Two Models of Communicating Transaction Processes

or

Are our colleagues letting us down?

Shaofa Yang & Dines Bjørner

China & The Royal Kingdom of Denmark

BCS-FACS Christmas Meeting

• 19 December 2005, London, UK •

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

- Communicating transaction processes (CTP) form a hybrid
  - ⋆ between condition event Petri nets
  - ⋆ and simple forms of message sequence charts
- CTPs were proposed by
  - ⋆ A. Roychoudhury and P.S. Thiagarajan in the paper:
  - ⋆ *Communicating Transaction Processes.*
  - ⋆ Proc. of the 3rd IEEE International Conference on Application of Concurrency in System Design (ACSD'03) (IEEE Press, 2003)

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Structure of Talk — I

- There are three-by-two parts to this talk:
  - ⋆ A presentation and a model-oriented semantics of CTP:
    - ◇ narrative and
    - ◇ formalisation.
  - ⋆ A biased review of the original CTP paper:
    - ◇ general overview of that paper and
    - ◇ focus on its presentation of the syntax and semantics of CTP.
    - ◇ Done by reference tp a copy of a CTP publication — in your hands.
  - ⋆ A lamentation and a plea:
    - ◇ Lamentoso: our colleagues do not apply formal specicifications!
    - ◇ Let all university courses — compiler design, operating system design, design of distributed & protocol systems, design of data base management systems, application systems — be based on the use of formal specifications.

  E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Structure of Talk — II

- After the narrative
  - ⋆ carried only by narration of "generalised" CTP diagram fragments
- we reformulate that "story" on CTPs,
  - ⋆ that is, we rephrase the referenced paper's, to us, rather cumbersome notation
  - ⋆ into a model-oriented formal specification in the tradition of `RAISE`, `VDM` and `Z`.
- Then
  - ⋆ lamentoso
  - ⋆ followed by wishes for a very merry Christmas and a Happy New Year!

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Intuition

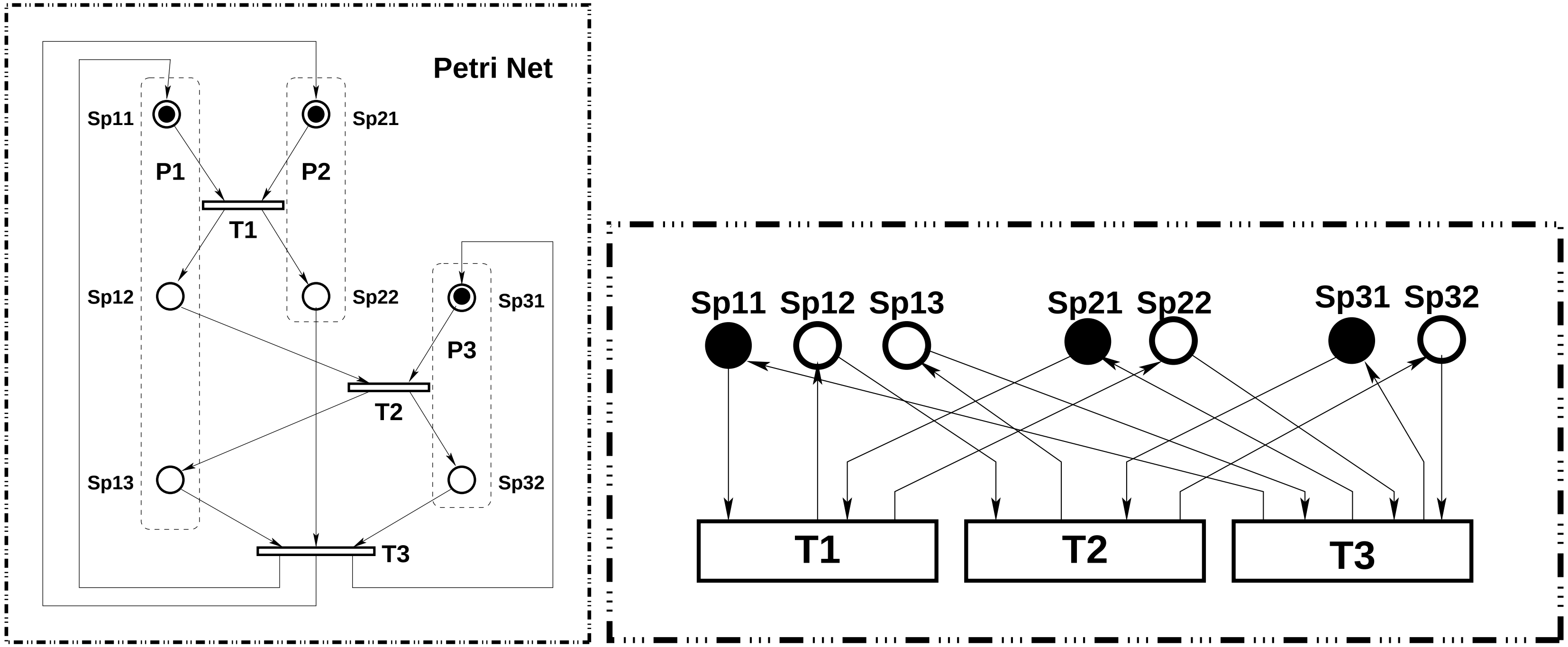

Figure 1.1: Left: a Petri net. Right: a concrete CTP diagram

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Narration of CTPs
CTP Diagrams

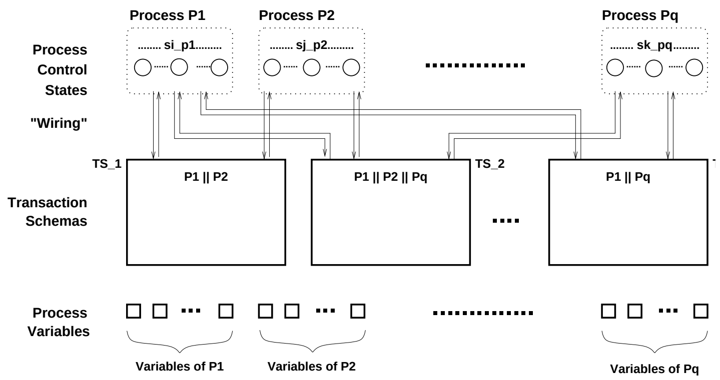

Figure 1.2: A schematic CTP diagram

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

- A CTP diagram consists of
  - ⋆ **an indexed set of sets of process (control) states,**
  - ⋆ **an indexed set of transaction schemas,**
  - ⋆ **an indexed set of sets of process variables,** and
  - ⋆ **a "wiring" connecting control states via transaction schemas to control states.**
- (The wiring of Fig. 1.2 on the facing page is shown by pairs of opposite directed arrows.)

