{P@>\alpha>>Q \qquad P:=E}{E@>\alpha>>Q}$$

**A Vending Machine**

$$M := \$1.\mathtt{teabutton.collect}.M + \$2.\mathtt{coffeebutton.collect}.M$$

**The Modal Language**

In the foregoing situation we envisage a *labelled transition system* $(\mathcal{P}, \{@>\alpha>>: \alpha \in \mathcal{A}\})$, in which $\mathcal{P}$ is a set of agents/processes/states, $\mathcal{A}$ is a set of action labels, and $@>\alpha>>$ is a binary transition relation on $\mathcal{P}$ for each $\alpha \in \mathcal{A}$. The set $Fma(\mathcal{A})$ of associated formulae of Hennessy-Milner logic is generated by the syntax

$$A ::= \bot \mid \neg A \mid A_1 \wedge A_2 \mid [\alpha]A.$$

Note especially that there are no atomic formulae.

**Satisfaction**

The relation $P \models A$, meaning "process $P$ satisfies property $A$", is given by

| | | |
|---|---|---|
| $P \not\models \bot$ | | |
| $P \models \neg A$ | iff | $P \not\models A$ |
| $P \models A_1 \wedge A_2$ | iff | $P \models A_1$ and $P \models A_2$ |
| $P \models [\alpha]A$ | iff | for all $Q \in \mathcal{P}$, $P@>\alpha>>Q$ implies $Q \models A$. |

**Bisimulation**

Two processes are regarded as equivalent if there is no observable action that either can perform to distinguish them. Informally this means that to each action that one can perform there is an action that the other can perform which is equivalent, i.e. leads to an equivalent outcome. Thus each process can *simulate* the other.

To formalise this, declare a binary relation $\rho$ on $\mathcal{P}$ to be a *bisimulation* if whenever $P\rho Q$ then for any $\alpha \in \mathcal{A}$,

- if $P@>\alpha>>P'$ then $Q@>\alpha>>Q'$ and $P'\rho Q'$ for some $Q'$; and
- if $Q@>\alpha>>Q'$ then $P@>\alpha>>P'$ and $P'\rho Q'$ for some $P'$.

Now put

$$P \sim Q \quad \text{iff} \quad P\rho Q \text{ for some bisimulation } \rho.$$

Then $\sim$ is the desired relation of *observational*, or *bisimulation, equivalence*.

Here is another way to view the definition of $\sim$. For an arbitrary binary relation $\rho \subseteq \mathcal{P} \times \mathcal{P}$, let $F(\rho)$ be the set of all pairs $(P,Q)$ that satisfy the above two "back-and-forth" clauses in the definition of bisimulation. Then $\rho$ is a bisimulation iff $\rho \subseteq F(\rho)$. Hence

$$\sim = \bigcup\{\rho : \rho \subseteq F(\rho)\}.$$

$\sim$ itself is a bisimulation and so is the *greatest* bisimulation on $\mathcal{P}$.

### Image-Finiteness

A relation $\rho$ is *image-finite* if, for each $P$, the image-set $\{Q: P\rho Q\}$ is finite. If all transition relations are image-finite, then there is a convenient alternative characterisation of $\sim$. To see this, define a sequence of relations $\sim_n$ inductively by

$$\sim_0 = \mathcal{P} \times \mathcal{P} \quad \text{(the largest relation on } \mathcal{P}\text{)}$$
$$\sim_{n+1} = F(\sim_n),$$

and then

$$\sim_\omega = \bigcap\{\sim_n : n \geq 0\}.$$

This gives a decreasing sequence

$$\sim_0 \supseteq \sim_1 \supseteq \cdots \supseteq \sim_n \supseteq \cdots\cdots \supseteq \sim_\omega,$$

and it turns out that

- *if each transition relation* $@ > \alpha >>$ *is image-finite, then*

$$P \sim Q \quad \text{iff} \quad P \sim_\omega Q.$$

### Logical Equivalence

The formulae of the present language can express many natural properties concerning transitions. Thus $P \models [\alpha]\bot$ when $P$ is unable to carry out action $\alpha$, $P \models [\alpha]{<}\beta{>}\top$ when any performance of $\alpha$ by $P$ will yield a process that can perform $\beta$, etc. In fact the language is sufficiently strong that the properties it expresses can characterise bisimulation equivalence.

Processes $P$ and $Q$ are *logically equivalent*, $P \equiv Q$, iff they satisfy the same formulae from $Fma(\mathcal{A})$:

$$P \equiv Q \quad \text{iff} \quad \text{for all } A \in Fma(\mathcal{A}),\ P \models A \quad \text{iff} \quad Q \models A.$$

The equivalence relation $\equiv$ contains every bisimulation $\rho$, and hence

$$P \sim Q \quad \text{implies} \quad P \equiv Q.$$

To show this it suffices to prove that

$$\text{for all } P \text{ and } Q, \quad P\rho Q \quad \text{implies} \quad (P \models A \quad \text{iff} \quad Q \models A).$$

This is established by structural induction on the formation of $A$, with the back-and-forth clauses for $\rho$ being used to prove that if the result holds for $A$ then it holds for $[\alpha]A$.

### The Characterisation

A logical characterisation of bisimulation can now be given:

- *if each transition relation* $@ > \alpha >>$ *is image-finite, then*

$$P \sim Q \quad \text{iff} \quad P \equiv Q.$$

The proof of this focuses on the set $Fma(\mathcal{A})^n$ of all formulae whose *modal degree* (depth of nesting of modalities) is at most $n$. Let

$$P \equiv_n Q \quad \text{iff} \quad \text{for all } A \in Fma(\mathcal{A})^n,\ P \models A \quad \text{iff} \quad Q \models A.$$

Then it is shown that in general,

$$P \sim_n Q \quad \text{iff} \quad P \equiv_n Q.$$

Since

$$P \equiv Q \quad \text{iff} \quad P \equiv_n Q \text{ for all } n \geq 0,$$

it follows that

$$P \equiv Q \quad \text{iff} \quad P \sim_\omega Q.$$

But in the image-finite case

$$P \sim_\omega Q \quad \text{iff} \quad P \sim Q.$$

### Infinitary Version

In the absence of image-finiteness, bisimulation equivalence can still be logically characterised if the syntax of the modal language is extended to allow the conjunction of an arbitrary set (possibly infinite) of formulae. The generating definition becomes

$$A ::= \top \mid \neg A \mid \bigwedge_{i \in I} A_i \mid [\alpha]A,$$

where $I$ is an arbitrary indexing set (in practice $I$ need be no larger in cardinality than $\mathcal{P}$, as will emerge below). The new semantic clause is

$$P \models \bigwedge_{i \in I} A_i \quad \text{iff} \quad P \models A_i \text{ for all } i \in I.$$

Let $Fma(\mathcal{A})^\infty$ be the resulting class of formulae, and $\equiv_\infty$ the relation of logical equivalence it induces between processes. Then

- *in any labelled transition system,*

$$P \sim Q \quad \text{iff} \quad P \equiv_\infty Q.$$

The proof that $\sim$ is contained in $\equiv_\infty$ is similar the one showing it is contained in $\equiv$. For the converse, it is enough to show that $\equiv_\infty$ is a bisimulation, for it will then be contained in the greatest bisimulation $\sim$.

So, suppose that $P \equiv_\infty Q$, with a view to establishing the back and forth conditions for $P$ and $Q$ relative to $\equiv_\infty$. For the *back* condition, let $Q@ > \alpha >> Q'$, and let $\{P_i : i \in I\}$ be an indexing of the image-set $\{P' : P@ > \alpha >> P'\}$. Now if there were no $P_i$ with $P@ > \alpha >> P_i \equiv_\infty Q'$, then for each $i$ there would be a formula satisfied at one of $P_i$ and $Q'$, but not at the other. Since formulae can be negated, we can assume there is some $A_i$ with $Q' \models A_i$ and $P_i \not\models A_i$. Let

$$A = \bigwedge_{i \in I} A_i.$$

Then $Q' \models A$, and so $Q \models <\alpha>A$ because $Q@ > \alpha >> Q'$. Hence $P \models <\alpha>A$, as $P \equiv_\infty Q$. But $P_i \not\models A$ for all $i \in I$, so there is no $\alpha$-transition from $P$ to a process satisfying $A$, whence $P \not\models <\alpha>A$. This contradiction forces us to conclude that $P@ > \alpha >> P_i \equiv_\infty Q'$ for some $i$.

The proof of the *forth* condition for $\equiv_\infty$ is similar.

### Further Study

This has been a brief introduction to the role of Hennessy-Milner logic. Process algebra is by now an extensively developed subject, with many concepts and constructions for building processes, and many important variations on the notion of observational equivalence relation, for instance taking into account the presence of silent steps. Here are some suitable sources, beginning with the seminal work on the subject.

Matthew Hennessy and Robin Milner, *Algebraic Laws for Nondeterminism and Concurrency*, J. Assoc. Comput. Mach. **32** (1985), 137–161.

Robin Milner, *Communication and Concurrency*, Prentice Hall, 1989.

Robin Milner, *Operational and Algebraic Semantics of Concurrent Processes*, in Handbook of Theoretical Computer Science, vol. B, J. van Leeuwen (ed.), Elsevier/The MIT Press, 1990, pp. 1201–1242.

Colin Stirling, *Modal Logics for Communicating Systems*, Theoretical Computer Science 49 (1987), 311–347.

## 5 Modal Mu-Calculus

Computer Science abounds with concepts and objects that are defined recursively, or self-referentially. Many of these have an elegant formulation as special *fixed points* of certain operations. Two examples of this will now be given, and then the general theory will be used to develop a powerful modal logic that incorporates all of the other systems that we have looked at so far.

### Transitive Closure

Suppose $R \subseteq S^2$. The transitive closure of $R$ is the relation $R^+$ defined by putting $sR^+t$ iff for some $n \geq 1$ there is a sequence

$$s = s_0 R s_1 R \ldots R s_n = t.$$

$R^+$ is transitive and contains $R$, and is the *least* such relation, in the sense that

$$R^+ = \bigcap\{\rho \subseteq S^2 : \rho \text{ is transitive and } R \subseteq \rho\}.$$

Now a relation $\rho$ is transitive just when $\rho \circ \rho \subseteq \rho$, so being a transitive relation containing $R$ amounts to having

(i)
$$R \cup (\rho \circ \rho) \subseteq \rho.$$

Defining a function $F$ on subsets of $S^2$ by

$$F(\rho) = R \cup (\rho \circ \rho),$$

(i) becomes

$$F(\rho) \subseteq \rho,$$

so

(ii)
$$R^+ = \bigcap\{\rho \subseteq S^2 : F(\rho) \subseteq \rho\}.$$

In fact $R^+$ is a *fixed point* of $F$, meaning that $F(R^+) = R^+$. From (ii) it follows that it is the least such fixed point: if $F(\rho) = \rho$ then $R^+ \subseteq \rho$.

**Bisimulations as Fixed Points**

In Section 4 we saw that a relation $\rho$ on a transition system $\mathcal{P}$ is a bisimulation iff $\rho \subseteq F(\rho)$, where $F$ is a certain function on subsets of $\mathcal{P}^2$. The bisimulation-equivalence relation $\sim$ satisfies

(iii)
$$\sim = \bigcup\{\rho \subseteq S^2 : \rho \subseteq F(\rho)\},$$

and in fact $F(\sim) = \sim$, so by (iii) $F$ is the *greatest* fixed point of $F$: if $F(\rho) = \rho$ then $\rho \subseteq \sim$.

**Monotone Operators**

A function $F : 2^S \to 2^S$, defined on the powerset of a set $S$, will be called an *operator on* $S$. A subset $T$ of $S$ is

- a *fixed point* if $F(T) = T$,
- a *pre-fixed point* if $F(T) \subseteq T$,
- a *post-fixed point* if $T \subseteq F(T)$.

$F$ is *monotone* if

$$T \subseteq T' \quad \text{implies} \quad F(T) \subseteq F(T').$$

It is this monotonicity that accounts for the preceding phenomena:

**Knaster-Tarski Theorem**

*If $F$ is a monotone operator on a set $S$, then*

- *$F$ has a least fixed point, namely the intersection*
$$\mu F = \bigcap\{T \subseteq S : F(T) \subseteq T\}$$
*of all pre-fixed points of $F$; and*
- *$F$ has a greatest fixed point, namely the union*
$$\nu F = \bigcup\{T \subseteq S : T \subseteq F(T)\}$$

*of all post-fixed points of $F$.*

Thus $\mu F$ and $\nu F$ are solutions of the equation

$$T = F(T),$$

and all such solutions satisfy

$$\mu F \subseteq T \subseteq \nu F.$$

Sometimes it is possible to construct $\mu F$ by starting with the least element $\emptyset$ of $2^S$ and applying $F$ iteratively to generate the increasing (by monotonicity) sequence

$$\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \subseteq \cdots\cdots\cdots$$

and then obtain $\mu F$ as the *union* of the sequence. The members of the sequence can be thought of as approximations to $\mu F$ "from below".

This works for instance with transitive closure, ultimately because $R^+$ is the union of the relations

$$R, R \circ R, R \circ R \circ R, R \circ R \circ R \circ R, \ldots\ldots\ldots$$

In other cases the iteration must continue into the transfinite to reach $\mu F$ from below in this way.

Dually, starting with the greatest element $S$ of $2^S$ produces

$$S \supseteq F(S) \supseteq F(F(S)) \supseteq \cdots\cdots\cdots$$

and for some operators $\nu F$ is the *intersection* of this decreasing sequence of approximations "from above". In Section 4 we saw that this works for the bisimulation $\sim$, but only under the constraint of image-finiteness.

**Formulae as Operators**

Let $A$ be a formula containing an atomic formula $p$. In a model $\mathcal{M} = (S, .., V)$, $A$ has a denotation $V(A)$ which is a subset of $S$, and which depends on the denotations of the atomic formulae in $A$. Now regard $p$ as a *variable* ranging over all subsets of $S$. As the denotation of $p$ varies, so to will the denotation of $A$, and in this way $A$ induces an operator on $S$. For certain $A$ this operator will be monotonic, typically when $A$ has no negations $\neg$, or when each of the atomic variables of $A$ occur only within the scope of an even number of negations.

We need a notation for describing the process of changing the valuation of a variable. Given a valuation $V$, let

$$V_{p:=T}$$

be the valuation that differs from $V$ only in that it assigns the subset $T$ of $S$ to $p$. Thus

$$V_{p:=T}(q) = \begin{cases} V(q) & \text{if } q \neq p \\ T & \text{if } q = p. \end{cases}$$

Then the operator induced by $A$ relative to $V$ as $p$ varies is the function

$$T \mapsto V_{p:=T}(A).$$

A fixed point of this operator might be though of as a solution of the syntactic equation

$$\text{“ p = A ”}.$$

**Syntax for Mu-Calculus**

| | |
|---|---|
| Formula variables: | $p \in \Phi$ |
| Action labels: | $\alpha \in \mathcal{A}$ |
| Formulae: | $A \in Fma(\Phi, \mathcal{A})$ |

$$A ::= p \mid \bot \mid \top \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid [\alpha]A \mid <\alpha>A \mid \mu p.A \mid \nu p.A$$

$\mu p.A$ and $\nu p.A$ are intended to denote the least and greatest solutions of the equation $p = A$. The operations $\mu p.$ and $\nu p.$ function like quantifiers, binding free occurrences of $p$ in $A$. Note that the syntax is negation-free.

