ccp and event structures: towards a true concurrency semantics of stochastic, real-time ccp

Camilo Rueda [1]

[1] Universidad Javeriana-Cali

November 26, 2007

The ccp model

- A collection of agents
- Agents compute partial information
- Information is recorded in a store
- the store grows monotonically
- Agents operate concurrently
- they synchronize by shared variables

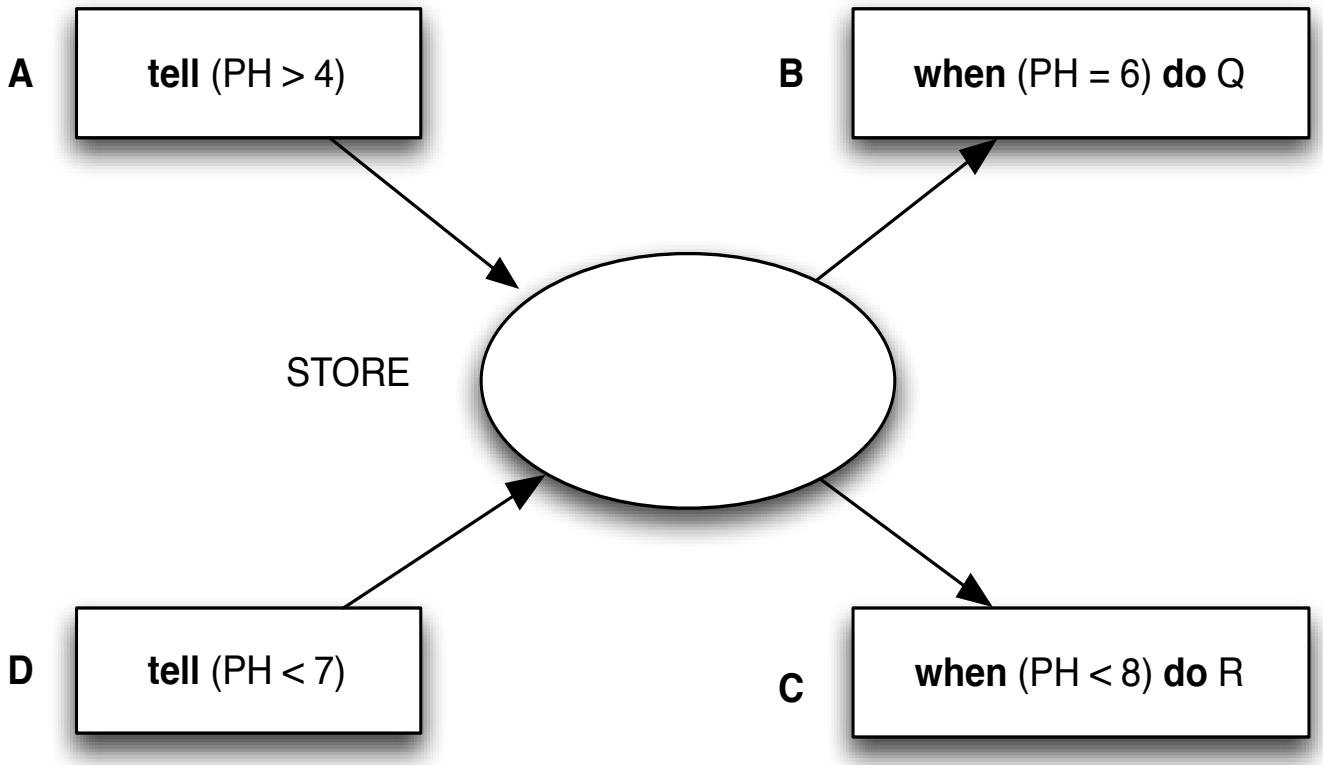
Agents

Two kinds of agents:

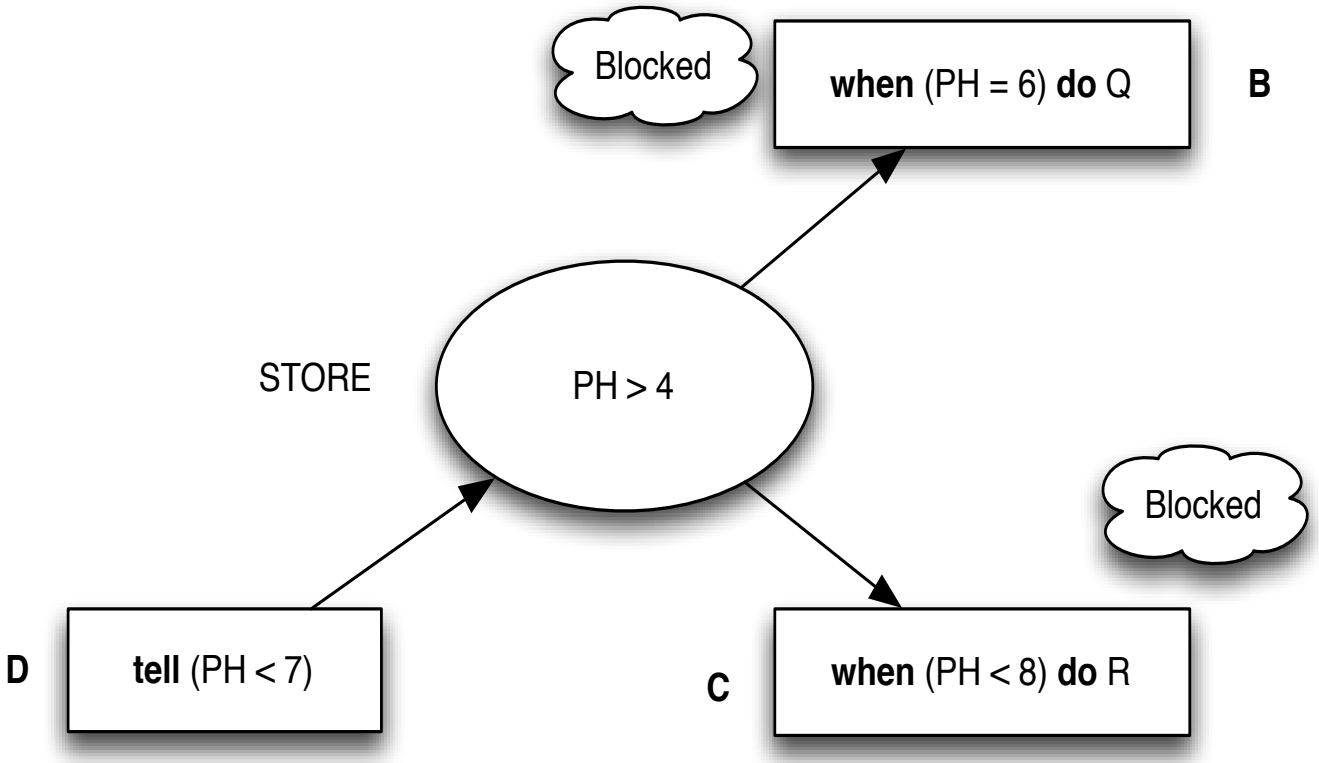
- tell agents add information to the store
- ask agents deduce information from the store

A computation is the activity resulting from the interaction of these two types of agents

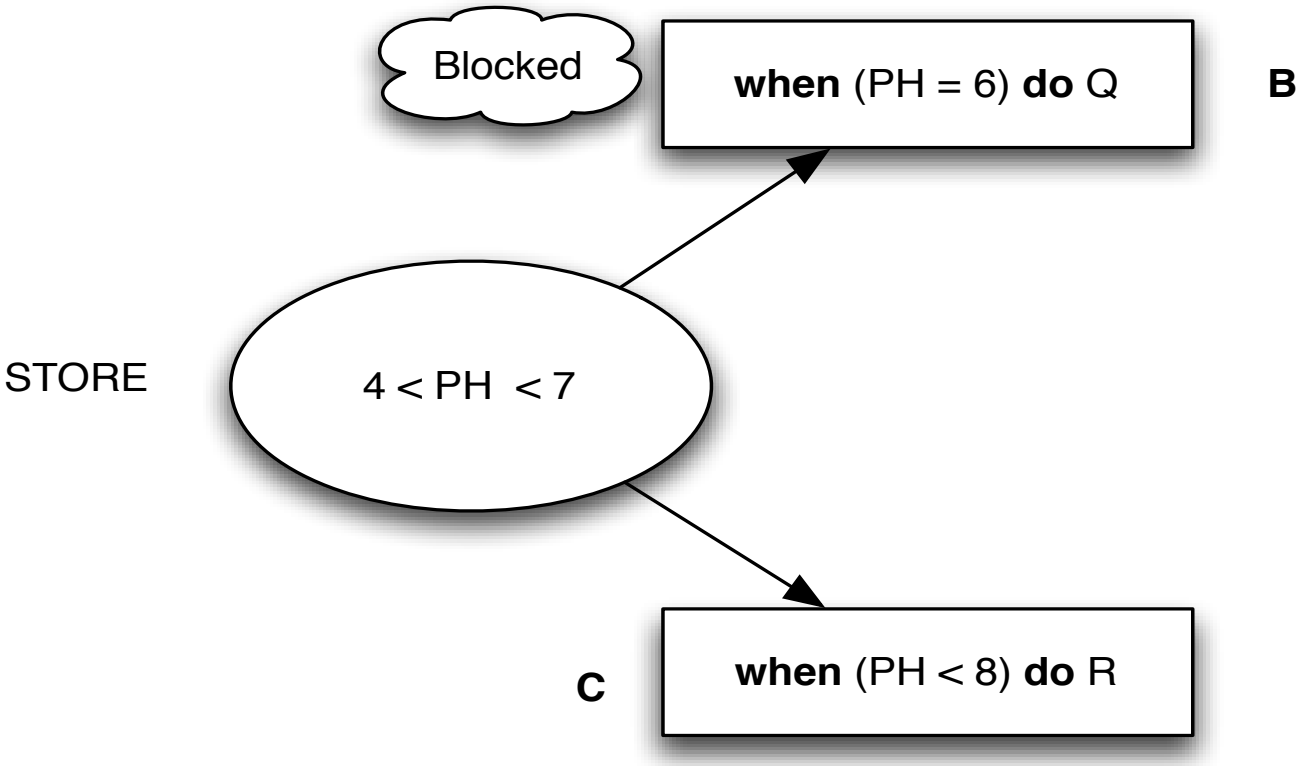
Computing with partial information in CCP



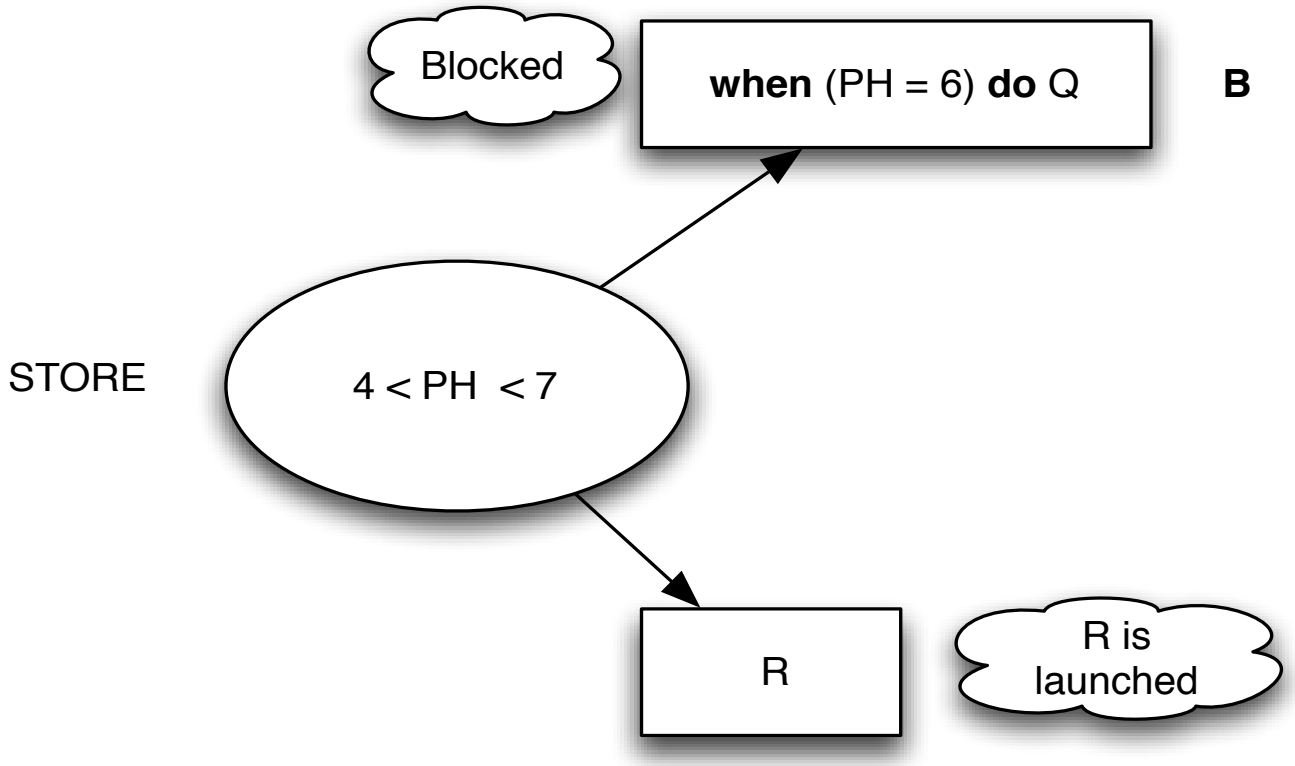
Computing with partial information in CCP



Computing with partial information in CCP



Computing with partial information in CCP



Concurrency

What does it mean agents *A* and *B* are concurrent?

- that they compute simultaneously?
-

Concurrency

What does it mean agents *A* and *B* are concurrent?

- that they compute simultaneously?
  A single observer (one absolute clock)?
-

Concurrency

What does it mean agents *A* and *B* are concurrent?

- that they compute simultaneously?
  A single observer (one absolute clock)?
  all observers read the same time for *A* and *B* in their clocks?
-

Concurrency

What does it mean agents *A* and *B* are concurrent?

-
- that they compute in some (unknown) order?

Concurrency

What does it mean agents *A* and *B* are concurrent?

-
- that they compute in some (unknown) order?
  the order is the same for all observers?

Concurrency

What does it mean agents *A* and *B* are concurrent?

-
- that they compute in some (unknown) order?
  the order is the same for all observers?
  the computation is the same no matter the order?

The standard ccp model

- accumulation of concurrent tells
  $tell\ (x > 3) \parallel tell\ (y \neq 0) \equiv tell\ (y \neq 0) \parallel tell\ (x > 3)$
  $\ldots \equiv tell\ (x > 3 \wedge y \neq 0)$
- interleaving of asks and tells
  $ask\ z < 1 \rightarrow P \parallel ask\ w = 5 \rightarrow Q$
  $\equiv$
  $ask\ z < 1 \rightarrow (P \parallel ask\ w = 5 \rightarrow Q\ )$
  $+$
  $ask\ w = 5 \rightarrow (\ Q \parallel ask\ z < 1 \rightarrow P\ )$
- grouping of asks
  $ask\ z < 1 \rightarrow\ (ask\ w = 5 \rightarrow P)$
  $\equiv\ \ ask\ (z < 1 \wedge w = 5) \rightarrow P$

Problems with the properties

nice properties, but:

they combine in such a way that intuitions do not scale up!

doing *A* and *B* concurrently might be intuitively the same as doing them sequentially in any order when *A* and *B* are not complex agents.

intuition accepts interleaving only for atomic processes

The determinate language

- A program is a declaration followed by an agent
  $P ::= L.A$
- Procedure declarations
  $L ::= \epsilon \mid p(X) :: A \mid L.L$
- Agents, $A ::=$
  - tell
    c
  - ask
    $c \rightarrow A$
  - Concurrent composition
    $A \parallel B$
  - Information hiding
    $\exists X.A$
  - procedure call
    $p(X)$

The determinate language: example

$$
\begin{array}{l}
append(In1, In2, Out) :: \\
\quad In1 = \lambda \rightarrow Out - In2 \\
\| \\
\quad c(In1) \rightarrow \exists X \; (Out = a(f(In1), X) \parallel append(r(In1), In2, X)
\end{array}
$$

Semantics: basic notions

- A set D of tokens or basic constraints
- $\vdash$, a decidable compact entailment relation
- non basic constraints are sets of tokens closed under entailment
  - $|D|$ is the set of finite non basic constraints
  - $c \in |D|$ implies that when $c \vdash P$ then $P \in c$
- $u \vdash P$ means: from constraint u, (basic) constraint P can be inferred

Semantics of the determinate language

- the only thing that deserves being observed is the store
- processes add information to the current store
- so they transform one store into another
- Since the language is determinate, transformation is a function:
  $f : |D| \rightarrow |D|$

Processes as constraint transformers

Say process *f* transforms constraint *c* (the store) into *d* and then stops. i.e. constraint *d* is a resting point.

- store *d* has at least as much information as *c*. i.e. $d \geq c$ (extensive)
- then *f* transforms *d* into *d* (idempotent)
- information is never substracted. i.e. $c \leq d \Rightarrow f(c) \leq f(d)$ (monotonicity)

Each language construct is a constraint transformer

Constraint transformers of tells and asks

We will focus only on ask and tell processes

- resting points of a $tell(c)$ is the set of constraints stronger than c

$$tell(c) = \{d \in |D| \mid d \geq c\}$$

- resting points of pprocess $c \rightarrow f$ is the set of constraints in the resting points of f that are stronger than c

$$(c \rightarrow f) = \{d \in |D| \mid d \geq c \Rightarrow d \in f\}$$

notice that constraints weaker than c are also included

Constraint transformers of parallel composition

- resting points of $P \parallel Q$ is the intersection of the resting points of P and Q

$$P \parallel Q = f_P \cap f_Q$$

Nondeterminate language: choice

$$\textit{choice}: \boxed{c_1 \rightarrow A_1 \ + \ c_2 \rightarrow A_2 \ + \ \ldots + \ c_n \rightarrow A_n}$$

In a store d, process

$$\sum_{j \in J} (c_j \rightarrow A_j)$$

transforms into

$$A_j \qquad \textit{if } d \geq c_j, \textit{ for some } j \in J$$

The store does not change.

choice

- choice adds expressivity to the language
- is at the heart of models in many applications
- but. . . it complicates semantics

For a given store, in a choice construct,

- many guards could be enabled
- computation may proceed along different paths

therefore,

processes are no longer constraints transformation functions

Semantics: idea

Since computation paths are important, for each process

- as before, record its resting points
- but also the interactions to get there

The idea is to keep a trace

$$a_1 \rightarrow (b_1 \parallel (a_2 \rightarrow b_2 \ldots (a_n \rightarrow b_n) \ldots)$$

of the path to reach resting point $c = b_1 \wedge \ldots \wedge b_n$. A process is thus

A set of pairs (t, c),
one for each resting point c

Semantics: trace operators

traces can be represented as bounded closure operators (f, u):

- u is a resting point of f (i.e. $u \in f$)
- the bound (i.e. domain) of f is $\downarrow u$

bco's can be ordered

$$(f, u) \leq (g, u) \text{ if } f \supseteq g$$

the (standard) semantics of a ccp process will then be

a set of bounded closure operators

True concurrency

for true concurrency

- the accummulation laws should not hold
- the interleaving law should not hold

observations should be more fine-grained:

- observe just tokens (i.e. basic constraints) in the store
- and also the conditions enabling them to get there

The observations

- Observations are contexted stores
- A contexted store is a set of contexted tokens a^u

  in the presence of tokens in u, token a is output

- contexted tokens are represented by bounded closure operators

Example

not(Modo, In, Out)::
$(Modo = ok \rightarrow Out = not(In)$
$+ Modo = trabado\_en\_1 \rightarrow Out = 1$
$+ Modo = trabado\_en\_0 \rightarrow Out = 0)$

Some contexted tokens in the presence of $M1 = ok, M2 = ok, X1 = 0, X2 = 1$ are:

$$Y1 = not(X1)^{M1=ok}$$
$$Y1 = 1^{M1=ok,X1=0}$$

And their bounded closure operators:

$$(M1 = ok \rightarrow not(X1), \downarrow not(X1))$$
$$(M1 = ok \rightarrow (X1 = 0 \rightarrow Y1 = 1), \downarrow Y1 = 1)$$

Semantics (Saraswat)

- Tell:

$$[\![tell(a)]\!] : \boxed{\{(f,u)|\ a \in u \wedge f \supseteq \tilde{a}\uparrow\}}$$

- parallel composition

$$[\![P \parallel Q]\!] : \boxed{\{(f \cap g, u)|\ (f,u) \in [\![P]\!] \wedge (g,u) \in [\![Q]\!]\}}$$

- choice,
  $[\![\sum_{j\in J}(c_j \rightarrow A_j)]\!]$:

$$\boxed{\begin{array}{l}\{(u\downarrow, u)|\ \forall j \in J.a_j \notin u\} \\ \cup \\ \bigcup_{j\in J}\{(f,u)|\ a_j \in u \wedge \exists (f_j,u) \in [\![A_j]\!] \wedge f \supseteq a_j \rightarrow f_j\}\end{array}}$$

Considerations about this semantics

- uses eventual tell
- extends nicely from the standard ccp semantics
- laws of interleaving and grouping do not hold

but...

- can it be extended to temporal ccp?
- can it be extended with probabilistic constructs?
- how about atomic tell?

Event structures

In a very general sense what is going on in a concurrent system?

- events occur in some determined or unknown order
- each event performs an action
- the actions performed change the state the system is in

event structures is a framework to represent the above in a precise way

Event structures: definition

An (prime) event structure is composed of

- a set A of events
- a causality (or enabling) relation "$\leq$" between events
- a conflict relation "$\#$" between events

with the properties:

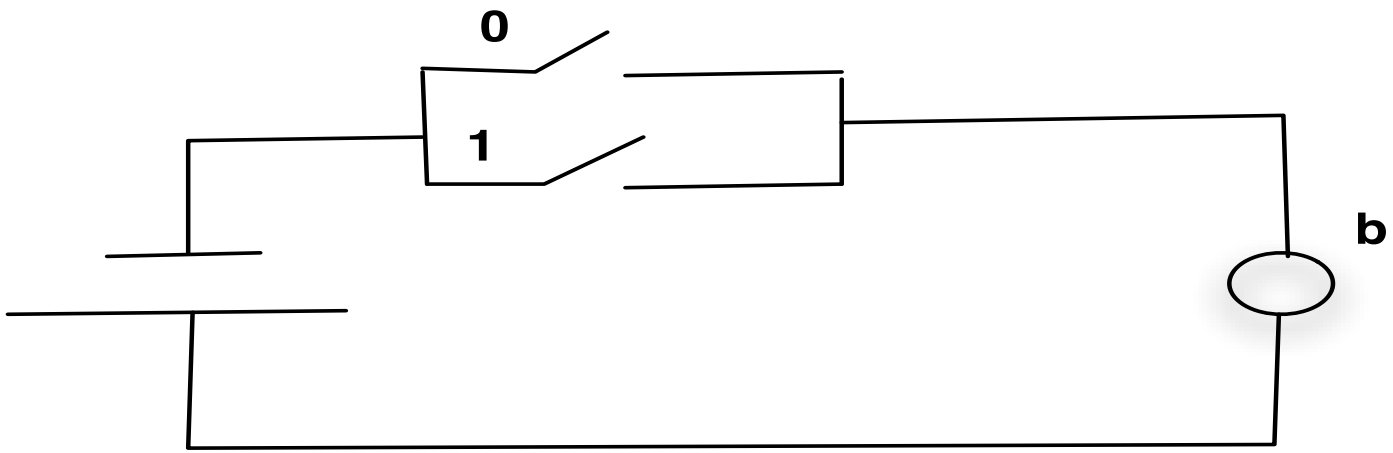
- $(A, \leq)$ is a partial order
- $\#$ is irreflexive and symmetric
- for all events $a, b, c \in A$,
  if a is in conflict with b and b causes c, then a is in conflict with c. i.e.,
  $a \# b \wedge b \leq c \Rightarrow a \# c$

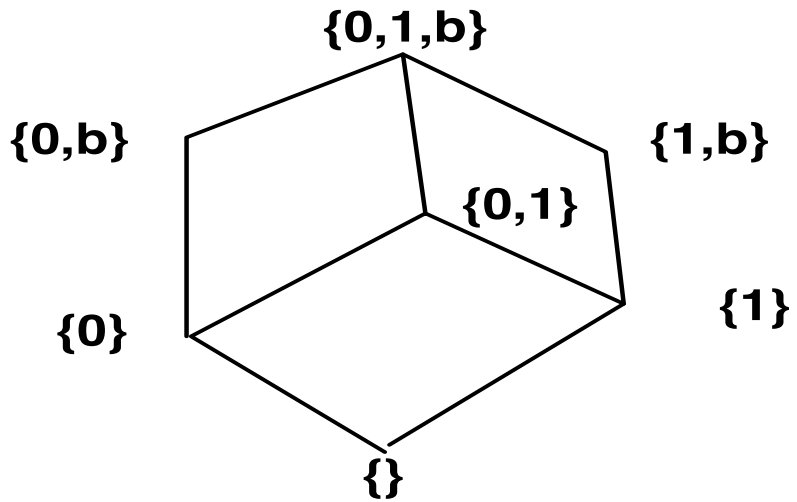
Event structures: the states of a system

States of a system are called configurations of an event structure

- configurations of a system are coherent sets of events
- events in a state mean that those events have already occurred when reaching that state
- therefore, if event b is in state s, and event a enables b, event a must also be in s
- states cannot have conflicting events:
  if $a \in s$ and $a \# b$ then $b \notin s$

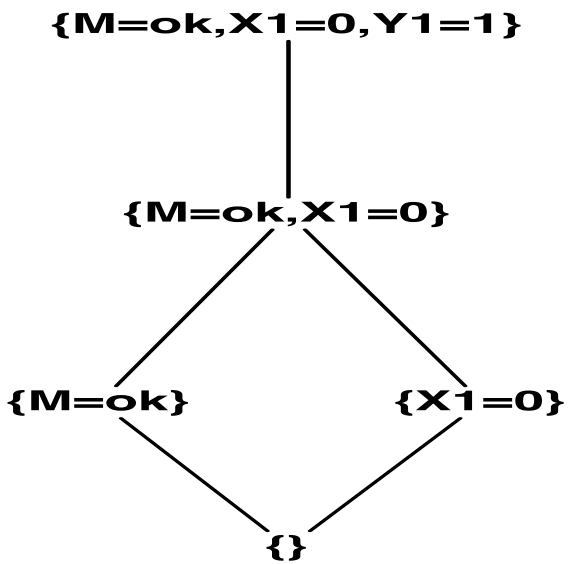
Event structures: an example





Event structures: an example (2)

Contexted tokens can be represented as event structures.
for example,
$(Y1 = 1)^{M1=ok,X1=0}$



so in ccp, events could be tokens, ordered by context?

Event structures and contexted stores

In a more precise way,

- events are occurrence of actions
- the action of each event is a set of contexted stores
- contexted stores are sets of tokens

For example,

$$\mathit{ask}\ M = \mathit{ok} \rightarrow \mathit{ask}\ X = 0 \rightarrow Y = 1$$

has events $E = \{e_1, e_2\}$ and

$$\begin{array}{l} \mathit{action}(e_2) = \ (Y = 1)^{\{\{X=0\}\}} \\ \mathit{action}(e_1) = \ (Y = 1)^{\{\{M=\mathit{ok},X=0\}\}} \end{array}$$

Representation of event structures: chu spaces

A chu space or "couple" is a matrix (A, X, Σ)

- Set A, the rows, is a set of events
- Set X, the columns, is a set of states

when $\Sigma = \{0, 1\}$,

- A system as a chu space gives, for each state, the events that have occurred
- dually, for each event, the states in which it has occured

Both events and states can then be regarded as functions

$a(s) = 1$ means "event a has occurred in state s"

or, equivalently,

$s(a) = 1$ means "event a belongs to configuration s"

Representation of event structures: chu spaces (2)

An event structure can be easily recovered from a chu space:

- given a chu space $\mathbb{S} = (A, X)$ with $x \in A \to \{0, 1\}$
- the event structure representd by $\mathbb{S}$ is $(A, \leq, \#)$ where
- $a \leq b$ if, in every state s, when b is in s, a is also in s
- and $a \# b$ if, in all states s,
  - when a is in s, event b is not in s,
  - and, when b is in s, then a is not in s

chu space of **tell** and **ask**

$tell(a)$: e_a | 0 1 | $act(e_a) = a^{\{\emptyset\}}$

$ask\ a \rightarrow b$: e_b | 0 1 | $act(e_b) = b^{\{\{a\}\}}$

chu space of composition and choice

$tett(a) \parallel tell(b)$:

e_a	0	1	0	1
e_b	0	0	1	1

$\{\{\}, \{a\}, \{b\}, \{a, b\}\}$

$tell(a) + tell(b)$:

e_a	0	1	0
e_b	0	0	1

$\{\{\}, \{a\}, \{b\}\}, \quad a \# b$

Defining combinators

- tell(a) simply adds contexted store $a^{\{\emptyset\}}$
- $a \rightarrow b$ adds contexted store $b^{\{\{a\}\}}$
- parallel composition $P \parallel Q$ states all possible combinations of contexted stores in P and Q
- choice $P + Q$ filters $P \parallel Q$ by precluding non common tokens of P and Q to be in the same configuration

Defining composition

$[\![P \parallel Q]\!] \triangleq (A \cup B, Z)$, where

- $Z = \{z \in 2^{A \cup B} \mid z \downarrow_A \in X \wedge z \downarrow_B \in Y\}$
- $z \downarrow_A$ is the restriction of the domain of z to the set A

so,

- for tokens (events) $a \in A, a \notin B$ and $b \in B, b \notin A$, states of z contain $\{a\}, \{b\}$ and $\{a, b\}$
- and for common events of A and B, compatible states are kept.

Since each event is unique, the last option never arises.
Actions of each event carry on to the composition

Defining composition: example

$$(tell(a) \parallel tell(b)) \parallel (tell(b) + tell(c))$$

$a \parallel b$:

e_a :	0	1	0	1
e_b :	0	0	1	1

$b + c$:

e'_b	0	1	0
e_c	0	0	1

$action(e'_b) = b^{\{\emptyset\}}$

$(a \parallel b) \parallel (b + c)$ ::

e_a :	0	1	0	1	0	0	1	1	0	0	1	1
e_b :	0	0	1	1	0	0	0	0	1	1	1	1
e'_b :	0	0	0	0	1	0	1	0	1	0	1	0
e_c :	0	0	0	0	0	1	0	1	0	1	0	1

Notice that four of the above 12 states are redundant

Defining choice

Let the sets of events of P be A and those of Q be B
the event structure of $P + Q$ contains