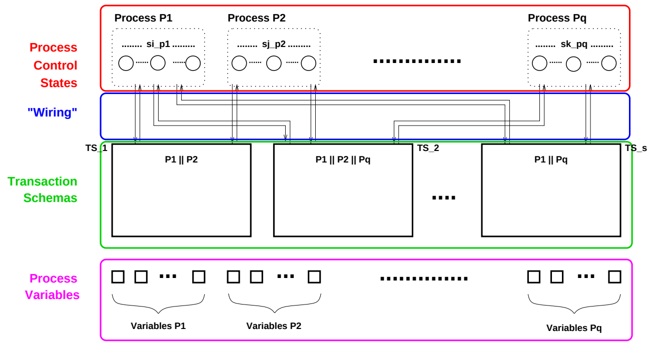

Figure 1.3: A schematic CTP diagram

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

CTP Processes

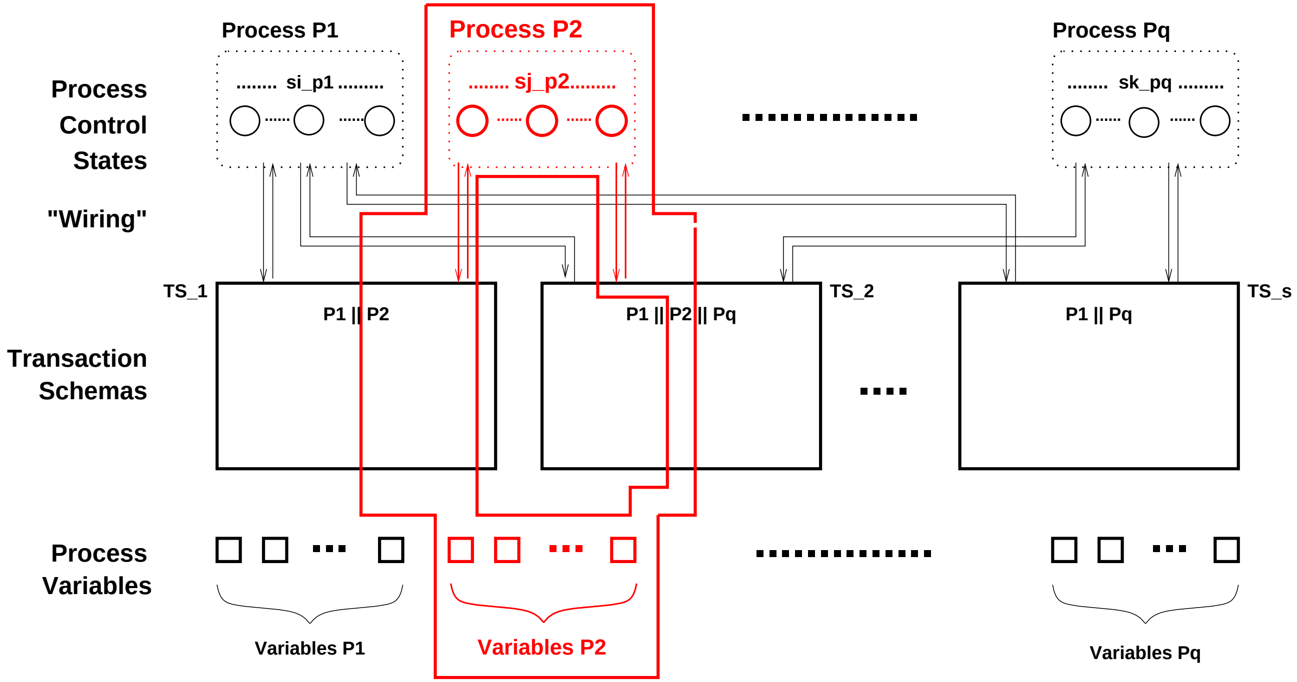

Figure 1.4: A schematic CTP diagram: **process P2** is "framed"

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

- The set of all allowable, i.e., specified state to next state transitions
- can be specified as a set of triples, each triple being of the form:
  - ⋆ (s, ts_n, s') for process p_i: $(s_{p_i}, ts_n, s'_{p_i})$
- If ts_n supports processes p_i, p_j ... p_k, then there will be triples:
  - ⋆ $(s_{p_i}, ts_n, s'_{p_i}), (s_{p_j}, ts_n, s'_{p_j}), \ldots, (s_{p_k}, ts_n, s'_{p_k})$

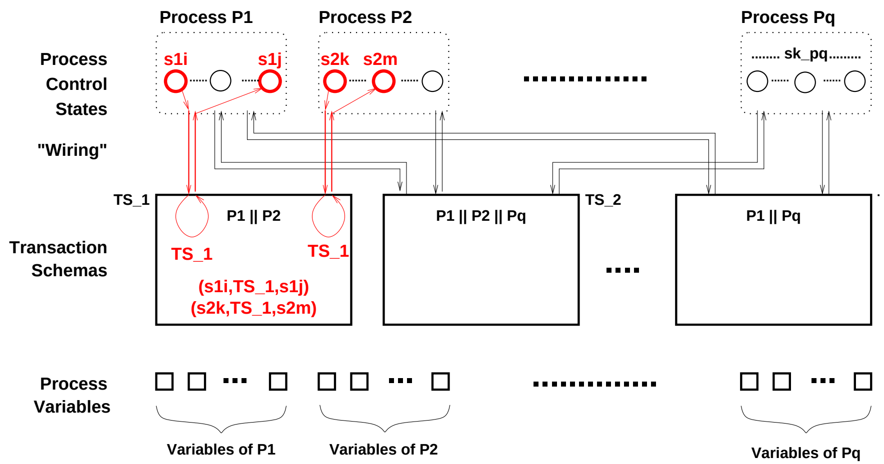

Figure 1.5: State to next state transitions shown for TS_1 only

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

CTP Transaction Schemas

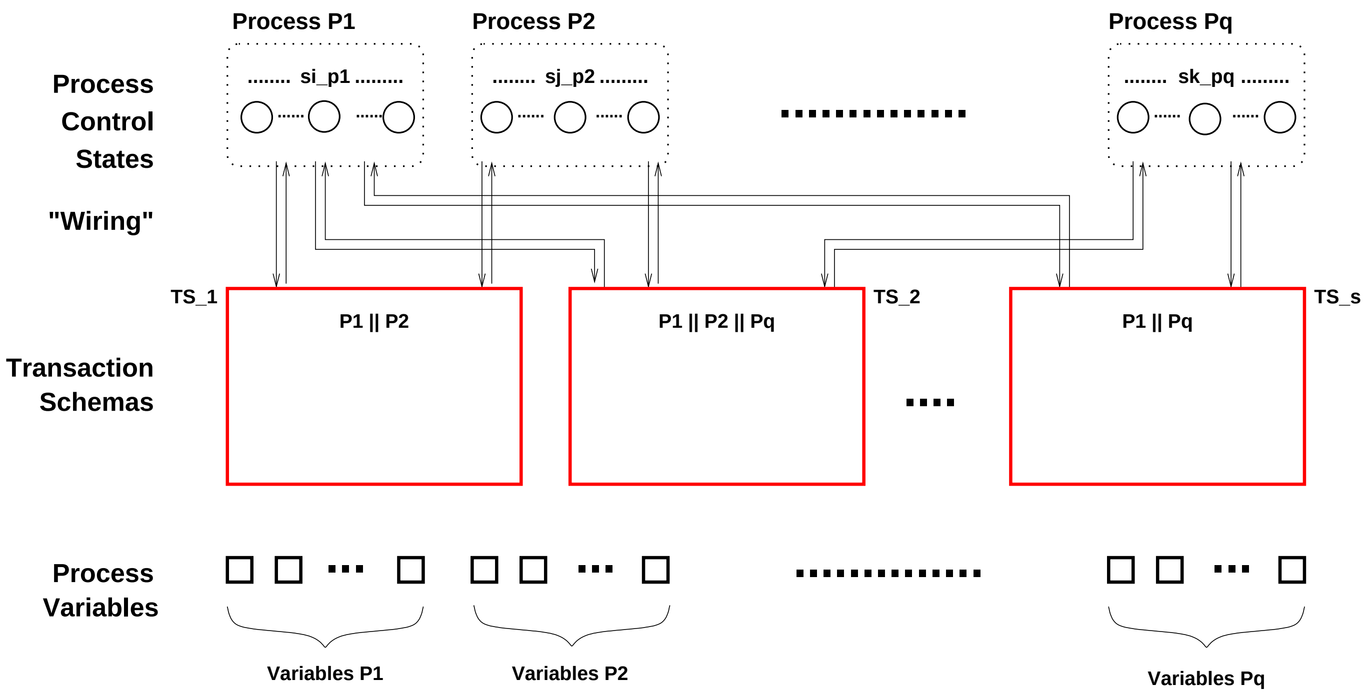

Figure 1.6: The schematic CTP **transaction schemas**

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

CTP Transaction Charts

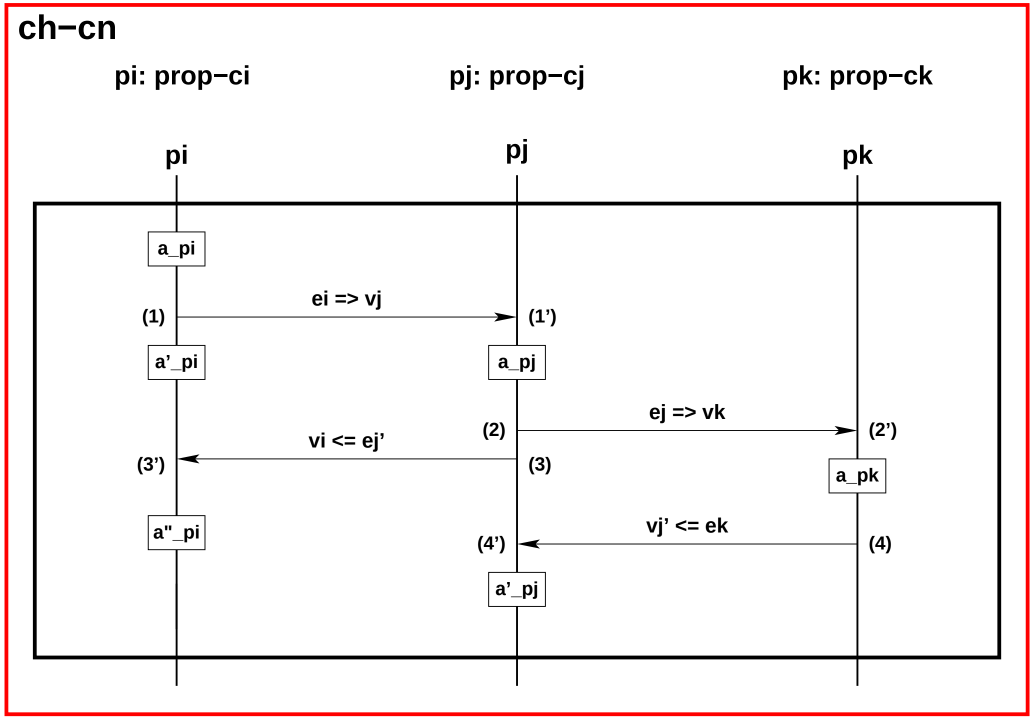

Figure 1.7: A transaction chart with simple message sequence chart

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Enabled CTP Transaction Charts

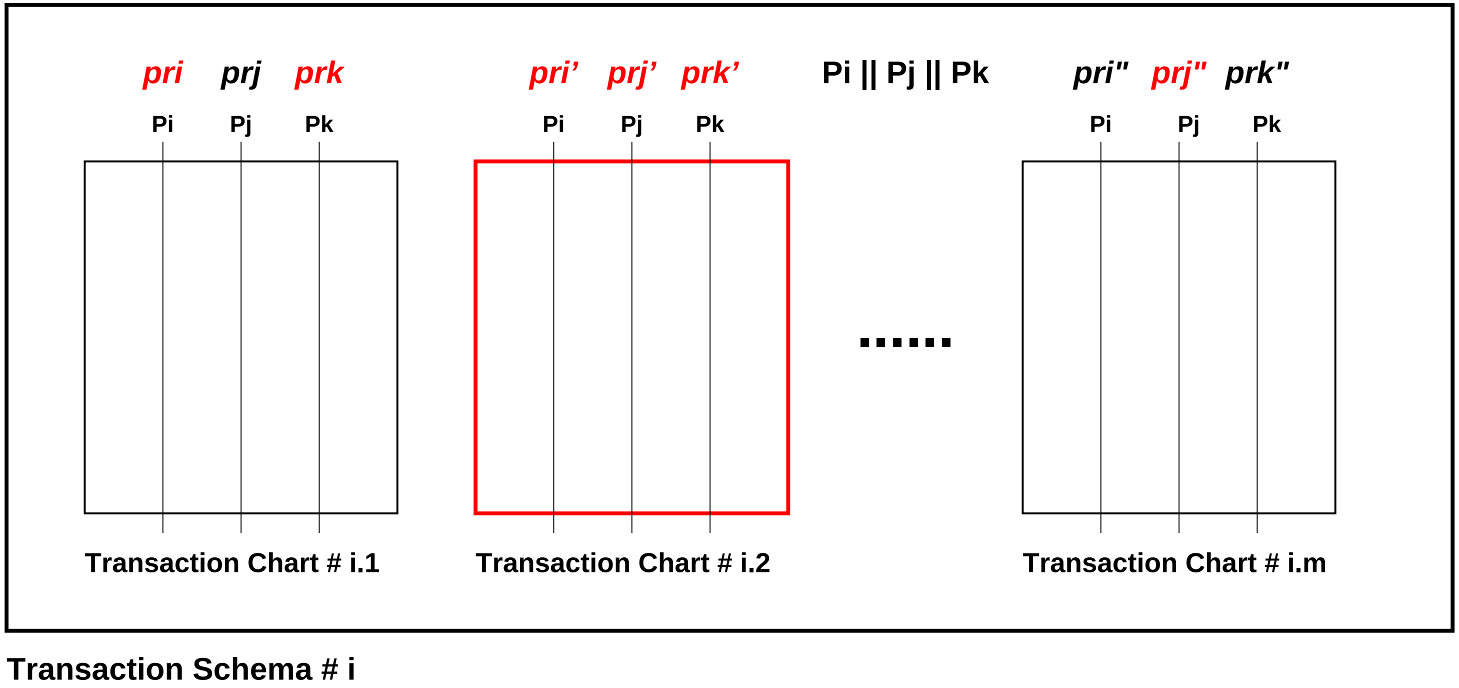

Figure 1.8: **Enabled chart** of a schema

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Enabled Versus Invoked Schemas and Charts

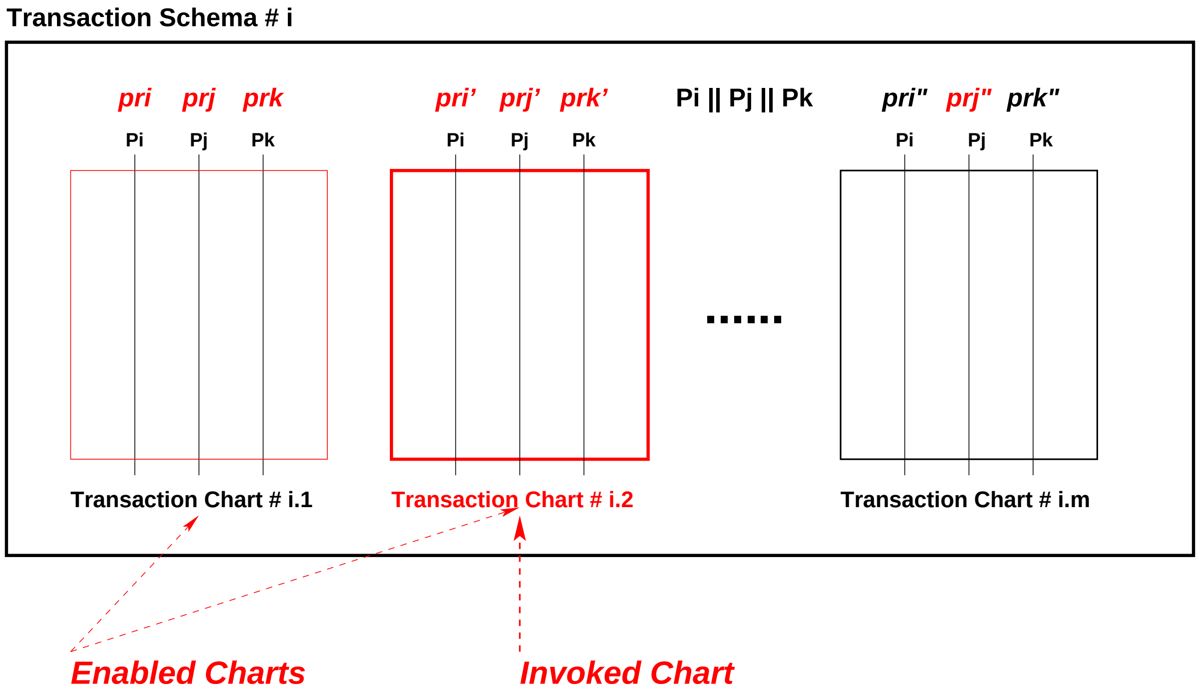

Figure 1.9: Two **enabled charts** and one **invoked chart** of a schema

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/˜db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

