A THEORY FOR COMMUNICATING SEQUENTIAL PROCESSES IN COQ

Carlos Alberto da Silva Carvalho de Freitas
(casc2@cin.ufpe.br)

Supervisor: Gustavo Carvalho

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Introduction

Concurrent systems

- Parallel execution of components
- Deadlock, nondeterminism and other issues
- Testing cannot guarantee properties such as determinism

CSP: a theory for Communicating Sequential Processes

- Clear and accurate description of concurrent systems
- Designs can be proven correct with respect to desired properties

Introduction

Refinement (model) checkers

- Analysis and verification of systems via state exploration
- FDR: most popular refinement checker for CSP
- State explosion problem

Verifying properties by proof development

Objectives

Main: *"provide an initial formalisation of the CSP language in Coq."*

Specific objectives

- Define the syntax of a subset of CSP in Coq
- Support for the LTS representation based on the SOS
- Verify traces refinement via property-based random testing

Main contributions

- Abstract and concrete syntax for a subset of CSP operators
- Contextual rules for CSP specifications
- Operational semantics via the SOS approach
- Inductive and functional definitions of LTSs and traces
- LTS visualisation using GraphViz
- Automation for checking contextual rules and is-a-trace relation
- Traces refinement verification using Quickchick

Centro de Informática UFPE
Universidade Federal de Pernambuco

Agenda

1. Background
   - CSP
   - Coq
   - QuickChick
2. A theory for CSP in Coq
   - Abstract and concrete syntax
   - Structured Operational Semantics
   - Labelled Transition Systems
   - Traces refinement
3. Conclusion
   - Related work
   - Future work

Agenda

1 Background
- CSP
- Coq
- QuickChick

2 A theory for CSP in Coq
- Abstract and concrete syntax
- Structured Operational Semantics
- Labelled Transition Systems
- Traces refinement

3 Conclusion
- Related work
- Future work

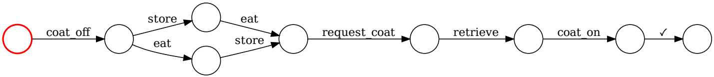
CSP: Communicating Sequential Processes

Example: the cloakroom attendant

- CSP_M

```
channel coat_on, coat_off, store, retrieve, request_coat, eat
SYSTEM =
   coat_off -> store -> request_coat -> retrieve -> coat_on -> SKIP
   [| {coat_off, request_coat, coat_on} |]
   coat_off -> eat -> request_coat -> coat_on -> SKIP
```

- LTS



- Traces
  - $\langle \mathit{coat\_off}, \mathit{store}, \mathit{eat} \rangle$, $\langle \mathit{coat\_off}, \mathit{eat}, \mathit{store}, \mathit{request\_coat} \rangle$

    ...

Centro de Informática UFPE

Universidade Federal de Pernambuco

Coq: a proof assistant

Functional and inductive definitions

```
Fixpoint evenb (n:nat) : bool :=
  match n with
    | O ⇒ true
    | S O ⇒ false
    | S (S n') ⇒ evenb n'
  end.
```

```
Inductive ev : nat → Prop :=
  | ev_0 : ev 0
  | ev_SS (n : nat) (H : ev n) : ev (S (S n)).
```

Proof development and the tactics language Ltac

```
Lemma negb_involutive : ∀ (b : bool),
  negb (negb b) = b.
Proof.
  destruct b.
  - simpl. reflexivity.
  - simpl. reflexivity.
Qed.
```

```
Ltac solve_negb_inv b :=
  destruct b; simpl; reflexivity.
```

QuickChick: a property-based testing tool

Example

```
Fixpoint remove (x : nat) (l : list nat) : list nat :=
  match l with
    | [] ⇒ []
    | h::t ⇒ if h =? x then t else h :: remove x t
  end.

Conjecture removeP : ∀ x l, ¬ (In x (remove x l)).

QuickChick removeP.
```

Output

```
0
[0, 0]
Failed!  After 17 tests and 12 shrinks
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Agenda

1 Background
- CSP
- Coq
- QuickChick

2 A theory for CSP in Coq
- Abstract and concrete syntax
- Structured Operational Semantics
- Labelled Transition Systems
- Traces refinement

3 Conclusion
- Related work
- Future work

Centro de Informática UFPE

Universidade Federal de Pernambuco

Abstract and concrete syntax

Event prefix

Abstract

```
ProcPrefix (Event "e") STOP
```

Concrete

```
"e" --> STOP
```

Example: process "PRINTER"

- CSP_{Coq}

Abstract

```
Proc "PRINTER"
 (ProcPrefix (Event "accept")
  (ProcPrefix (Event "print")
   STOP))
```

Concrete

```
"PRINTER" ::= "accept" -->
     "print" --> STOP
```

- CSP_M

```
PRINTER = accept ->
       print -> STOP
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Abstract and concrete syntax

