CSP Syntax

Markus Roggenbach

Feburary 2021

The process algebra CSP

CSP = Communicating Sequential Processes

Selected bibliography

Classical process algebras:
1985 Hoare – Communicating Sequential Processes (CSP)
1989 Milner – Calculus of communicating systems (CCS)
1984 Bergstra, Klop – Algebra of Communicating Processes (ACP)

Important books on CSP:
1998 B Roscoe – The theory and practice of concurrency
2000 S Schneider – Concurrent and Real-time Systems
2004 A E Abdallah et al - CSP – The First 25 Years
2010 B Roscoe – Understanding Concurrent Systems

What is it good for?

Modelling and analysing distributed / parallel systems

Applications include

- Formal Methods for Human-computer Interaction
- Formal Verification of Security Protocols
- Control systems
- Railways
- Algorithms

Elements of today's lecture

- Narrative
  "By a narrative we shall understand a document text, which, in precise, unambiguous language, introduces and describes all relevant properties of entities, functions, events and behaviour of a set of phenomena and concepts, in such a way that two or more readers will basically obtain the same idea as to what is described."
  D Bjoerner: Software Engineering 3: Domains, Requirements, and Software Design, page 78, Springer 2006.
- CSP models of the narrative
- CSP grammar

Further planning

- Friday's lecture: You play with the CSP processes in FDR 4.
- Tuesday, 1st lecture: Semantics – Traces and Transition Systems & What tools can do for you

Hint: ATM models in CSP in HCI

One can model the human Short Term Memory in CSP and investigate, if an ATM design leads to a ‘memory overload’.
Such an overload would make it likely that users forget their cards or their money.
Purpose here: a simple modelling exercise.

Variations of the ATM theme

Narrative 1: Elementary ATM

Initially the ATM shows a ready screen. A customer then proceeds to

- insert a bank card, and
- enter the pin for the card

following which the ATM will

- deliver cash to the customer, and
- return the bank card,

before getting ready for a new session.

Modelling in CSP

First step: identify ‘events’ that define system evolution.

Here:

$$\Sigma = \{\mathit{cardI}, \mathit{pinE}, \mathit{cashO}, \mathit{cardO}, \mathit{ready}\}.$$

ATM0 and ATM1

Single session – ending with STOP:

```
ATM0 = ready -> cardI -> pinE -> cardO -> cashO -> STOP
```

Infinitely many sessions – using recursion:

```
ATM1 = ready -> cardI -> pinE -> cardO -> cashO -> ATM1
```

Observation: CSP distinguishes between events and processes.

- Events are instantaneous and mark system evolution.
- Processes represent system states.

CSP grammar: action prefix, STOP, process equation

CSP grammar defined relatively to an alphabet of events Σ and a set of process names PN.

$$\begin{array}{rcl} P & ::= & Stop \\ & | & a \rightarrow P \\ & | & N \end{array}$$

where $a \in \Sigma$, $N \in PN$.
A process equation takes the form

$$n = P$$

where $n \in PN$ and P as described by the above grammar.

Note: CSP has a blackboard (pretty print) and a tool syntax

Narrative 2: ‘interfaces’

An ATM has several ‘interfaces’:

- there is a `Display`, which can show the `ready` message;
- there is a `KeyPad`, which allows the user to enter the pin;
- there is a `CardSlot`, which takes the card in or gives the card back;
- there is a `CashSlot`, which delivers the money.

ATM2

```
ATM2 = Display.ready -> CardSlot.cardI
     -> KeyPad.pinE -> CardSlot.cardO
     -> CashSlot.cashO -> ATM2
```

Observation: CSP uses channels to ‘structure’ the alphabet.

CSP grammar: channels

Communicating an event e over a channel c adds to the CSP grammar the following primitives

$$\begin{array}{rcl} P & ::= & \ldots \\ & | & c.e \rightarrow P \end{array}$$

Narrative 3

left out

Narrative 4: different choices

Initially the ATM shows a ready screen. A customer then proceeds to insert a bank card, and enters the pin for the card following which the ATM will offer the choice between cash withdrawal and checking the balance.

In case of cash withdrawal, it will deliver the cash to the customer, and return the bank card, and get ready for a new session.

In case of checking the balance, it will display the account balance and return the bank card, and get ready for a new session.