CTP Transitions

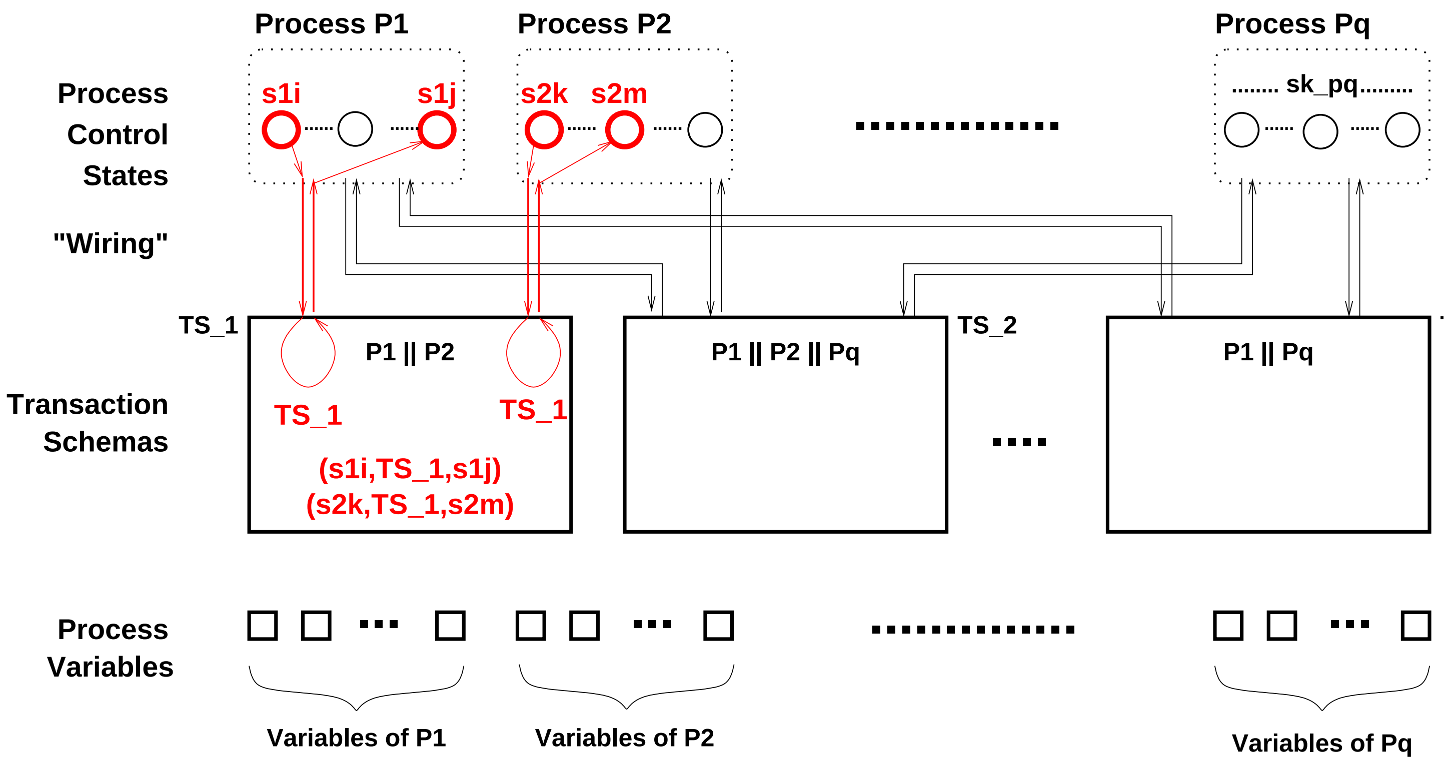

Figure 1.10: State to next state transitions shown for TS_1 only

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

An invoked transaction chart will then result

- in the appropriate input states no longer being marked,
- in the execution of the simple message sequence chart, from top to bottom,
- in the updating of process variables (as the result of execution of each of the instances of the simple message sequence chart),
- and, once message sequence chart execution terminates, in the marking of one appropriate output state for each of the processes labelling that transaction chart.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Which of the output states, for processes p_i, p_j and p_k, that is,

- which of $s_{p_i}^{\prime 1}$, $s_{p_i}^{\prime 2}$, $\ldots$, $s_{p_i}^{\prime m_i}$, and
- which of $s_{p_j}^{\prime 1}$, $s_{p_j}^{\prime 2}$, $\ldots$, $s_{p_j}^{\prime m_j}$, and
- which of $s_{p_k}^{\prime 1}$, $s_{p_k}^{\prime 2}$, $\ldots$, $s_{p_k}^{\prime m_k}$

are selected is determined by which of the

- $(s_{p_i}^{\alpha}, ts_n, s_{p_i}^{\beta})$

transition rules had their

- $s_{p_i}^{\alpha}$

part apply in the invocation of transaction schema ts_n to which this chart belongs.

  E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Formalisation of CTPs
The Syntactic and Some Semantic Types

type
P, T, S, Var, Typ, VAL, Chtn, Exp, AP, Act

Annotation:

P, T, S, Var, Typ, VAL, Chtn, Exp, AP, Act: Process names, transaction schema names, process control states (i.e., names), variable identifiers, type designators (for example `integer, Boolean` and so on), semantic values (for example **Int, Bool** and so on), chart names, expressions (further undefined, but are usually variables, prefix expressions and infix expressions over usual integer operators and Boolean connectives), atomic propositions (i.e., Boolean valued expressions over variables) and internal actions (assignments, conditional actions, etc.). ■[1]

[1] ■ means: end of annotation.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

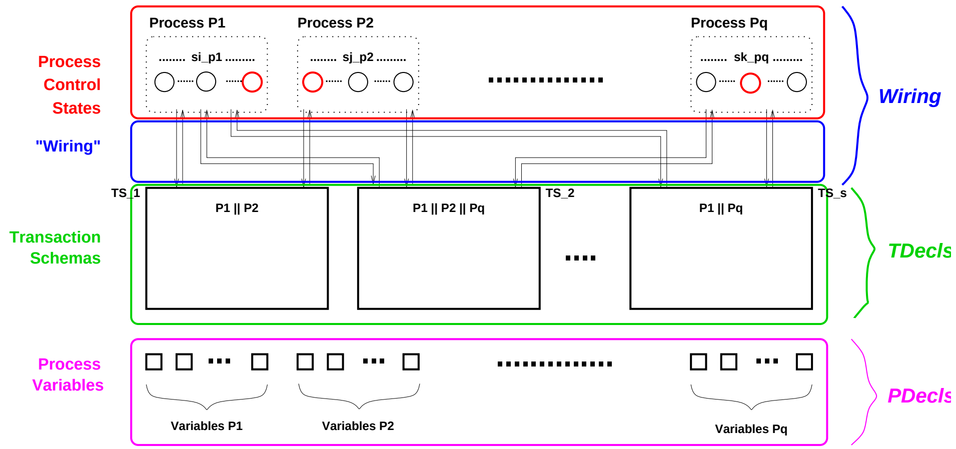

Figure 1.11: A schematic CTP diagram

type

$\text{Prog}' = \text{PDecls} \times \text{TDecls} \times \text{Wiring} \times \text{Init}$

$\text{Prog} = \{| \text{ prog:Prog}' \cdot \text{wf\_Prog(prog)} |\}$

Annotation:

Prog: A CTP program consists of well-formed combinations of **process variable** and **transaction schema declarations** , of **wiring** and the **definition of an intialisation (of process control states and variable values)** . ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

type

$\text{PDecls} = \text{P} \overrightarrow{m} \text{VarDecl}$

$\text{TDecls} = \text{T} \overrightarrow{m} (\text{Chtn} \overrightarrow{m} (\text{Gd} \times \text{Cht}))$

Annotation:

PDecls, VarDecl: For each process there is a set of variables of specified type.