Constructor	CSP_M	CSP_{Coq}
Stop	STOP	STOP
Skip	SKIP	SKIP
Event prefix	e -> P	e --> P
External choice	P [] Q	P [] Q
Internal choice	P |~| Q	P |~| Q
Alphabetised parallel	P [A || B] Q	P [[A \\ B]] Q
Generalised parallel	P [| A |] Q	P [| A |] Q
Interleave	P ||| Q	P ||| Q
Sequential composition	P ; Q	P ;; Q
Event hiding	P \ A	P \ A
Process definition	P = Q	P ::= Q
Process name	P	ProcRef “P”

Abstract and concrete syntax

```
Record specification : Type := Build_Spec
{
  ch_list : list channel;
  proc_list : list proc_def;
  non_empty_proc_ids : ¬ In EmptyString (map get_proc_id proc_list);
  non_empty_events : ¬ In EmptyString (concat_channels ch_list);
  no_dup_events_proc_ids : NoDup ((concat_channels ch_list) ++ (map get_proc_id proc_list));
  no_missing_proc_defs : incl (get_proc_refs proc_list) (map get_proc_id proc_list);
  no_missing_events : incl (get_events proc_list) (concat_channels ch_list)
}.

Ltac solve_spec_ctx_rules spec_cons := apply spec_cons;
  repeat (
    match goal with
    | ⊢ ¬ In _ _ ⇒ solve_not_in
    | ⊢ NoDup _ ⇒ solve_nodup
    | ⊢ incl _ _ ⇒ solve_incl
    end
  ); fail "One or more contextual rules were not fulfilled".
```

Structured Operational Semantics

Inference rule

Event prefix

$$\frac{}{(a \rightarrow P) \xrightarrow{a} P}$$

External choice

$$\frac{P \xrightarrow{a} P'}{P \,\square\, Q \xrightarrow{a} P'} \quad (a \neq \tau)$$

Inductive definition: *sosR*

```
Inductive sosR : specification →
  proc_body → event_tau_tick → proc_body → Prop :=
  | prefix_rule (S : specification) (P : proc_body) (a : event) :
    S # (a --> P) // Event a ==> P
  | ext_choice_left_rule (S : specification) (P Q : proc_body) :
    ∀ (P' : proc_body) (a : event_tau_tick),
      ¬ eq a Tau →
      (S # P // a ==> P') →
      (S # P [] Q // a ==> P')
```

Labelled Transition Systems

Propositional function *ltsR*

```
Definition ltsR (S : specification) (name : string) (T : set
transition) : Prop :=
  match get_proc_body S name with
  | Some body ⇒ NoDup T ∧ ltsR' S T [body] nil
  | None ⇒ False
  end.
```

Labelled Transition Systems

Inductive definition: *ltsR'*

`Inductive` *ltsR'* : *specification* $\rightarrow$ `set` *transition* $\rightarrow$ `set` *proc_body* $\rightarrow$ `set` *proc_body* $\rightarrow$ `Prop`.

- *lts_empty_rule*: no states remain to be visited; the corresponding LTS is empty
- *lts_inductive_rule*: for all process states, it is valid operation according to the SOS, if, and only if, the corresponding 3-tuple belongs to the set of transitions

Centro de Informática UFPE

Universidade Federal de Pernambuco

Labelled Transition Systems

Functional definitions: *compute_ltsR* and *generate_dot*

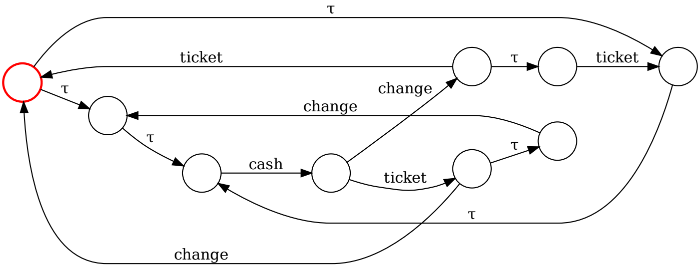
```
Definition compute_ltsR
  (S : specification) (name : string) (limit : nat) : option (set transition).
Definition generate_dot (lts : option (set transition)) : string.
```

Example: process “MACHINE”

```
Definition PARKING_PERMIT_MCH : specification.
Proof.
  solve_spec_ctx_rules (
    Build_Spec
    [ Channel {{“cash”, “ticket”, “change”}} ]
    [ “TICKET” ::= “cash” --> “ticket” --> ProcRef “TICKET”
    ; “CHANGE” ::= “cash” --> “change” --> ProcRef “CHANGE”
    ; “MACHINE” ::= ProcRef “TICKET”
                        [[ {{“cash”, “ticket”}} \\ {{“cash”, “change”}} ]]
                        ProcRef “CHANGE” ]
  ).
Defined.
```

Labelled Transition Systems

Graph visualisation (GraphViz)



Is-a-trace relation

Inductive definition: *traceR'*

```
Inductive traceR' : specification → proc_body → trace → Prop.
```

- *empty_trace_rule*
- *event_trace_rule*
- *tau_trace_rule*

Proof automation for the is-a-trace relation: *solve_trace*

```
Example MACHINE_TRACE :
  traceR PARKING_PERMIT_MCH "MACHINE" ["cash" ; "ticket" ; "change"].
Proof. solve_trace. Qed.
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Traces refinement

Traces refinement formalisation

```
Definition trace_refinement
  (S : specification) (Spec Imp : string) : Prop :=
   ∀ (t : trace), traceR S Imp t → traceR S Spec t.