**Semantics**

Let $\mathcal{M} = (S, \{@ > \alpha >>: \alpha \in \mathcal{A}\}, V)$ be a model on an $\mathcal{A}$-labelled transition system. The valuation $V: \Phi \to 2^S$ extends to give a denotation $V(A)$ to all formulae by the equations (cf. page 16):

$$\begin{aligned}
V(\bot) &= \emptyset \\
V(\top) &= S \\
V(A_1 \wedge A_2) &= V(A_1) \cap V(A_2) \\
V(A_1 \vee A_2) &= V(A_1) \cup V(A_2) \\
V([\alpha]A) &= \{s\text{: for all } t \in S,\ s@ > \alpha >> t \text{ implies } t \in V(A)\} \\
V(<\alpha>A) &= \{s\text{: for some } t \in S,\ s@ > \alpha >> t \text{ and } t \in V(A)\} \\
V(\mu p.A) &= \bigcap\{T \subseteq S\colon V_{p:=T}(A) \subseteq T\} \\
V(\nu p.A) &= \bigcup\{T \subseteq S\colon T \subseteq V_{p:=T}(A)\}.
\end{aligned}$$

Because of the absence of negation, the operator $T \mapsto V_{p:=T}(A)$ is always monotone, so this semantics does indeed give the desired fixed point denotation to $\mu p.A$ and $\nu p.A$. The theory can be extended to allow negation, but then there has to be a restriction on the formation of $\mu p.A$ and $\nu p.A$ to gaurantee monotonicity. On the other hand, with negation available fewer connectives are needed, since $\nu p.A$ can be defined as

$$\neg\mu p.\neg A[\neg p/p],$$

where $A[\neg p/p]$ is the result of replacing (free) p by $\neg p$ in $A$. For instance $\neg\mu p.[\alpha]p$ has the same meaning as $\nu p.<\alpha>p$.

**Understanding Fixed Point Formulae**

To get a sense of the meaning of $\mu p.A$ and $\nu p.A$ for particular $A$, think of them informally as solutions of the equation $p = A$, so that we can replace occurrences of $p$ in $A$ by $A$ itself, and keep on "unfolding" the equation in this way.

For instance, consider the equation $p = (A \vee <\alpha>p)$. The unfolding produces

$$A \vee <\alpha>(A \vee <\alpha>(A \vee <\alpha>(A \vee \cdots\cdots\cdots$$

Stopping at a finite stage gives a formula

$$A \vee <\alpha>(A \vee <\alpha>(A \vee \cdots\cdots \vee <\alpha>A)) \cdots)$$

which asserts that there is a finite $\alpha$-path leading to a state in which $A$ is true.

The meaning of the least fixed point here is given by asking that some finite-stage unfolding be true, i.e. that there is some finite $\alpha$-path ending with $A$ true. But this assertion corresponds to the dynamic logic formula $<\alpha^*>A$ ! So that formula is equivalent to $\mu p.(A \vee <\alpha>p)$.

The formula $\nu p.(A \wedge [\alpha]p)$ has the unfolding

$$A \wedge [\alpha](A \wedge [\alpha](A \wedge [\alpha](A \wedge \cdots\cdots\cdots$$

To get the meaning of a greatest fixed point we must allow the possibility of the unfolding running forever, so here we are saying that all finite $\alpha$-paths must have $A$ true at their end-points, an assertion which is expressed by the $PDL$-formula $[\alpha^*]A$, dual to $<\alpha^*>A$.

For another example, consider the equation $p = A \wedge <\alpha>p$. The least fixed point is just $\emptyset$, reflecting the fact that $A \wedge <\alpha>\perp$ is equivalent to $\perp$. The relevant unfolding is

$$A \wedge <\alpha>(A \wedge <\alpha>(A \wedge <\alpha>(A \wedge \cdots\cdots\cdots$$

Allowing the unfolding to run forever indicates that the property expressed by $\nu p.A \wedge <\alpha>p$ is that there exists some infinite $\alpha$-path with $A$ true at every point.

Taking the case that $A$ is $\top$ here, it follows that the meaning of $\nu p.<\alpha>p$ is that there exists an infinite $\alpha$-path starting from the present state. But this is the same as the meaning of the formula **repeat**($\alpha$) from Section 2.

**Translation of PDL**

$PDL$ can be translated into the Mu-Calculus, because $[\alpha^*]A$ can be identified with $\nu p.(A \wedge [\alpha]p)$, and the other $PDL$-modalities can be reduced by the equivalences

$$\begin{aligned} &[A?]B \leftrightarrow (A \rightarrow B), \\ &[\alpha;\beta]A \leftrightarrow [\alpha][\beta]A, \\ &[\alpha \cup \beta]A \leftrightarrow [\alpha]A \wedge [\beta]A. \end{aligned}$$

Formally this is done by the translation $A \mapsto A^{\#}$, where

$$\begin{aligned} p^{\#} &= p \\ \perp^{\#} &= \perp \\ (A_1 \rightarrow A_2)^{\#} &= A_1^{\#} \rightarrow A_2^{\#} \\ ([\pi]A)^{\#} &= [\pi]A^{\#} \\ ([A?]B)^{\#} &= A^{\#} \rightarrow B^{\#} \\ ([\alpha_1;\alpha_2]A)^{\#} &= ([\alpha_1][\alpha_2]A)^{\#} \\ ([\alpha \cup \beta]A)^{\#} &= ([\alpha]A)^{\#} \wedge ([\beta]A)^{\#} \\ ([\alpha^*]A)^{\#} &= \nu p.(A^{\#} \wedge ([\alpha]p)^{\#}) \end{aligned}$$

Furthermore, putting

$$\mathbf{repeat}(\alpha)^{\#} = \nu p.<\alpha>p$$

translates the logic $RPDL$ into the Mu-Calculus. Since the formulae **loop**($\alpha$) are definable in $RPDL$, this leads also to a translation of $LPDL$.

**Translating CTL**

A *CTL*-model can be viewed as a Mu-Calculus model with a single transition relation @ > $\alpha$ >>. Then the following pairs of formulae have the same meaning, indicating that there is a translation of *CTL* into the Mu-Calculus.

| | |
|---|---|
| $\forall[\mathrm{X}]A$ | $[\alpha]A$ |
| $\exists[\mathrm{X}]A$ | $<\alpha>A$ |
| $\exists(A\mathcal{U}B)$ | $\mu p.B \vee (A \wedge <\alpha>p)$ |
| $\forall(A\mathcal{U}B)$ | $\mu p.B \vee (A \wedge [\alpha]p)$ |

(cf. the *CTL*-axioms $\exists\mathcal{U}$ and $\forall\mathcal{U}$ on page 28).

There is also a translation of *CTL**, too complex to reproduce here (cf. the paper by Dam cited below). The formula

$$\mu p.A \wedge [\alpha][\alpha]p$$

expresses "along all branches $A$ holds at every second state", a property that is inexpressible in *CTL**.