TDecls: For each transaction schema name, T, there is a set of uniquely named, Chtn, transaction charts, with each chart consisting of a guard, Gd, and the chart proper Cht. ∎

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

type

$$\mathrm{Wiring} = \mathrm{T} \ \overrightarrow{m}\ (\mathrm{P} \ \overrightarrow{m}\ \mathrm{S} \times \mathrm{S})$$

$$\mathrm{Init} = \mathrm{P} \ \overrightarrow{m}\ (\mathrm{S} \times \mathrm{VarInit})$$

$$\mathrm{VarDecl} = \mathrm{Var} \ \overrightarrow{m}\ \mathrm{Typ}$$

Annotation:

Wiring: For each transaction schema and for each process (that applies to this schema) there is a pair of respectively input and output control states.

Init, VarInit: With each process a control state, S, is associated an initialisation, respectively the current values of all variables of this process. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

type

$\mathrm{Gd} = \mathrm{P} \xrightarrow[m]{} \mathrm{Prop}$

$\mathrm{Prop} == \mathrm{mkTrue} \mid \mathrm{mkAP(ap{:}AP)} \mid \mathrm{mkNot(pr{:}Prop)}$
$\quad \mid \mathrm{mkAnd(pr{:}Prop,pr'{:}Prop)} \mid \mathrm{mkOr(pr{:}Prop,pr'{:}Prop)}$

Annotation:

Gd, Prop: A transaction chart guard associates

- to each of the processes associated with that chart
- a proposition which is
- either the value true,
- or is an atomic proposition,
- or a negated,
- or a conjunctive
- or a disjunctive

proposition. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

type

$\mathrm{Cht} = (\mathrm{P} \underset{m}{\rightarrow} \mathrm{Ev}^*) \times \mathrm{SendRecv}$

$\mathrm{Ev} == \ \mathrm{mkSe(p{:}P,e{:}Exp)} \mid \mathrm{mkRe(p{:}P,v{:}Var)} \mid \mathrm{mkAct(act{:}Act)}$

$\mathrm{SendRecv} = (\mathrm{P} \times \mathrm{Pos}) \underset{m}{\rightarrow} (\mathrm{P} \times \mathrm{Pos})$

$\mathrm{Pos} = \mathbf{Nat}$

$\Sigma = \mathrm{Var} \underset{m}{\rightarrow} \mathrm{VAL}$

$\mathrm{VarInit} = \Sigma$

Annotation:

Cht, Ev*, SendRecv: A transaction chart maps each of its associated processes into an instance — which is an event list — and a mapping, SendRecv, that relates output and input events in respective process instances.

Ev: An event is either a send event, or a receive event, or an internal action.

Pos: A position is an index into an event list. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Auxiliary Syntactic and Semantic Function Signatures

value

typeof: Exp → VarDecl → Typ

wf_AP: AP → VarDecl → **Bool**
wf_Act: Act → VarDecl → **Bool**

Annotation:

typeof: Extracts from an expression, given a set of variable declarations, the type of the value of the expression, if well–formed.

wf_AP: Examines whether an atomic proposition is well–formed.

wf_Act: Examines whether an internal action text is well–formed. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

wf_Exp: Exp → VarDecl → **Bool**
eval_AP: AP → Σ → **Bool**
eval_Act: Act → Σ → Σ
eval_Exp: Exp → Σ → VAL

Annotation:

eval_AP: Evaluates an atomic proposition.

int_Act: Interprets an internal action, possibly leading to changes in the values of variables.

eval_Exp: Evaluates an expression.

wf_Exp: Examines whether an expression is well–formed. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Auxiliary Function Signatures and Definitions

value

participants: T → Prog′ → P-**set**

participants(t)(prog) ≡ **let** (_,_,wiring,_)= prog **in dom** wiring(t) **end**

instances : Cht → P-**set**

instances(cht) ≡ **let** (pevs,_) = cht **in dom** pevs **end**

Annotation:

participants: Extracts the set of process (names) participating in a transaction schema

instances: Extracts the set of instances of a chart. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

xtr_APs: Prop → AP-**set**

xtr_APs(pr) ≡ **case** pr **of** mkTrue → {}, mkAP(ap) → {ap}, ... **end**

eval_Prop: Prop → PΣ → **Bool**

eval_Prop(pr)(pσ) ≡

case pr **of** mkTrue → **true**, mkAP(ap) → eval_AP(ap)(pσ), ... **end**

Annotation:

xtr_APs: Extracts, from a proposition, the set of atomic propositions occuring in a proposition.

eval_Prop: Evaluates a proposition. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Well-formedness of CTP

value
  wf_Prog : Prog′ → **Bool**
  wf_Prog(prog) ≡
    let (_,_,wiring,_) = prog **in**
    All_Wired(prog) ∧
    All_Initialized(prog) ∧
    wf_Gds_and_Chts(prog) ∧
    wf_Wiring(prog) ∧
    wf_Init(prog)
    end

Annotation:

wf_Prog: Conjunction of five constraints.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
All_Wired: Prog′ → **Bool**
All_Wired(prog) ≡
let (_,tdecls,wiring,_) = prog **in** **dom** tdecls = **dom** wiring **end**

All_Initialized: Prog′ → **Bool**
All_Initialized(prog) ≡
let (pdecls,_,_,init) = prog **in** **dom** pdecls = **dom** init **end**

Annotation:

All_Wired: All transaction schemas are wired.

All_Initialized: Each process is initialized. (The initialization of a process includes not only the variables but also an initial control state.) ■

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
wf_Gds_and_Chts: Prog′ → **Bool**
wf_Gds_and_Chts(prog) ≡
  let (pdecls,tdecls,_,_) = prog **in**
  ∀ t:T•t ∈ **dom** tdecls ⇒
    let (gd,cht) = tdecls(t)(chtn) **in**
    dom gd = instances(cht) = participants(t)(prog) ∧
    wf_Gd(gd)(pdecls) ∧ wf_Cht(cht)(pdecls)
  end end

wf_Gd: Gd → PDecls → **Bool**
wf_Gd(gd)(pdecls) ≡
  ∀ p:P•p ∈ **dom** gd ⇒ ∀ ap:AP • ap ∈ xtr_APs(gd(p))
    ⇒ wf_AP(ap)(pdecls(p))

Annotation:

wf_Gds_and_Chts: The guards and charts are well-formed. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
  wf_Cht: Cht → PDecls → **Bool**
  /∗ see later ∗/

  wf_Wiring: Prog′ → **Bool**
  wf_Wiring(prog) ≡
    let (pdecls,_,wiring,_) = prog **in**
      ∀ t:T·t ∈ **dom** wiring ⇒
      participants(t)(prog)⊆**dom** pdecls ∧
      ∀ p:P·p ∈ **dom** wiring(t) ⇒ **let** (s,s′)=wiring(t)(p) **in** s ≠ s′ **end**
    end

Annotation:

wf_Wiring: The wiring is well-formed. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
wf_Init: Prog′ → **Bool**
wf_Init(prog) ≡
  let (pdecls,_,_,init) = prog **in**
  ∀ p:P•p ∈ **dom** init ⇒
    let (s,varinit) = init(p) **in**
    (∃ t:T,s′:S • (s,s′)=wiring(t)(p)) ∧ wf_VarInit(varinit)(vardecl(p))
  end end

Annotation:

wf_Init: The initialisation is well-formed (the initialisation includes both initial control states and initial values of variables). ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

wf_VarInit: VarInit → VarDecl → **Bool**

wf_VarInit(varinit)(vardecl) ≡

(**dom** vardecl = **dom** varinit) ∧

(∀ var:Var•var ∈ **dom** vardecl ⇒ typeof(varinit(var))=vardecl(var))

Annotation:

wf_VarInit: All variables are initialised to values of the declared type.