Notation "S '#' P '[T=' Q" := (trace_refinement S P Q)
   (at level 150, left associativity).
```

Traces refinement

Traces generator: *gen_valid_trace'*

```
Fixpoint gen_valid_trace'
  (S : specification) (P : proc_body) (size : nat)
  : G (option semantics_trace.trace) :=
  match size with
  | O ⇒ ret nil
  | S size' ⇒
    freq_ (ret nil) [
      (1, ret nil) ;
      (size,
        bind (gen_valid_trans S P) (
          fun t ⇒ (
            match t with
            | nil ⇒ ret nil
            | (Event e, Q) :: _ ⇒
              bind (gen_valid_trace' S Q size') (
                fun ts ⇒ ret (Event e :: ts)
              )
            | (Tick, Q) :: _ ⇒
              bind (gen_valid_trace' S Q size') (
                fun ts ⇒ ret (Tick :: ts)
              )
            | (Tau, Q) :: _ ⇒
              bind (gen_valid_trace' S Q size') (
                fun ts ⇒ ret ts
              )
            end
          ) ) ) ] end.
```

Traces refinement

Demonstrating the random generator of valid traces

Sample (*gen_valid_trace PARKING_PERMIT_MCH* "MACHINE" 10).

Output:

```
[Some []; Some ["cash"; "change"; "ticket"; "cash";
"change"]; Some ["cash"]; Some ["cash"; "ticket";
"change"; "cash"]; Some []; Some ["cash"; "change";
"ticket"; "cash"]; Some ["cash"; "change"; "ticket"];
...]
```

Traces refinement

Executable property: *traceP*

```
Definition traceP
  (S : specification)
  (proc_id : string)
  (fuel : nat)
  (t : option semantics_trace.trace) : bool.
```

Refinement checker: *trace_refinement_checker*

```
Definition trace_refinement_checker
  (S : specification)
  (Imp Spec : string)
  (trace_max_size : nat)
  (fuel : nat) : Checker :=
    forAll (gen_valid_trace S Imp trace_max_size)
      (traceP S Spec fuel).
```

Traces refinement

Searching for counterexamples with QuickChick

```
Definition EXAMPLE : specification.
Proof.
  solve_spec_ctx_rules (
    Build_Spec
      [ Channel {{"a", "b", "c"}} ]
      [ "P" ::= "a" --> "b" --> ProcRef "P" ;
        "Q" ::= ("a" --> "b" --> ProcRef "Q") [] ("c" --> STOP) ]
  ).
Defined.
QuickChick (trace_refinement_checker EXAMPLE "Q" "P" 5 1000).
```

Output:

```
Some ["c"]
*** Failed after 3 tests
and 0 shrinks.  (0 discards)
```

Agenda

1 Background
- CSP
- Coq
- QuickChick

2 A theory for CSP in Coq
- Abstract and concrete syntax
- Structured Operational Semantics
- Labelled Transition Systems
- Traces refinement

3 Conclusion
- Related work
- Future work

Centro de Informática UFPE

Universidade Federal de Pernambuco

Conclusion

CSP_{Coq}: an initial formalisation of the CSP language in Coq

- Inductive and functional definitions of LTSs and traces
- Third-party visualisation support for LTS representation
- Automation for checking contextual rules and is-a-trace relation
- Random testing for refinement relations using QuickChick

Related work

CSP-Prover

- Interactive theorem prover based on Isabelle
- Stable-failures model as the underlying denotational semantics
- Semi-automated proof tactics for refinement verification

Isabelle/UTP

- Implementation of the *Unifying Theories of Programming*
- Support for construction of denotational semantic meta-models
- Useful to construct program verification tools

Distinguishable features of CSP_{Coq}

- Graphical representation of LTSs
- Property-based testing for checking traces refinement relations

Centro de Informática UFPE

Universidade Federal de Pernambuco

Future work

- Extend the CSP_{Coq} dialect to include other CSP operators
- Check for invalid recursions (hiding and parallelism operations)
- Define a tactic to automate proofs involving the relation *ltsR*
- Prove correctness of definition *compute_ltsR*
- Prove correctness of generator *gen_valid_trace*
- Define traces refinement in terms of bi-simulation

A THEORY FOR COMMUNICATING SEQUENTIAL PROCESSES IN COQ

Carlos Alberto da Silva Carvalho de Freitas
(casc2@cin.ufpe.br)

Supervisor: Gustavo Carvalho

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil