Process Algebra: a unifying approach

Tony Hoare

Marktoberdorf Summer School
August 2004

Applications

- Hardware
- Communications
- Parallel programs
- Multi-processors
- Networks
- W W W

- Scientific models
  in biology, psychology, sociology,…

Process Algebra

gives mathematical support for

- Specification
- Development
- Implementation
- Testing

- Design
- Optimisation
- Analysis
- Verification

of computer systems

Lecture one

- Deterministic Transition Systems
- Traces
- Refinement

A deterministic transition system

is an edge-labelled graph that has

- nodes representing processes: p,q,…
- labels representing events: e,f,…
- a special node: * (not a process)
- a function after: nodes **X** labels → nodes

After

- p/e is the state of p after it has done e
  - p/e = * if p cannot do e
  - */e = *
- * makes / into a total function

Traces

- Extend / to sequences of labels:
  - $p/\langle\ \rangle =_{def} p$
  - $p/(\langle e\rangle s) =_{def} (p/e)/s$
- $traces(p) =_{def} \{ s \mid (p/s) \neq * \}$
  - $traces(*) = \{\ \}$

Refinement

- $p \geq q \;=_{def}\; traces(q) \subseteq traces(p)$
- $p \equiv q \;=_{def}\; p \geq q \;\&\; q \geq p$
  - called trace equivalence
  - implies equality in automata theory
- refinement is basic to CSP
  - supports specification
  - and stepwise development

Relations

- A relation is a set of ordered pairs

  e.g., the empty relation

  $$\mathrm{id} \;=_{def}\; \{(p,q) \mid p = q\}$$

  $$\geq \;=_{def}\; \{(p,q) \mid p \geq q\}$$

- $\text{-e->} \;=_{def}\; \{(p,q) \mid p/e = q \;\;\&\;\; q \neq * \neq p\}$

Transitions

- p -e-> q means that a process in initial state p , on occurrence of event e, will move to state q
- p -e-> $=_{def}$ p/e ≠ * ≠ p
- p -|e-> $=_{def}$ p/e =   *

Relational Composition

- If S and T are relations,

  $S \, ; \, T =_{def} \{(p,r) \mid p \, S \, q \;\; \& \;\; q \, T \, r, \text{ for some } q\}$
- S **U** T = their set union
- S n T = their intersection
- S **<u>C</u>** T means set inclusion
  - p in S implies p in T, for all p

Relational Algebra

- id ; S = S = S ; id
- (S ; T) ; R = S ; (T ; R)
- S $\underline{\mathbf{C}}$ T implies S ; R $\underline{\mathbf{C}}$ T ; R
  and R ; S $\underline{\mathbf{C}}$ R ; T
- S ; (T **U** R) = (S ; T) **U** (S ; R)
- (T **U** R) ; S = (T ; S) **U** (R ; S)

Refinement order

- Thm. $\geq$ is reflexive and transitive, ie,
  - $\mathrm{id} \quad \underline{\mathbf{C}} \quad \geq$
  - $\geq \,;\, \geq \quad \underline{\mathbf{C}} \quad \geq$

- F is monotonic $=_{def}$

  $p \geq p'$ & $q \geq q'$ & …

  implies $F(p, q, \ldots) \geq F(p', q', \ldots)$

Monotonicity

- Thm. _ /e is monotonic, ie,

  p ≥ q implies p/e ≥ q/e

- Thm. (≥ ; -e->) $\subseteq$ (-e-> ; ≥)
  – transitions respect trace refinement
  – just restates the previous theorem

Lecture two

- Simulation
- Unification
- Operational semantics

A simulation

is any relation S between processes s.t.

S ; -e-> **<u>C</u>** -e-> ; S

– the empty relation, identity,
  refinement , trace equivalence

- composition of simulations is a simulation
  – so is the union of a set of simulations,
  – and the intersection of a non-empty set

Bisimulation

- A bisimulation is a symmetric simulation
  - e.g: empty, identity, trace equivalence
- Bisimulation is basic to CCS
  - justifies automatic model checking
  - supports co-inductive proofs

Simulation implies refinement

- Proof: by induction on the length of traces. See lecture notes

Similarity

- similarity $=_{def}$ the union of all simulations
  - which is itself a simulation
  - the largest one, includes all the others

Unification

- In a deterministic transition system
  similarity and refinement coincide

Proof: similarity is a simulation, and so implies refinement. Refinement is a simulation, and so implies similarity.

A process algebra

defines a syntax to name all nodes

- STOP, RUN , e**.**p , (p |&| q) , (p |**v**| q)
  - where p and q are processes
- distinct syntax names distinct nodes
  - unless equated by structural equivalence

Structural Equivalence

defined by axioms like

- (e.*) = *
- (p |&| *) = * = (* |&| p)
- (p |**v**| *) = p = (* |**v**| p)

- * cannot be expressed in the syntax

An operational semantics
defines _ /e by induction on its syntax

- STOP/e = *
- RUN/e = RUN
- (f.p)/e = p if f = e

  = * otherwise

- STOP does nothing
- RUN does anything
- f.p does f , then behaves like p

Trace semantics

- Proved from the operational semantics
  - not the other way round
  - because processes with same traces will later be differentiated by non-determinism

Theorems

- traces(*) = { }
- traces(STOP) = {< >}
- traces(RUN) = all sequences of labels
- traces(f.p) = {< >} **U** {<f>t | t in traces(p)}

Semantics of parallel

- (p |&| q)/e = (p/e) |&| (q/e)
- (p |**v**| q)/e = (p/e) |**v**| (p/e)
- (p ||| q)/e = ….

Traces

- Thm: traces((p|**v**|q)) =
  traces(p) **U** traces(q)

- Thm: traces((p|&|q)) = their intersection

Boolean Algebra

- RUN |&| p $\equiv$ p unit law
- STOP |&| p $\equiv$ STOP zero law
- p |&| p $\equiv$ p idempotence
- p |&| q $\equiv$ q |&| p symmetry
- (p |&| q) |&| r $\equiv$ p |&| (q |&| r) assoc
- (p |**v**| q) |&| r $\equiv$ (p |&| r) |**v**| (q |&| r)
- dually for |**v**|

External choice

- ((e.p) |**v**| (f.q)) |&| (e.r)

  $\underline{\equiv}$ e.(p |&| r) if e $\neq$ f

- ((e.p) |**v**| (f.q)) |&| (g.r)

  $\underline{\equiv}$ STOP if g $\neq$ f and g $\neq$ e

Problem?

- $((e.p)\ |\mathbf{v}|\ (e.q)) \quad |\&| \quad (e.r)$

  $\equiv \quad e.((p\ |\mathbf{v}|\ q)\ |\&|\ r\ )$

  – delayed choice, which is inefficient

- That's why we introduce

  non-determinism

Lecture three

- Non-determinism
- Reduction
- Operational Semantics

Non-determinism

- Let -tau-> be a relation between processes

- interpreted as
  - a ‘silent’ transition
  - an internal computation
  - an algebraic reduction
  - a committed choice

Healthiness condition

- (-tau-> ; -e->)   **C** (-e-> ; –tau?->)
  – where   -tau?->   =   (id **U** -tau->)

  – formalises invisibility of tau
  – permits optimisation
  – by postponement of –tau->
  – or its elimination

Reduction (→)

- Define → $=_{def}$ (-tau->)*

- → is a reflexive transitive simulation

- Define p to be stable iff p-|tau->

Weak Transitions

- =e==>   =def   → ; -e-> ; →
  - non-deterministic, as in CCS

- Lemma: =e==>   =   (-e-> ; →)

  = (→ ; =e==>)   =   (=e==> ; →)

Proof: from simulation and transitivity of →

A weak simulation

- is a relation W such that

  (W ; =e=>) **<u>C</u>** (=e=> ; W)

  – e.g., { }, id,
- the composition and union of weak simulations is a weak simulation
  – not the intersection
- weak similarity is largest weak simulation

Theorem

- If W is a weak simulation then
  (→ ; W) is a simulation

Proof: (→;W) ; -e->

= → ; W ; -e-> ; → lemma

= → ; W ; =e=> lemma

$\underline{C}$ -e-> ; (→ ;W) weak simulation

Theorem

- If S is a simulation, (S ; →) is a weak one

Proof: (S ; →) ; (-e-> ; →)

=	S ; -e-> ; →	lemma
$\subseteq$	-e-> ; S ; →	simulation
$\subseteq$	(-e-> ; →) ; (S ; →)	→ reflexive

Unification

- Thm: weakly similar = similar

- Proof:

  Let W be weak similarity. So (→ ; W) is a simulation, and therefore contained in similarity. Similarly, the reverse containment.