■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Well-formedness of Charts

value

wf_Cht: Cht → PDecls → **Bool**

wf_Cht(cht)(pdecls) ≡ wf_Evs(cht)(pdecls) ∧ wf_SendRecv(cht)

Annotation:

wf_Cht: All events are well-formed and so are all send-receive pairs.■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

```
value
  wf_Evs: Cht → PDecls → Bool
  wf_Evs(cht)(pdecls) ≡
    let (pevs,_) = cht in
    ∀ p:P,ev:Ev•
      p ∈ dom pevs ∧ ev ∈ elems pevs(p) ⇒
        case ev of
          mkSe(q,exp)→q ∈ dom pevs\{p}∧wf_Exp(exp)(pdecls(p)),
          mkRe(q,var)→q ∈ dom pevs\{p}∧wf_Var(var)(pdecls(p)),
          mkAct(act)→wf_Act(act)(pdecls(p))
        end
    end
```

Annotation:

wf_Evs: All events are well-formed (with respect to source,target processes, expressions, etc.)

- Sends and receives are between different instances, that is, processes.
- Corresponding expressions and variables are well-formed.
- Internal actions are well-formed.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
  wf_Var: Var → VarDecl → **Bool**
  wf_Var(var)(vardecl) ≡ var ∈ **dom** vardecl

  wf_SendRecv: Cht → **Bool**
  wf_SendRecv(cht) ≡
    Well_Matched(cht) ∧ All_Matched(cht) ∧ ∼is_cyclic(cht)

Annotation:

wf_SendRecv: The send-receive matching relation is well-formed. ■

value
  is_cyclic: Cht → **Bool**
  is_cyclic(cht) ≡ ... /∗ trivial ∗/

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
  Well_Matched: Cht → **Bool**
  Well_Matched(cht) ≡
    let (pevs,sendrecv) = cht **in**
    card dom sendrecv = **card rng** sendrecv ∧
    ∀ (p,i),(q,j):P × Pos•sendrecv((p,i)) = (q,j) ⇒
      ∃ exp:Exp,var:Var•
        pevs(p)(i)=(q,exp) ∧
        pevs(q)(j)=(p,var) ∧
        typeof(exp)=typeof(var)
    end

Annotation:

Well_Matched: The matching is proper. ■

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
  All_Matched: Cht → **Bool**
  All_Matched(cht) ≡
    let (pevs,sendrecv) = cht **in**
    dom sendrecv = {(p,i)|(p,i):P×Pos · is_Send_Ev(pevs(p)(i))}
    end

Annotation:

All_Matched: All send/receive events are matched. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

is_Send_Ev: Ev → **Bool**

is_Send_Ev(ev) ≡ **case** ev **of** mkSe(_,_) → **true**, _ → **false end**

Annotation:

is_Send_Ev: The event must be a send event. ■

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Dynamic Semantics, Types

Semantic Types

type

$P\Psi = P \underset{m}{\rightarrow} \Psi$

$\Psi = \Pi \times \Sigma \times \Theta$

Annotation:

PΨ : The current "stage" of a CTP program is given by associating each process, a "stage", Ψ.

Ψ**:** The process state consists of a triple: the current program point, Π, the current values of all its variables, Σ, and the (evaluated) values of expressions of executed output (send) events, Θ. ■

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

type

$\Pi == \mathrm{mkS(s:S)} \mid \mathrm{mkT(t:T,chtn:Chtn,i:Pos)}$

$\Theta = \mathrm{Pos} \underset{m}{\rightarrow} \mathrm{VAL}$

$\mathrm{Pos} = \mathbf{Nat}$

Annotation:

Π : The program pointer (of a process) either designates a process control state mkS(s:S) or a position i:Pos within a transaction chart chtn:Chtn of a transaction schema t:T;

i=0 i=0 indicates that the process has just entered the chart.

Θ**:** The input/output queue is related to the position, Pos, of the input/output event and holds a value VAL.

Pos: position of the input/output event. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

type

$$\mathrm{P}\Delta = \mathrm{P} \underset{m}{\rightarrow} \Delta$$

Annotation:

PΔ : For each (invoked) process P we record their stepwise progress Δ of that process.

■

  E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

type

$\Delta = \mathrm{T} \times \mathrm{Chtn} \times \Phi$

$\Phi == \mathrm{mkEnter} \mid \mathrm{mkEv(i:Pos)} \mid \mathrm{mkExit}$

Annotation:

Δ : The stepwise progress within a transaction chart, Chtn, of a transaction schema, T, is recorded by a quantity Φ.

Φ : Either the process , at an instance, is at the point of entering, mkEnter, or leaving, mkExit, or is at some event position, mkEv(i:Pos).

i=0 indicates that the chart has just been entered. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Well-formedness

value
  wf_PΔ: PΔ → Prog → **Bool**
  wf_PΔ(pδ)(prog) ≡
    let (pdecls,_,_,_) = prog **in**
    dom pδ ⊆ **dom** pdecls ∧
    ∀ p:P•p ∈ **dom** δ ⇒ wf_Δ(p)(pδ)(prog)
    end

Annotation:

wf_PΔ :

- The invoked processes must first have been declared.
- And for each such process its progress must be well-formed. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
wf_Δ: P → PΔ → Prog → **Bool**
wf_Δ(p)(pδ)(prog) ≡
  let (pdecls,tdecls,_,_) = prog, (t,chtn,φ) = pδ(p) **in**
  t ∈ **dom** tdecls ∧ chtn ∈ **dom** tdecls(t) ∧ p ∈ participants(t)(prog) ∧
  case φ **of**
    mkEv(i:Pos)
      → **let** (pevs,_) = tdecls(t)(chtn) **in** i ∈ **inds** pevs(p) **end**
    _ → ∀ q:P•q ∈ participants(t)(prog) ⇒ pδ(q) = pδ(p)
  end end

Annotation:

wf_Δ : For the invoked process

- the designated transaction schema and transaction chart (of that schema) must be declared, and the designated process (name) must be an instance of that chart.
- In addition the program point (ppt) must be well-formed:
  - ⋆ if an event index it must be into the process instance, otherwise
  - ⋆ all processes of that transaction chart must be in the same (either entry or exit) state. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Dynamic Semantics, Functions

Auxiliarly Functions

value
  xtr_preS: Prog → T → P → S
  xtr_preS(prog)(t)(p) ≡
    let (_,_,wiring,_) = prog **in**
    let (s,_) = wiring(t)(p) **in** s **end end**
    pre t ∈ **dom** wiring ∧ p ∈ **dom** wiring(t)

Annotation:

xtr_preS : Extract from a transaction schema, the precondition (a control state) corresponding to a process. ■

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
  xtr_postS: Prog → T → P → S
  xtr_postS(prog)(t)(p) ≡
    let (_,_,wiring,_) = prog **in**
    let (_,s) = wiring(t)(p) **in** s **end end**
    pre t ∈ **dom** wiring ∧ p ∈ **dom** wiring(t)

Annotation:

xtr_postS : Given a

- program, a transaction schema (name) and a process (name)
- yield the output control state (from the wiring).