ATM4 – external choice

```
ATM4 = Display.ready -> CardSlot.cardI
     -> KeyPad.pinE -> Display.menu
     ->  ( (Buttons.checkBalance -> Display.accountBalance
           -> CardSlot.cardO -> ATM4)
         []
           (Buttons.withdrawCash -> CardSlot.cardO
            -> CashSlot.cashO -> ATM4)
         )
```

Observation: the environment chooses how to proceed

ATM5 – input, output, conditional

```
ATM5 = Display!ready -> CardSlot.cardI
     -> KeyPad.pinE -> Display!menu
     -> Buttons?x -> if x==checkBalance then BALANCE
                                        else WITHDRAWEL
BALANCE = Display!accountBalance
          -> CardSlot!card0 -> ATM5

WITHDRAWEL = CardSlot!card0
             -> CashSlot!cash0 -> ATM5
```

Observations:

- ? indicates the process gives choice to the environment
- ! indicates the process offers a fixed value
- System states can be denoted by process names

CSP grammar: external choice, conditional

$$
\begin{array}{rcl}
P, Q & ::= & \ldots \\
& | & c?x \to P \\
& | & c!e \to P \\
& | & P \,\Box\, Q \\
& | & \textbf{if } \mathit{cond} \textbf{ then } P \textbf{ else } Q
\end{array}
$$

Here, c is a channel, and *cond* is a condition in a logic of choice (not determined by CSP).

Narrative 5: pin verification; parallel subcomponents

Additionally to the above described functionality that only concerns the dialog with the customer, an ATM includes a subsystem which determines if the entered PIN is a valid one. Should the entered PIN be wrong, the ATM informs the user about this, and returns to the ready state. In case the PIN is valid, the ATM proceeds as normal.

Abstract PIN verification – internal choice

```
PinVerification =
     RequestPCheck -> comparePinWithCard
  ->((Check.pinOK -> PinVerification)
     |~|
     (Check.pinWrong -> PinVerification))
```

Observation: the process internally chooses how to proceed

User Dialog

```
UserDialog =
    Display.ready -> CardSlot.cardI
 -> KeyPad.pinE -> RequestPCheck
 -> ((Check.pinOK -> SERVICES)
     []
     (Check.pinWrong -> Display.messagePinWrong -> UserDialog))

SERVICES= Display.menu -> (CashWithdrawel [] BalanceCheck)

BalanceCheck =
  Buttons.checkBalance -> Display.accountBalance
  -> CardSlot.cardO -> UserDialog

CashWithdrawel =
     Buttons.withdrawCash -> CardSlot.cardO
  -> CashSlot.cashO -> UserDialog
```

Glueing things together

```
ATM6 = UserDialog
       [| {| RequestPCheck  |} |]
       PinVerification

ATM7 = ATM6 \ {| RequestPCheck, Check, comparePinWithCard |}
```

Observations:

- processes can run in parallel
- hiding creates ‘blackboxes’, in which system progress can’t be observed

CSP grammar: internal choice, parallel, hiding

$$
\begin{array}{rcl}
P, Q & ::= & \ldots \\
& | & P \sqcap Q \\
& | & P \,[\!|\, A \,|\!]\, Q \\
& | & P \,[\, A \parallel B \,]\, Q \\
& | & P \,|||\, Q \\
& | & P \parallel Q \\
& | & P \setminus A
\end{array}
$$

where $A, B \subseteq \Sigma$ are a sets of events.

Narrative 6: three attempts to enter the Pin

The ATM behaves as before, but now the customer has three attempts to enter the correct PIN: in case that the PIN is valid, the customer is offered the functionality as described before; in case that the PIN is incorrect and the customer still has an attempt left, the ATM displays that the entered PIN was wrong and gives the customer another opportunity to enter the PIN; in case that the PIN is incorrect and the customer has no attempt left, the ATM displays that the entered PIN was wrong, informs the customer that the card is kept, and returns to the ready screen.

Checking pins

```
UserDialog = Display.ready -> CardSlot.cardI
           -> PINCHECK(3)

PINCHECK(n) = KeyPad.pinE -> RequestPCheck
           -> ((Check.pinOK -> SERVICES)
               []
               (Check.pinWrong
                -> if (n==1)
                   then Display.messagePinWrong
                        -> Display.cardSwallowed
                        -> UserDialog
                   else Display.messagePinWrong
                        -> PINCHECK(n-1)
               )
              )
```

Observation: processes can have parameters

No change of CSP grammar

It is not necessary to expand our CSP grammar at this point. We gave more detail how the elements of the set of process names PN can look like. As the CSP grammar is defined relatively to PN, it needs no change.

Summary

We have seen

- Modelling of ATM, refining narrative and formal model
- CSP syntax – blackboard and tool notation
- CSP allows concise descriptions of complex system