- states where neither A nor B have occurred

$$\overline{A \cup B}$$

- states of P, together with non occurrence of those events in B not in A

$$P \parallel \overline{B - A}$$

- states of Q, together with non occurrence of those events in A not in B

$$Q \parallel \overline{A - B}$$

Defining choice: formalization

Given a set of events $A = \{a_1, \ldots, a_n\}$, let $\overline{A}$ be the event structure

$$\begin{array}{c|c} a_1 & 0 \\ \ldots & \ldots \\ a_n & 0 \end{array} \qquad \textit{i.e.}(A, \leq, \#) \textit{ with configurations } \{\,\{\}\,\}$$

the semantics of choice is

$$[\![P + Q]\!] \triangleq \overline{A \cup B} \vee (P \parallel \overline{B - A}) \vee (Q \parallel \overline{A - B})$$

where disjunction of processes $P \vee Q$ is defined as

$$(A \cup B, \{z \in 2^{A \cup B} \mid z \downarrow_A \in X \vee z \downarrow_B \in Y\}$$

Defining ask

the structure for $a \rightarrow Q$ contains states of Q but

- each contexted action of Q is augmented with $\{a\}$

formally:

$$[\![a \rightarrow Q]\!] \triangleq [\![Q]\!]$$

and , for each event e of $[\![Q]\!]$ such that $action(e) = b^w$, set

$$action'(e) = b^{w'}$$

where $w' = \{z \cup \{a\} \mid z \in w\}$

Where is the store?

At each state the store is the conjunction of tokens corresponding to event entries equal to 1
but,

what is the definition of conjunction for contexted stores?

A little algebra of contexted store reductions,

$$a^x \wedge a^y \longrightarrow a^{x \cup y}$$
$$a^x \longrightarrow a^{x \setminus \{s\}} \quad \text{if there exists } u \in x \text{ such that } u \subseteq s$$
$$a^x \wedge b^y \longrightarrow b^y, a^{x \setminus \{u\}} \quad \text{if there exists } s \in y \text{ such that } s \cup \{b\} \subseteq u$$

in particular, $a^{\{\emptyset\}} \wedge a^x \longrightarrow a^{\{\emptyset\}}$ and
$a^{\{b\}} \wedge b^{\{\emptyset\}} \longrightarrow b^{\{\emptyset\}}, a^{\{\emptyset\}}$

The store

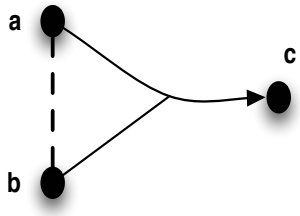
Let s be a state.
Let $\mathcal{C}$ be the colection of events a such that$s(a) = 1$
Let $Clos(\mathcal{C})$ be the contexted stores resulting from applying the reductions to actions of $\mathcal{C}$

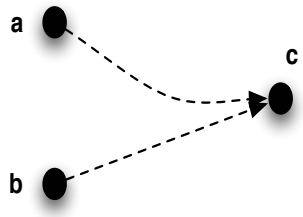
$$store(s) = \boxed{\{\hat{b} \mid b^{\{\emptyset\}} \in Clos(\mathcal{C})\}}$$

where $\hat{b}$ is the entailment closure of b

timed event structures

prime event structures are extended to bundle event structures:

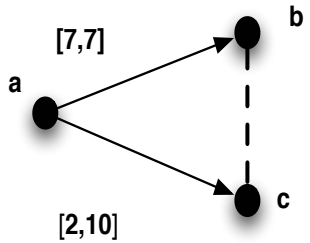


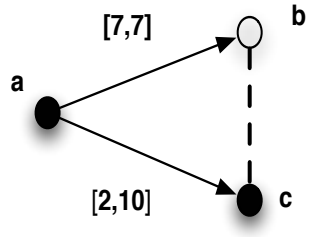


- A set of events
- an irreflexive conflict relation
- a bundle relation

timed event structures

bundle event structures are extended with temporal labels:





- A set of events
- an irreflexive conflict relation
- a bundle relation
- a labelling function
- an event delay function
- a bundle delay function
- a set of urgent events

atomic tell?

we have seen eventual tell. The atomic tell,

- implements test and set operation
- the store can never be inconsistent

$tell(x = 3) \parallel tell(x > 7)$ may reduce to

$$< skip, x = 3 >$$

it would be nice to...

- work out in detail the eventual tell chu-space semantics of ccp
- then the atomic tell
- then integrate temporal labels into the chu-space framework