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

xtr_Ev: Prog → (T × Chtn × P × Pos) → Ev

xtr_Ev(prog)(t,chtn,p,i) ≡

let (_,tdecls,_,_) = prog **in** **let** (_,(pevs,_)) = tdecls(t)(chtn) **in**

pevs(p)(i) **end** **end**

pre t ∈ **dom** tdecls ∧ chtn ∈ **dom** tdecls(t) ∧

p ∈ **dom** pevs ∧ i ∈ **inds** pevs(p)

Annotation:

xtr_Ev : Given

- a program,
- a transaction schema name (within that program),
- the name of a chart (within that schema),
- a process (name) and
- a position (within the designated chart),

yield the designated event. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
xtr_Prop: Prog → (T × Chtn) → P → Prop
xtr_Prop(prog)(t,chtn)(p) ≡
let (_,tdecls,_,_) = prog **in**
let (gd,_) = tdecls(t)(chtn) **in** gd(p) **end end**
pre t ∈ **dom** tdecls ∧ chtn ∈ **dom** tdecls(t)

Annotation:

xtr_Prop :

- Given
  - ⋆ a program,
  - ⋆ a transaction schema name (within that program),
  - ⋆ the name of a chart (within that schema), and
  - ⋆ a process (name)
- yield the designated proposition.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
last_Pos: Prog → (T × Chtn) → P → Pos
last_Pos(prog)(t,chtn)(p) ≡
  let (_,tdecls,_,_) = prog **in**
  let (_,(pevs,_)) = tdecls(t)(chtn) **in len** pevs(p) **end end**
  pre t ∈ **dom** tdecls ∧ chtn ∈ **dom** tdecls(t)

Annotation:

last_Pos :

- Given
  - a program,
  - a transaction schema (name, within that program),
  - a chart (name, withing that schema), and
  - a process (name)
- yield the position of the last event of the designated process instance. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
  xtr_Send: Prog → (T × Chtn) → (P × Pos) → (P × Pos)
  xtr_Send(prog)(t,chtn)(p,i) **as** (q,j)
    pre
      let (_,tdecls,_,_) = prog **in**
      t ∈ **dom** tdecls ∧ chtn ∈ **dom** tdecls(t) ∧
      let (_,(pevs,_)) = tdecls(t)(chtn) **in** i ∈ **inds** pevs(p) **end end**
    post
      let (_,tdecls,_,_) = prog **in**
      let (_,(_,sendrecv)) = tdecls(t)(chtn) **in**
      sendrecv((q,j)) = (p,i) **end end**

Annotation:

xtr_Send : Extract the matching send event, given a receiving event.

- The transaction schema and chart names must be declared and the event position be appropriate.
- The matching send event (q,j) is then found from the send-receive mapping.■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Initialization

value
init_PΨ: Prog → PΨ
init_PΨ(prog) ≡
  let (_,_,_,init) = prog **in**
  [p↦convert_Ψ(init(p))|p:P•p ∈ **dom** init] **end**

convert_Ψ: (S × VarInit) → Ψ
convert_Ψ(s,varinit) ≡ (mkS(s),varinit,[ ])

Annotation:

init_PΨ : To initialise a program is to create the collection of all process initial states.

convert_Ψ : Mark the initial control state, use the initial control vaiable values and set the initial queues of values of expression of send events to empty. ■

  E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Enabling

value

is_enabled: PΔ → (Prog × PΨ) → **Bool**

is_enabled(pδ)(prog,pψ) ≡

∀ p:P•p ∈ **dom** pδ ⇒ **let** (t,chtn,ϕ) = pδ(p) **in**

case ϕ **of**

mkEnter → is_enabled_Enter_Chtn(t,chtn)(prog,pψ),

mkExit → is_enabled_Exit_Chtn(t,chtn)(prog,pψ),

mkEv(i) → is_enabled_Ev(t,chtn,p,i)(prog,pψ)

end end

pre wf_PΔ(pδ)(prog)

Annotation:

is_enabled : A program step, pδ, is enabled at the current stage of the program, if every process step corresponding to processes in the domain of this program step is enabled:

- either all are enabled for entering or all are enabled for leaving the chart,
- or all are enabled for an event in that state. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

is_enabled_Enter_Chtn: (T × Chtn) → (Prog × PΨ) → **Bool**
is_enabled_Enter_Chtn(t,chtn)(prog,pψ) ≡
∀ p:P•p ∈ participants(t)(prog) ⇒
let s = xtr_preS(prog)(t)(p),
pr = xtr_Prop(prog)(t,chtn)(p),
(π,σ,_) = pψ(p) **in**
(π=mkS(s)) ∧ eval_Prop(pr)(σ) **end**

Annotation:

is_enabled_Enter_Chtn : A chart of a transaction schema can be entered if for every process participating in this transaction schema, its current control state is the precondition of this transaction schema, and the proposition associated with this process in the guard associated with this chart evaluates to true with respect to the current values of variables. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

is_enabled_Exit_Chtn: (T × Chtn) → (Prog × PΨ) → **Bool**
is_enabled_Exit_Chtn(t,chtn)(prog,pψ) ≡
  ∀ p:P•p ∈ participants(t)(prog) ⇒
   let (mkT(t,chtn,i),σ,_)=pψ(p) **in** i=last_Pos(prog)(t,chtn)(p) **end**

Annotation:

is_enabled_Exit_Chtn : A chart of a transaction schema can be exited if for every process participating in this transaction schema, it has executed all its events in this chart. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
  is_enabled_Ev: (T × Chtn × P × Pos) → (Prog × PΨ) → **Bool**
  is_enabled_Ev(t,chtn,p,i)(prog,pψ) ≡
    let (mkT(t,chtn,i−1),_,_) = pψ(p) **in**
      case xtr_Ev(prog)(t,chtn,p,i) **of**
        mkRe(q,_) →
          let (q,j) = xtr_Send(prog)(t,chtn)(p,i) **in**
          let (mkT(t,chtn,j′),_,_) = pψ(q) **in** j ≤ j′ **end end**
        _ → **true**
    end end

Annotation:

is_enabled_Ev : An event at a position of a process in a chart of a transaction schema is enabled, if this process has come to the previous position, and in case this event is a receive event, the matching send event has been executed. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Firing

value
  fire: $(\mathrm{Prog} \times \mathrm{P}\Psi) \to \mathrm{P}\Delta \to (\mathrm{Prog} \times \mathrm{P}\Psi)$
  fire(prog,pψ)(pδ) **as** (prog,pψ')
    pre enabled(pδ)(prog,pψ)
    post pψ'=pψ†[p$\mapsto$upd_Ψ(prog,pψ)(pδ)(p)|p $\in$ **dom** pδ]

Annotation:

fire : Firing an enabled program step updates the current stage of every process. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value
upd_Ψ: (Prog × PΨ) → PΔ → P → Ψ
upd_Ψ(prog,pψ)(pδ)(p) ≡
  let (π,σ,θ) = pψ(p), (t,chtn,ϕ) = pδ(p) **in**
   case ϕ **of**
    mkEnter → (mkT(t,chtn,0),σ,[ ])
    mkEv(i) →
     let σ' = upd_Σ(prog,θ)(p)(t,chtn,i),
       θ' = upd_Θ(prog,θ)(p)(t,chtn,i) **in**
     (mkT(t,chtn,i),σ',θ') **end**
    mkExit → **let** s = xtr_postS(prog)(t)(p) **in** (mkS(s),σ,[ ]) **end**
  end end
pre ...