**The Status of the Mu-Calculus**

It is known that the Mu-Calculus has the finite model property, and is decidable in non-deterministic exponential time. An axiomatisation has been given that uses an infinitary inference rule, but the existence of a finitary deductive system remains unresolved. The semantics validates the schema

$$A[\mu p.A/p] \to \mu p.A$$

and is sound for the rule

$$\frac{A[B/p] \to B}{\mu p.A \to B}$$

The schema expresses that $\mu p.A$ is a pre-fixed point of the operator defined by $A$, and the rule expresses that it is the least such pre-fixed point.

It is natural to conjecture that the Mu-Calculus is the smallest normal logic to contain this rule and schema and the corresponding ones for $\nu$.

**References**

Julian Charles Bradfield, *Verifying Temporal Properties of Systems*, Birkhäuser, 1992.

Mads Dam, *CTL* and ECTL* as Fragments of the Propositional $\mu$-Calculus*, Theoretical Computer Science **126** (1994), 77–96.

Dexter Kozen, *Results on the Propositional $\mu$-Calculus*, Theoretical Computer Science **27** (1983), 333–354.

Dexter Kozen, *A Finite Model Theorem for the Propositional $\mu$-Calculus*, Studia Logica **47** (1988), 233–241.

## 6 Canonical Models and Filtrations

Let $\Lambda$ be a normal logic in a multimodal language $Fma_I(\Phi)$ as in Section 1. To prove that $\Lambda$ is complete with respect to a certain class of models, it has to be shown that each non-theorem of $\Lambda$ can be falsified by some model in this class. To begin with we will construct a single *canonical* $\Lambda$-model

$$\mathcal{M}^\Lambda = (S^\Lambda, \{R_i^\Lambda : i \in I\}, V^\Lambda),$$

that falsifies every non-theorem of $\Lambda$. This may not however be a model of the intended type – for instance it may be infinite, so will not establish the finite model property – and hence its construction may only be an intermediate step in obtaining the desired completeness proof.

The basic idea of $\mathcal{M}^\Lambda$ is this. In any model $\mathcal{M}$, each point $s$ is associated with the set

$$\Gamma_s = \{A \in Fma_I(\Phi) : \mathcal{M} \models_s A\}$$

of all formulae true at $s$. The properties of membership of $\Gamma_s$ reflect the properties of the truth relation $\mathcal{M} \models_s A$, and sets of the form $\Gamma_s$ can be characterised in a syntactic way. The members of $S^\Lambda$ will thus be *defined* as certain sets of formulae whose properties are determined by the logic $\Lambda$. Moreover, in a model $\mathcal{M}$, if $sR_it$ then $[i]A \in \Gamma_s$ implies $A \in \Gamma_t$, so

$$\{A \in Fma_I(\Phi) : [i]A \in \Gamma_s\} \subseteq \Gamma_t,$$

and this relationship will be used to define the relation $R_i^\Lambda$ in $\mathcal{M}^\Lambda$.

**$\Lambda$-Maximal Sets**

A set $\Gamma \subseteq Fma(\Phi)$ is *$\Lambda$-inconsistent* if there are finitely many formulae $B_1, \dots, B_n$ in $\Gamma$ such that

$$\vdash_\Lambda B_1 \wedge \cdots \wedge B_n \to \bot.$$

It this does not hold, $\Gamma$ is *$\Lambda$-consistent*. $\Gamma$ is defined to be *$\Lambda$-maximal* if

- $\Gamma$ is $\Lambda$-consistent, and
- for any $A \in Fma_I(\Phi)$, either $A \in \Gamma$ or $\neg A \in \Gamma$.

Put

$$S^\Lambda = \{\Gamma \subseteq Fma_I(\Phi) : \Gamma \text{ is } \Lambda\text{-maximal}\}.$$

**Properties of $\Lambda$-Maximal Sets**

If $\Gamma$ is $\Lambda$-maximal, then:

- If $A \notin \Gamma$, then $\Gamma \cup \{A\}$ is not $\Lambda$-consistent. Hence if $\Gamma \subseteq \Delta$ and $\Delta$ is $\Lambda$-consistent, then $\Gamma = \Delta$ (this explains the use of the adjective "maximal").
- $\Lambda \subseteq \Gamma$.
- $\bot \notin \Gamma$.
- For any formula $A$, *exactly one* of $A$ and $\neg A$ belongs to $\Gamma$, i.e.,

$$\neg A \in \Gamma \quad \text{iff} \quad A \notin \Gamma.$$

- $(A \to B) \in \Gamma$ iff ($A \in \Gamma$ implies $B \in \Gamma$).
- $A \wedge B \in \Gamma$ iff $A, B \in \Gamma$.
- $A \vee B \in \Gamma$ iff $A \in \Gamma$ or $B \in \Gamma$.
- $(A \leftrightarrow B) \in \Gamma$ iff ($A \in \Gamma$ iff $B \in \Gamma$).

**Lindenbaum's Lemma**

*Every $\Lambda$-consistent set of formulae $\Gamma$ is contained in a $\Lambda$-maximal set $\Delta$.*

*Proof.* Assume $\Phi$ and $I$ are countable, so that $Fma_I(\Phi)$ can be enumerated

$$A_0, A_1, \dots, A_n, \dots\dots\dots$$

Put

$$\Delta_0 = \Gamma$$
$$\Delta_{n+1} = \begin{cases} \Delta_n \cup \{A_n\}, & \text{if this is } \Lambda\text{-consistent} \\ \Delta_n \cup \{\neg A_n\}, & \text{otherwise} \end{cases}$$

and then

$$\Delta = \bigcup\{\Delta_n : n \geq 0\}.$$

**Corollary**

$$\vdash_\Lambda A \quad \text{iff} \quad \text{for all } s \in S^\Lambda,\ A \in s.$$

*Proof.*
If $\nvdash_\Lambda A$ then $\{\neg A\}$ is $\Lambda$-consistent, so by Lindenbaum's Lemma there exists $s \in S^\Lambda$ with $\{\neg A\} \subseteq s$, hence $A \notin s$.

**Box-Lemma**

*Define the relation $R_i^\Lambda$ on $S^\Lambda$ by*

$$sR_i^\Lambda t \quad \textit{iff} \quad \{A \in Fma_I(\Phi) : [i]A \in s\} \subseteq t.$$

*Then for any $s \in S^\Lambda$, and any $B \in Fma(\Phi)$,*

$$[i]B \in s \quad \textit{iff} \quad \textit{for all } t \in S^\Lambda,\ sR^\Lambda t \ \textit{implies} \ B \in t.$$

*Proof.*
From left to right holds by definition of $R_i^\Lambda$. For the converse, suppose $[i]B \notin s$. We have to construct a $t \in S^\Lambda$ with $B \notin t$. Let

$$\Gamma = \{A \in Fma_I(\Phi) : [i]A \in s\}.$$

If $\Gamma \cup \{\neg B\}$ were not $\Lambda$-consistent then there would exist formulae $A_1, \ldots, A_n \in \Gamma$ such that

$$\vdash_\Lambda A_1 \wedge \ldots \wedge A_n \wedge \neg B \to \bot,$$

and hence

$$\vdash_\Lambda A_1 \wedge \ldots \wedge A_n \to B.$$

But then as $\Lambda$ is normal,

$$\vdash_\Lambda [i]A_1 \wedge \ldots \wedge [i]A_n \to [i]B$$

(cf. page 21). But $[i]A_1, \ldots, [i]A_n \in s$, and so this would imply $[i]B \in s$, contrary to hypothesis.