Semantics for tau

- STOP -|tau->   RUN -|tau->
- (e.p) -|tau->
- (p |&| q) -tau-> (p’ |&| q’)

  iff   p -tau-> p’ and q = q’

  or   q -tau-> q’ and p = p’
- similarly for |**v**|

New processes

- CHAOS
  - totally non-deterministic
- (p **v** q)
  - internal (demonic) choice
- (p [] q)
  - external (environmental) choice

Non-determinism

- distinguishes processes with the same traces, e.g., CHAOS vs. RUN.

  – CHAOS/e = CHAOS, for all e
  – CHAOS –tau-> p for all p

  – RUN/e = RUN for all e
  – RUN –|tau->

Internal non-determinism

- (p **v** q)/e   =   (p/e **v** q/e)
- (* **v** p)   =   p   =   (p **v** *)
- (p **v** q) –tau-> p and (p **v** q) –tau-> q
  - like internal choice in CSP
  - implemented in CCS as: (tau.p + tau.q)

External non-determinism

- (p [] q)/e = (p or q)/e
  - after first step, the choice is internal
- (p [] q) -tau-> (p’ [] q’)

  iff p -tau-> p’ & q = q’

  or q -tau-> q’ & p = p’
  - (as in CSP) initial internal reductions cannot remove the external choice
  - (in CCS) + is different

Problem?

- (p [] q) $\equiv$ (p **v** q) $\equiv$ (p |**v**| q)

- trace equivalence does not distinguish different kinds of choice

- that's why we introduce

Barbs

Lecture four

- Barbed Transition Systems
- Refusals
- Divergences

Barbs

- serve as labels for the nodes
  - appearing only at the end of a trace
- explain why the process has terminated
  - either it has finished successfully
  - or it has deadlocked (refuses to act)
  - or it has entered an infinite loop (diverged)
- eliminate the need for tau

Barbs

- Let # (sharp) be a distinguished process
  - such that traces(#) = {< >}

- Let BARB be a distinguished set of labels

  appearing only just before # , so

  If p –e-> q then (e in BARB iff q = #)

A barbed simulation

is a simulation S s.t. for all b in BARB

(S ; -b->) $\underline{C}$ -b->

- Thm: All simulations are barbed simulations (and vice versa)

  Since a barb is a label, if S is a simulation,

  p S q and q-b->r implies p-b->p' and p' S #

  By the property of barbs, r = p' = #

Refusals

- Let ref(X) be a barb
  - where X is a set of normal labels
  - not including barbs
  - indicates deadlock in an environment that expects any of the events of X

Healthiness condition

- p –ref(X)-> # iff $X \subseteq \{e \mid p/e = * \}$
  – if p is stable
- p –ref(X)-> # iff p $\rightarrow$; –ref(X)-> #
  – if p is unstable

- this is the defining property of refusals

Theorem

- STOP/ref(X) = # for any X
- RUN/ref(X) = *
- CHAOS/ref(X) = # for any X
- (f.p)/ref(X) = # iff f is not in X

continued

- (p |&| q)/ref(X **U** Y) = # if p/ref(Y) = #
  and q/ref(Z) = #
- (p [] q)/ref(X) = # iff p/ref(X) = #
  and q/ref(X) = #
- (p **v** q)/ref(X) = # iff p/ref(X) = #
  or q/ref(X) = #

Divergences

- p/div = # iff p –tau-> p' –tau-> … forever
- CHAOS/div = #

  (because CHAOS -tau-> CHAOS)
- STOP, RUN and f.p

  have no divergence barb

  (because they have no tau transitions)

continued

- (p |&| q), (p |**v**| q), (p **v** q), (p [] q)
  have a divergence barb

  iff one (or both) of their operands has a
  divergence barb.

Communicating Sequential Processes

- CSP failures are just traces
  - with ref(X) barbs at the end
- CSP divergences are just traces
  - with a div barb at the end

Summary

- similarity
- barbed similarity
- trace refinement
- weak similarity
- failures refinement
- FDR

- ... by curious coding tricks…
- … all turn out to be the same.

Acknowledgements

- Robin Milner, Bill Roscoe, He Jifeng, Sriram Rajamani, Jakob Rehof, Cedric Fournet, Paul Gardiner, Gavin Lowe, Rob van Glabbeek,…
- and many others.