Annotation:

upd_Ψ : Upon firing an enabled program step, the current stage of a process should be updated as follows.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

- If this process enters a chart of a transaction schema, then this process goes to position zero of this chart (in this transaction schema), retains the current values of variables and initializes an empty map of positions to values of expressions of send events.
- If this process executes an event at a position of a chart of a transaction schema, then this process goes to this position and updates the current values of variables and the map of positions to values of expressions of send events.
- If this process exits a chart of a transaction schema, then this process goes to the postcondition associated with this process of this transaction schema, retains the current values of variables and empties the map of positions to values of expressions of send events. ■

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

upd_Σ: (Prog × PΨ) → P → (T × Chtn × Pos) → Σ

upd_Σ(prog,pψ)(p)(t,chtn,i) ≡

let (_,σ,_) = pψ(p), ev = xtr_Ev(prog)(t,chtn,p,i) **in**

case ev **of**

mkSe(q,exp) → σ

mkRe(q,var) → **let** (_,_,θ) = pψ(q), (q,j) = xtr_Send(prog)(t,chtn)(p,i) **in** σ † [var ↦ θ(j)] **end**

end end

pre ...

Annotation:

upd_Σ : Upon execution of an event, the current value of variables should be updated as follows.

- Executing a send event does not change the values of any variable.
- Executing a receive event amounts to assigning the value of the expression of the matching send event to the variable associated with this receive event, and leaving the values of all other variables untouched.
- Executing an internal action amounts to evaluating it with respect to the current values of variables, possibly leading to changes in the values of variables. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

value

upd_Θ: (Prog × PΨ) → P → (T × Chtn × Pos) → Θ
upd_Θ(prog,ψ)(p)(t,chtn,i) ≡
  let (_,σ,θ) = pψ(p) **in**
  case ev **of** mkSe(q,exp) → θ ∪ [i ↦ eval_Exp(exp)(σ)],
      _ → θ **end end**

pre ...

Annotation:

upd_Θ : Upon execution of an event, the map of positions to values of expression of send events is updated as follows. Executing a send event amounts to adding to this map the value of the expression of this send event associated with its position. Executing a receive event or an internal action does not touch this map. ■

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

The Original CTP Paper

The CTP Paper

- DB has handed out the 10 page conference version of the cited CTP paper.
  - ⋆ DB goes through this paper by asking attendees to “thumb” through the paper.
  - ⋆ DB points out (mathematical) texts
  - ⋆ while commenting on these texts.
- DB recounts the story on attempts to analyse the cited paper.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Questions about CTP

1. What is the need of the restriction for "free choice"? Condition (1), just before Definition 1.

2. In the definition of CTP, variables are not specified explicitly.

   - Should the initial state of a CTP consists of initial control states and values of variables (instead of truths of atomic propositions)?
   - In the examples in the paper, atomic propositions are also used as Boolean variable of processes. It might be good to make a distinction.
   - Correspondingly, in the Petri net semantics, do we have to deal with variables explicitly?

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

3. Do processes have to jointly enter a transaction schema?

- The Petri net semantics suggests it is not the case.
- In other words,
  - ⋆ if t1,t2 are two transaction schemas such that p,q participate in t1, and q,r participate in t2,
  - ⋆ then it is possible to p to enter t1 (without waiting for q) and q to enter t2 (without knowing p has entered t1) simultaneously.
- But this differs from the informal semantics of the high level condition event Petri net.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

4. In a transaction schema, do the processes jointly choose a chart and then execute that chart?
   - Or could they make up their mind on which chart to execute as they go along in the transcation schema?
   - The former seems to be suggested by informal description of semantics and the latter by the Petri net semantics.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

5. Do processses have to synchronize (i.e. wait for each other) when exiting from a transaction schema?
6. In a transaction schema, events in different charts with isomorphic history are collapsed into one single class.
   - Is this essential or just for efficiency?
   - i.e. can't we just translate each single chart into a Petri net?
7. Suppose in some transaction schema, there are events in different charts with a large isomorphic history.
   - Does it mean that this transaction schema is not well-specified,
   - i.e. should it be decomposed into smaller transaction schemas?
8. Does the current implementation of CTP follow "closely" the Petri net semantics?

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Inconsistencies of the CTP Journal Paper

1. The implementations of the CTP language consist of several tools that translate CTP programs to SystemC, Verilog, HandelC codes.
   - It is not known whether these translators follow “closely” the Petri net semantics of CTP.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

2. The free-choice control flow restriction, that is, condition (1) (preceding definition 1) of page 6, does not achieve what its claimed purpose as in line 2 of page 6.
   (a) The claimed purpose of condition (1) is that at every local state of a process p, the choices as to which transaction schema that p will take part in is decided locally by p.
   (b) The definition of condition (1) however only says that at every local state of a process p, no matter which transaction schema p chooses to take part in, p will land at the same next local state.
   (c) It is not known whether this restriction is demanded in the various CTP implementations.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

3. In the Petri net semantics, during the execution of a CTP program, one keeps track of only the truths of some atomic propositions about the values of variables, instead of the exact values of variables.
   (a) However, one need the exact values of variables for execution of the send, receive and internal actions in a message sequence chart.
   (b) In fact, in the CTP to SystemC translator tool, exact values of variables are used.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

4. Corresponding to that exact values of variables should be kept track of instead of atomic propositions, the definition of the "blow-up" Petri net need to handle variables explicitly.
   - Actually it would be simpler to define the "blow-up" Petri net just as a colored Petri net instead of an elementary net system (as also pointed out in the CTP journal paper).

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Discussion
What's the Problem?

- The way I see it, the problem is
  - ⋆ that it is possible to formulate,
    - ◇ what I claim to be far more precise,
    - ◇ in fact formally precise definitions of, in this case CTP
    - ◇ than in the conventional "classico-mathematical" style.
  - ⋆ But that it is not done!

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Two Disciplines of Computers

- Computer science and computing science are two closely related disciplines.
- **Computer science**
  - ⋆ is the study and knowledge of what may exist inside computers: data and processes,
  - ⋆ and usually computer science papers are couched in "classico-mathematics".
- **Computing science**
  - ⋆ is the study and knowledge of how to construct the artifacts that exist inside computers: data and processes,
  - ⋆ and usually computing science papers are couched in some refinement calculus, in B, VDM-SL, RSL, Z or other.

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Hindrances to Pro$\frac{\text{gress}}{\text{pagation}}$

- Our CS department have staff of both kinds— and that is good.
- But usually the computer scientists are refinement calculi, B, VDM-SL, RSL and Z illiterates — and that is bad.
- The computing scientists are usually well-trained in the topics and notations of computer science.
- How can students of computing science lecturers take formal methods serious when our computer scientist colleagues do not?

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

Conclusion

- B or VDM-SL or RSL or Z, integrated with Petri Nets and/or Message or Live Sequence Charts and/or Statecharts and/or TLA+ or Duration Calculus
- can be used to formalise domain, requirements and design of
  - ⋆ programming languages and their compilers,
  - ⋆ operating systems,
  - ⋆ database management systems and databases,
  - ⋆ distributed systems,
  - ⋆ etc.
- In consequence we ought to teach these topics based on formal models.
- The Danish CHILL and Ada compilers were so developed (early 80s).
- Wolfgang Paul, Saarbrücken, is now doing it — verification included.
- We should all do it!

ShaoFa Yang & Dines Bjørner
NUS, Singapore & DTU, Denmark
May–June + Nov.–Dec., 2005
Work in progress

A Very Merry Christmas

And A Happy New Year

E-mail: bjorner@gmail.co, dines@bjorner.biz, URL: www.imm.dtu.dk/~db