It follows by Lindenbaum's Lemma that there exists a $t \in S^\Lambda$ with $\Gamma \cup \{\neg B\} \subseteq t$. But then $\Gamma \subseteq t$ makes $sR_i^\Lambda t$, and $\neg B \in t$ forces $B \notin t$, as desired.

The definition of $\mathcal{M}^\Lambda$ is completed by defining

$$V^\Lambda(p) = \{s \in S^\Lambda : p \in s\}.$$

Then the Box Lemma and the earlier listed properties of maximal sets lead to the following result.

**Truth Lemma**
*For any $A \in Fma_I(\Phi)$ and any $s \in S^\Lambda$,*

$$\mathcal{M}^\Lambda \models_s A \quad \textit{iff} \quad A \in s.$$

Since in general $\vdash_\Lambda A$ iff $A$ belongs to all $\Lambda$-maximal sets, this yields

$$\vdash_\Lambda A \quad \text{iff} \quad \mathcal{M}^\Lambda \models A,$$

showing that the canonical $\Lambda$-model determines the logic $\Lambda$. In particular, the smallest normal logic $K_I$ is determined by its canonical model, and hence by the class of all models for the given language.

**Properties of $\mathcal{M}^\Lambda$**
If the logic $\Lambda$ contains the schema

(T) $$[i]A \to A,$$

then all instances of it belong to any $\Lambda$-maximal set $s \in S^\Lambda$. Consequently

$$\{A \in Fma_I(\Phi) : [i]A \in s\} \subseteq s,$$

making $sR_i^\Lambda s$, so $R_i^\Lambda$ is reflexive. In this way it is seen that the smallest normal logic containing schema (T) is determined by its reflexive canonical model, and hence by the class of models in which $R_i$ is reflexive.

Similarly, the schema

$$[i]A \to [i][i]A$$

forces $R_i^\Lambda$ to be transitive,

$$A \to [i] <i> A$$

makes it symmetric, and

$$[i]\neg A \leftrightarrow \neg[i]A$$

makes it a total function on $S^\Lambda$.

This technique provides a general method for completeness proofs, by connecting the proof-theoretic properties of $\Lambda$ (derivability of various schemata) to the model-theoretic properties of $\mathcal{M}^\Lambda$.

## Filtrations

We have seen that if $\nvdash_\Lambda A$ then there will be some point in the canonical model $\mathcal{M}^\Lambda$ at which $A$ is false. But in its capacity as a falsifying model for a *particular* non-theorem $A$, $\mathcal{M}^\Lambda$ provides a good deal of superfluous information. To calculate the truth-value of $A$ at points in $\mathcal{M}^\Lambda$, we need only know the truth-values in $\mathcal{M}^\Lambda$ of the members of the set $Sf(A)$ of subformulae of $A$, whereas $\mathcal{M}^\Lambda$ provides truth-values for all formulae whatsoever. Moreover, if $\Phi$ and/or $I$ is infinite, then $S^\Lambda$ will be infinite (in fact uncountable), and so a point of $\mathcal{M}^\Lambda$ will in general be indistinguishable from many other points as to how it treats the *finitely* many members of $Sf(A)$. Thus we many as well identify points that assign the same truth-values to all members of $Sf(A)$. The identification process allows us to collapse $\mathcal{M}^\Lambda$ to form a new falsifying model for $A$, one that has room for variety in its definition. This process, known as *filtration*, gives a way of proving certain technical results (finite model property, decidability) about certain logics $\Lambda$. It also gives a new way of constructing models that comes into its own in cases where $\mathcal{M}^\Lambda$ is not of the desired type for a completeness theorem.

Here, for the record, is a formal definition of $Sf(A)$:

$$\begin{aligned} Sf(p) &= \{p\} \\ Sf(\bot) &= \{\bot\} \\ Sf(A_1 \to A_2) &= \{A_1 \to A_2\} \cup Sf(A_1) \cup Sf(A_2) \\ Sf([i]A) &= \{[i]A\} \cup Sf(A). \end{aligned}$$

Now fix a logic $\Lambda$ and a set $\Gamma \subseteq Fma(\Phi)$ that is closed under subformulae, i.e.

$$A \in \Gamma \quad \text{implies} \quad Sf(A) \subseteq \Gamma.$$

Define an equivalence relation $\sim_\Gamma$ on $S^\Lambda$ by

$$\begin{aligned} s \sim_\Gamma t \quad &\text{iff} \quad s \cap \Gamma = t \cap \Gamma \\ &\text{iff} \quad \text{for all } A \in \Gamma,\ A \in s \text{ iff } A \in t. \end{aligned}$$

Let

$$|s| = \{t \in S : s \sim_\Gamma t\}$$

be the $\sim_\Gamma$-equivalence class of $s$, and define

$$S_\Gamma = \{|s| : s \in S\}$$

to be the set of all such equivalence classes. Then

$$|s| = |t| \quad \text{iff} \quad s \cap \Gamma = t \cap \Gamma,$$

and so the map $|s| \mapsto s \cap \Gamma$ is a well-defined injection of $S_\Gamma$ into the powerset of $\Gamma$. In particular,

- *if $\Gamma$ is finite, then $S_\Gamma$ is finite and has at most $2^n$ elements, where $n$ is the number of elements of $\Gamma$.*

Now let $I_\Gamma = \{i \in I\colon [i] \text{ occurs in } \Gamma\}$, and let $\Phi_\Gamma = \Phi \cap \Gamma$ be the set of atomic formulae that belong to $\Gamma$. Define

$$V_\Gamma : \Phi_\Gamma \to 2^{S_\Gamma}$$

by putting

$$|s| \in V_\Gamma(p) \quad \text{iff} \quad p \in s$$

whenever $p \in \Phi_\Gamma$ (since then $p \in \Gamma$, $V_\Gamma$ is well-defined).

We are going to consider $\Phi_\Gamma$-models of the form

$$\mathcal{M} = (S_\Gamma, \{R_i\colon i \in I_\Gamma\}, V_\Gamma)$$

with the property that the truth-values of members of $\Gamma$ in $\mathcal{M}$ and in $\mathcal{M}^\Lambda$ are left invariant by the correspondence $s \mapsto |s|$. Reflection on what is required to make this work leads to the following definition.

A binary relation $R_i$ on $S_\Gamma$ is called a *$\Gamma$-filtration of $R_i^\Lambda$* if it satisfies

(F1) if $sR_i^\Lambda t$, then $|s|R_i|t|$; and

(F2) if $|s|R_i|t|$, then $\{A : [i]A \in s \cap \Gamma\} \subseteq t$.

Any $\Phi_\Gamma$-model $\mathcal{M} = (S_\Gamma, \{R_i\colon i \in I_\Gamma\}, V_\Gamma)$ in which each $R_i$ satisfies F1 and F2 is called a *$\Gamma$-filtration of the model $\mathcal{M}^\Lambda$*. The crucial property of such an $\mathcal{M}$ is the

**Filtration Lemma**

*If $A \in \Gamma$, then for any $s \in S^\Lambda$,*

$$\mathcal{M} \models_{|s|} A \quad \textit{iff} \quad A \in s.$$

**Existence of Filtrations**

There are always at least two:

- The *smallest* filtration $R_i^\sigma$ is given by

$$|s| R_i^\sigma |t| \quad \text{iff} \quad \exists s' \in |s| \, \exists t' \in |t| \, (s' R_i^\Lambda t').$$

- The *largest* filtration $R_i^\lambda$ has

$$|s| R_i^\lambda |t| \quad \text{iff} \quad \{A : [i]A \in s \cap \Gamma\} \subseteq t.$$

Any other filtration $R_i$ of $R_i^\Lambda$ has

$$R_i^\sigma \subseteq R_i \subseteq R_i^\lambda.$$

**Finite Model Property for $K_I$**

Suppose $\nvdash_{K_I} A$. Then there is a point $s$ in the canonical $K_I$-model at which $A$ is false , i.e. $A \notin s$. Let $\Gamma = Sf(A)$. Then $\Gamma$ is closed under subformulae, so we can construct $\Gamma$-filtrations $\mathcal{M}$ of $\mathcal{M}^{K_I}$ as above. By the Filtration Lemma, $A$ is false at $|s|$ in any such model. Moroever, $\mathcal{M}$ has at most $2^n$ elements, where $n$ is the number of subformulae of $A$.

**Extending the Method**

To apply this technique to a logics $\Lambda$ containing $K_I$, it is necessary to prove that the filtration $\mathcal{M}$ is a $\Lambda$-model, i.e. $\mathcal{M} \models \Lambda$.

For example, if $R_i^\Lambda$ is reflexive, then the filtration condition F1 guarantees that $R_i$ is reflexive in $\mathcal{M}$, whence $\mathcal{M} \models [i]A \rightarrow A$. This leads to a proof of the finite model property for the smallest normal logic that contains the schema T.

Other conditions are not so readily preserved in passing from the canonical model to a filtration, and require more careful analysis, or further modification of the quotient model. In the next section the method will be adapted to the context of linear temporal logic, and for this application the following important feature of filtrations will be needed.

**Definability Lemma**

*If $\Gamma$ is finite, then for any subset $X \subseteq S_\Gamma$ there is a formula $A_X$ such that for all $s \in S^\Lambda$,*

$$A_X \in s \quad \textit{iff} \quad |s| \in X.$$

*Proof.*

For each $t \in S^\Lambda$ let $A_t$ be the conjunction of the members of

$$\{A \in \Gamma : A \in t\} \cup \{\neg A : A \in \Gamma \;\&\; A \notin t\}.$$

Then

$$A_t \in s \quad \text{iff} \quad s \cap \Gamma = t \cap \Gamma \quad \text{iff} \quad |s| = |t|.$$

Now if

$$X = \{|t_1|, \ldots, |t_n|\},$$

put

$$A_X = A_{t_1} \vee \cdots \vee A_{t_n}.$$

## 7 Completeness of Linear Temporal Logic

Recall from Section 3 that propositional linear temporal logic has the syntax

| | |
|---|---|
| Atomic formulae: | $p \in \Phi$ |
| Formulae: | $A \in Fma(\Phi)$ |

$$A ::= p \mid \bot \mid A_1 \rightarrow A_2 \mid [\mathrm{X}]A \mid A_1 \mathcal{U} A_2,$$

with $<\mathrm{F}> A = \top \mathcal{U} A$, and $[\mathrm{F}] = \neg <\mathrm{F}> \neg A$. *PLTL* is defined to be the set of all formulae in this language that are true in all models on state sequences.

Let $\Lambda$ be the smallest $[\mathrm{X}]$-normal logic in $Fma(\Phi)$ that contains the schemata

| | |
|---|---|
| *Fun*: | $[\mathrm{X}]\neg A \leftrightarrow \neg[\mathrm{X}]A$ |
| $\mathcal{U}1$: | $A\mathcal{U}B \rightarrow <\mathrm{F}> B$ |
| $\mathcal{U}2$: | $A\mathcal{U}B \leftrightarrow B \vee (A \wedge [\mathrm{X}](A\mathcal{U}B))$ |

and is closed under the *<F>-Rule*

$$\textit{if} \ \vdash A \rightarrow B \quad \textit{then} \ \vdash <F> A \rightarrow <F> B,$$

and the *Induction Rule*

$$\textit{if} \ \vdash A \rightarrow [X]A \quad \textit{then} \ \vdash A \rightarrow [F]A.$$

It will now be shown that $\Lambda = PLTL$. That $\Lambda \subseteq PLTL$ (soundness) is straightforward: all $\Lambda$-axioms are true in models on state sequences, and the rules of Necessitation for $[\mathrm{X}]$ and Induction, and the $<\mathrm{F}>$-Rule, all preserve this property. Hence *PLTL* is a logic that contains these schemata and is closed under these rules, so it contains the smallest such logic $\Lambda$.

For the converse, fix a formula $D \notin \Lambda$, with the objective of showing $D$ is falsifiable on some state sequence model. This involves a process of $\Gamma$-filtration, in which $\Gamma$ will need to contain more than just the subformulae of $D$. Its required closure properties are:

$D \in \Gamma$;

$\Gamma$ is closed under subformulae;

$A\mathcal{U}B \in \Gamma$ implies $[\mathrm{X}](A\mathcal{U}B), <\mathrm{F}> B \in \Gamma$.

Putting

$$\Gamma = Sf(D) \cup \{\top\} \cup \{[\mathrm{X}](A\mathcal{U}B), <\mathrm{F}> B, [\mathrm{X}](\top\mathcal{U}B) : A\mathcal{U}B \in Sf(D)\}$$

provides a finite set with these properties. $\Gamma$ has no more than $4n$ elements, where $n$ is the number of subformulae of $D$.

Let $S^\Lambda$ be the set of $\Lambda$-maximal subsets of $Fma(\Phi)$, and $R_\mathrm{X}$ the relation on $S^\Lambda$ induced by the modality $[\mathrm{X}]$, i.e

$$sR_\mathrm{X}t \quad \text{iff} \quad \{A : [\mathrm{X}]A \in s\} \subseteq t.$$

Define the *finite* quotient set $S_\Gamma$ as in Section 6, and let $R$ be the least $\Gamma$-filtration of $R_\mathrm{X}$, so that for $x, y \in S_\Gamma$,

$$xRy \quad \text{iff} \quad \exists s \in x \, \exists t \in y \, (sR_\mathrm{X}t).$$

### Admissible Sequences

A sequence $\sigma_0, \sigma_1, \ldots, \sigma_j, \ldots\ldots$ of points in $S^\Lambda$ is *admissible* if for all $j \geq 0$,

- $|\sigma_j|R|\sigma_{j+1}|$ in $S_\Gamma$, and
- if $<\mathrm{F}> B \in \sigma_j \cap \Gamma$, then $B \in \sigma_k$ for some $k \geq j$.

### Admissibility Lemma

*If $\sigma$ is an admissible sequence in $S^\Lambda$, then for all $j \geq 0$,*

(1) *if $[X]A \in \Gamma$, then $[X]A \in \sigma_j$ iff $A \in \sigma_{j+1}$;*

(2) *if $A\mathcal{U}B \in \Gamma$, then $A\mathcal{U}B \in \sigma_j$ iff for some $k \geq j$, $B \in \sigma_k$ and for every $i$ such that $j \leq i < k$, $A \in \sigma_i$.*

This Lemma will be proven below. What it states is that for formulae from $\Gamma$, membership of a point in an admissible sequence behaves exactly like truth at that point in a model on the sequence. Thus, defining a model $\mathcal{M} = (\sigma, V)$ by putting

$$V(p) = \{\sigma_j : p \in \sigma_j\},$$

an inductive argument based on the Admissibility Lemma and properties of $\Lambda$-maximal sets establishes the

**Truth Lemma**
*For any $A \in \Gamma$ and any $j \geq 0$, $\mathcal{M} \models_j A$ iff $A \in \sigma_j$.*

The problem then is to show that there are any admissible sequences.

**Sequence Lemma**
*For any $s \in S^\Lambda$, there exists an admissible sequence $\sigma$ starting with $s$.*

This is the deepest part of the argument, and again will be proven below. The result allows us to construct the desired falsifying model for $D$, thereby finishing the completeness theorem: since $D \notin \Lambda$ there exists a $\Lambda$-maximal $s \in S^\Lambda$ with $D \notin s$. Taking an admissible sequence $\sigma$ in $S^\Lambda$ with $\sigma_0 = s$, and the model $\mathcal{M}$ on $\sigma$ as above, it then follows by the Truth Lemma that $\mathcal{M} \not\models_0 D$.

**Proof of the Admissibility Lemma**

(1) Let $[\mathrm{X}]A \in \Gamma$. Now $|\sigma_j|R|\sigma_{j+1}|$ by admissibility, and $R$ is a filtration of $R_\mathrm{X}$, so by filtration condition F2,

$$\{B\colon [\mathrm{X}]B \in \sigma_j \cap \Gamma\} \subseteq \sigma_{j+1}.$$

It is then immediate that $[\mathrm{X}]A \in \sigma_j$ implies $A \in \sigma_{j+1}$. Conversely, suppose $A \in \sigma_{j+1}$. Now by definition of $R$, there exist $s \in |\sigma_j|$ and $t \in |\sigma_{j+1}|$ with $sR_\mathrm{X}t$. Then $A \in t$, as $A \in \Gamma$ and $t \sim_\Gamma \sigma_{j+1}$. Hence $\neg A \notin t$, and so $[\mathrm{X}]\neg A \notin s$ as $sR_\mathrm{X}t$. Thus by axiom *Fun*, $\neg[\mathrm{X}]A \notin s$, so $[\mathrm{X}]A \in s$, which finally yields $[\mathrm{X}]A \in \sigma_j$ because $s \sim_\Gamma \sigma_j$.

(2) Let $A\mathcal{U}B \in \Gamma$. Suppose $A\mathcal{U}B \in \sigma_j$. Then by axiom $\mathcal{U}1$, $<\mathrm{F}>B \in \sigma_j$. But by the closure conditions on $\Gamma$, $<\mathrm{F}>\in \Gamma$, so admissibility entails that $B \in \sigma_k$ for some $k \geq j$. Take the least such $k$. We have then to show that $A \in \sigma_i$ whenever $j \leq i < k$. If $k = j$ there is nothing to prove, so assume $j < k$. Then by definition of $k$, $B \notin \sigma_i$ whenever $j \leq i < k$.

Now it will be shown that $A\mathcal{U}B \in \sigma_i$ for $j \leq i < k$, by induction on $i$. If $i = j$, the result hold by the assumption $A\mathcal{U}B \in \sigma_j$. For the induction step, if $A\mathcal{U}B \in \sigma_i$ and $j < i+1 < k$, then as $B \notin \sigma_i$, axiom $\mathcal{U}2$ yields $[\mathrm{X}](A\mathcal{U}B) \in \sigma_i$. But $[\mathrm{X}](A\mathcal{U}B) \in \Gamma$, so by part (1) of this Lemma, $A\mathcal{U}B \in \sigma_{i+1}$, and the induction argument is finished.

Finally, if $j \leq i < k$ we now have $A\mathcal{U}B \in \sigma_i$ and $B \notin \sigma_i$, from which $\mathcal{U}2$ yields $A \in \sigma_i$ as desired.

Conversely, assume that $B \in \sigma_k$ for some $k \geq j$, with $A \in \sigma_i$ whenever $j \leq i < k$. We then show that $A\mathcal{U}B \in \sigma_i$ whenever $j \leq i \leq k$, by backwards induction on $i$, giving finally $A\mathcal{U}B \in \sigma_j$ to complete the Lemma.

For the case $i = k$, since $B \in \sigma_k$, $A\mathcal{U}B \in \sigma_k$ is immediate by $\mathcal{U}2$. For the induction step, if $A\mathcal{U}B \in \sigma_i$ with $j < i \leq k$, then $[\mathrm{X}](A\mathcal{U}B) \in \sigma_{i-1}$ by part (1) again, and $A \in \sigma_{i-1}$ by assumption, and this is enough to force $A\mathcal{U}B \in \sigma_{i-1}$ by $\mathcal{U}2$ once more.

**Proof of the Sequence Lemma**
This originates in work of Gabbay, Pnueli, Shelah and Stavi. The exposition given here uses the elegant notion of a *scheduler* due to Stirling [1992, §6.3].

For each $x \in S_\Gamma$, let

$$x^+ = \{y \in S_\Gamma\colon xRy\}$$

be the set of all *R-successors* of $x$. Then $x^+$ is non-empty, because $R$ is a total relation on $S_\Gamma$. This follows by the axiom *Fun*, which implies

$$\vdash_\Lambda [\mathrm{X}]\top \to \neg[\mathrm{X}]\bot.$$

Since Necessitation implies $\vdash_\Lambda [\mathrm{X}]\top$, for any $s \in S^\Lambda$ we get $[\mathrm{X}]\bot \notin s$, so there is some $t \in S^\Lambda$ with $sR_\mathrm{X}t$ (Box Lemma, page 37). But then $|s|R|t|$ in $S_\Gamma$.

**Fair Sequences**
Consider an $R$-branch (infinite $R$-sequence)

$$\tau_0 R \tau_1 R \cdots R \tau_n R \cdots\cdots\cdots$$

in $S_\Gamma$. $\tau$ will be called *fair* if it satisfies the condition

- *if $x$ occurs infinitely often in $\tau$, then so too does every member of $x^+$.*

The role of the Induction Rule is conveyed by the next result, which uses the Definability Lemma (page 40) stating that for any subset $X \subseteq S_\Gamma$ there is a formula $A_X$ such that for all $s \in S^\Lambda$,

$$A_X \in s \quad \text{iff} \quad |s| \in X.$$

**Induction Lemma**

*If $\tau$ is a fair $R$-branch in $S_\Gamma$, and*

$$X = \{x \in S_\Gamma : \ x \textit{ occurs infinitely often in } \tau\},$$

*then* $\vdash_\Lambda A_X \to [F]A_X$.

*Proof.*

By the Induction Rule, it is enough to show

$$\vdash_\Lambda A_X \to [\mathrm{X}]A_X,$$

so by properties of maximal sets it is enough to show that any $\Lambda$-maximal set containing $A_X$ must contain $[\mathrm{X}]A_X$.

So, let $A_X \in s \in S^\Lambda$. Then $|s| \in X$. Now if $sR_{\mathrm{X}}t$, then $|s|R|t|$, so $|t| \in |s|^+$. The fairness condition on $\tau$ then implies that $|t|$ is also in $X$, so $A_X \in t$. We thus have

$$sR_{\mathrm{X}}t \quad \text{implies} \quad A_X \in t,$$

which by the Box Lemma yields $[\mathrm{X}]A_X \in s$ as desired.

**Fairness Lemma**

*If $\tau$ is a fair $R$-branch in $S_\Gamma$, and $\sigma$ is any sequence in $S^\Lambda$ with $\sigma_j \in \tau_j$ for all $j \geq 0$, then $\sigma$ is admissible.*

*Proof.*

The requirement $|\sigma_j|R|\sigma_{j+1}|$, i.e. $\tau_j R \tau_{j+1}$, is immediate from the definition of $R$-branch.

Now let $<\mathrm{F}> B \in \sigma_j \cap \Gamma$. We have to show $B \in \sigma_k$ for some $k \geq j$. Suppose, for the sake of contradiction, that $B \notin \sigma_k$ for all $k \geq j$. Let $X$ be as in the Induction Lemma. Then if $s \in S^\Lambda$ has $A_X \in s$, $|s|$ occurs infinitely often in $\tau$, so $|s| = \tau_k$ for some $k \geq j$. Then $s \sim_\Gamma \sigma_k \in \tau_k$, so as $B \in \Gamma$ and $B \notin \sigma_k$ we get $B \notin s$. It follows, contrapositively, that every $\Lambda$-maximal set that does contain $B$ cannot contain $A_X$. Hence

$$\vdash_\Lambda B \to \neg A_X.$$

Then by the $<\mathrm{F}>$-Rule

$$\vdash_\Lambda <\mathrm{F}> B \to <\mathrm{F}> \neg A_X,$$

and so

(i) $$\vdash_\Lambda <\mathrm{F}> B \to \neg[\mathrm{F}]A_X$$

as $\neg[\mathrm{F}]A = \neg\neg <\mathrm{F}> \neg A$.

Now since $S_\Gamma$ is finite, there must indeed be points that occur infinitely often in $\tau$, and so there must be a $k \geq j$ with $\tau_k \in X$. Then we show that $<\mathrm{F}> B \in \sigma_i$ whenever $j \leq i \leq k$, by induction on $i$. The case $i = j$ is immediate from the assumption $<\mathrm{F}> B \in \sigma_j$. For the induction step, suppose $<\mathrm{F}> B \in \sigma_i$ and $j \leq i < k$. Since $\tau_i R \tau_{i+1}$, there exist $s \in \tau_i$ and $t \in \tau_{i+1}$ with $sR_{\mathrm{X}}t$. Then $<\mathrm{F}> B \in s$ as $s \sim_\Gamma \sigma_i$. Likewise $B \notin s$, as $B \notin \sigma_i$ by assumption. But putting $A = \top$ in axiom $\mathcal{U}2$ gives

$$\vdash_\Lambda \top \mathcal{U} B \leftrightarrow B \vee (\top \wedge [\mathrm{X}](\top \mathcal{U} B)),$$

yielding

$$\vdash_\Lambda <\mathrm{F}> B \to B \vee [\mathrm{X}] <\mathrm{F}> B.$$

It follows from this that $[\mathrm{X}] <\mathrm{F}> B \in s$. Hence $<\mathrm{F}> B \in t$ as $sR_{\mathrm{X}}t$, so finally $<\mathrm{F}> B \in \sigma_{i+1}$ as $t \sim_\Gamma \sigma_{i+1}$. This finishes the inductive argument.

We have now established that $<\mathrm{F}> B \in \sigma_k$, so from (i), $\neg[\mathrm{F}]A_X \in \sigma_k$ and therefore $[\mathrm{F}]A_X \notin \sigma_k$. But $|\sigma_k| = \tau_k \in X$, so $A_X \in \sigma_k$ and hence the Induction Lemma implies the contradictory conclusion $[\mathrm{F}]A_X \in \sigma_k$.

This contradiction forces us to accept that $B \in \sigma_k$ for some $k \geq j$, and the proof is complete.

## Listings and Schedules

The task before us is now reduced to the construction of fair $R$-branches.

A *listing* of $x^+$ is a finite sequence $y_0, \dots y_n$ of members of $x^+$ that includes each member of $x^+$ exactly once (i.e. a permutation of $x^+$). A *schedule* is a function $\Sigma$ that assigns to each $x \in S_\Gamma$ a listing $\Sigma(x)$ of $x^+$. The purpose of a schedule is to specify the order in which successors of $x$ are chosen in defining a fair branch.

Now given an arbitrary $x \in S_\Gamma$, define a sequence of points $\tau_j$ and schedules $\Sigma_j$ by induction, starting by putting $\tau_0 = x$ and letting $\Sigma_0$ be an arbitarily chosen schedule. For the induction step, supposing $\tau_j$ and $\Sigma_j$ are defined, let $\Sigma_j(\tau_j)$, the listing of $\tau_j^+$ given by $\Sigma_j$, be $y_0, \dots y_n$. Put

$$\tau_{j+1} = y_0$$

and define a new schedule $\Sigma_{j+1}$ by declaring

$$\Sigma_{j+1}(\tau_j) = y_1, \dots, y_n, y_0$$

and otherwise keeping $\Sigma_{j+1}$ the same as $\Sigma_j$.

Since $\tau_{j+1}$ is selected from $\tau_j^+$, we have $\tau_j R \tau_{j+1}$, so this process generates an $R$-branch. But the effect of the scheduling is that once selected, $\tau_{j+1}$ is moved to the back of the queue, so will not be selected again at a later stage until all other members of $\tau_j^+$ have been selected. Therefore as a particular point re-occurs in the sequence it will be followed by each of its $R$-successors in turn. In particular, if a point $z$ occurs infinitely often, the scheduler will cycle infinitely often through $z^+$, so each member of $z^+$ will occur infinitely often, and a fair $R$-branch is generated.

The proof of the Sequence Lemma can now be completed. Given $s \in S^A$, construct a fair $R$-branch $\tau$ by the process just described, starting with $\tau_0 = |s|$. Put $\sigma_0 = s$ and otherwise let $\sigma_j$ be any member of $\tau_j$. By the Fairness Lemma, $\sigma$ is an admissible sequence starting with $s$.

## References

D. Gabbay, A. Pnueli, S. Shelah, J. Stavi, *On the Temporal Analysis of Fairness*, Proc. 7th ACM Symposium on Principles of Programming Languages, 1980, pp. 163-173.

Colin Stirling, *Modal and Temporal Logics*, in Handbook of Logic in Computer Science, vol. 2, S. Ambramsky et. al. (eds.), Clarendon Press, Oxford, 1992, pp. 447–563.

## Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

### Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines.

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled. Figures should be submitted as original line drawings/printouts, and not photocopies.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1–3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) LaTeX file(s), either on a diskette, or via e-mail/ftp – a LaTeX style file is available from the production editor;
2. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

   Further instructions on how to reduce page charges can be obtained from the production editor.
3. In camera-ready format – a detailed page specification is available from the production editor;
4. In a typed form, suitable for scanning.

Authors will be expected to sign a copyright release form.

### Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R30-00 per final page for LaTeX or camera-ready contributions that require no further attention. The maximum is R120-00 per page for contributions in typed format (charges include VAT).

These charges may be waived upon request of the author and at the discretion of the editor.

### Proofs

Proofs of accepted papers in categories 2 and 4 above may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Camera-ready submissions will only be accepted if they are in strict accordance with the detailed guidelines. It is the responsibility of the authors to ensure that their submissions are error-free.

### Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

### Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

### Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

## References

1. E Ashcroft and Z Manna. 'The translation of 'goto' programs to 'while' programs'. In *Proceedings of IFIP Congress 71*, pp. 250–255, Amsterdam, (1972).
2. C Bohm and G Jacopini. 'Flow diagrams, turing machines and languages with only two formation rules'. *Communications of the ACM*, 9:366–371, (1966).
3. S Ginsburg. *Mathematical theory of context free languages*. McGraw Hill, New York, 1966.

**South African Computer Journal**

Number 13, April 1995
ISSN 1015-7999

**Suid-Afrikaanse Rekenaar-tydskrif**

Nommer 13, April 1995
ISSN 1015-7999

# Contents

UP Drukkers Printers