Programming with Monadic CSP-Style Processes in Dependent Type Theory

Bashar Igried and Anton Setzer

Swansea University, Swansea,Wales, UK

bashar.igried@yahoo.com , *a.g.setzer@swansea.ac.uk*

JAIST, Japan, 6 Sep 2016

Overview

1. Agda
2. Why Agda?
3. Process Algebra
4. CSP
5. CSP-Agda
6. Choice Sets
7. Simulator
8. Future Work
9. Conclusion

Agda

- Agda is a theorem prover and dependently typed programming language, which extends intensional Martin-Löf type theory.
- The current version of this language is Agda 2 which has beendesigned and implemented by Ulf Norell in his PhD in 2007.
- Agda has a termination and coverage checker. This makes Agda a total language, so each Agda program terminates.
- The termination checker verifies that all programs terminate.
- Without the termination and coverage checker, Agda would be inconsistent.
- Agda has a type checker which refuses incorrect proofs by detecting unmatched types.
- The type checker in Agda shows the goals and the environment information related to proof.
- The coverage checker guarantees that the definition of a function covers all possible cases.

Agda

- The user interface of Agda is Emacs.
- This interface has been useful for interactively writing and verifying proofs.
- Programs can be developed incrementally, since we can leave parts of the program unfinished.

Agda

- There are several levels of types in Agda, the lowest is for historic reasons called Set.
- Types in Agda are given as:
  - dependent function types.
  - inductive types.
  - coinductive types.
  - record types(which are in the newer approach used for defining coinductive types).
  - generalisation of inductive-recursive definitions.

Agda

Inductive data types are given as sets A together with constructors which are strictly positive in A.
For instance the collection of finite sets is given as

```
data Fin : ℕ → Set where
  zero : {n : ℕ} → Fin (suc n)
  suc  : {n : ℕ} (i : Fin n) → Fin (suc n)
```

- Here $\{n : \mathbb{N}\}$ is an implicit argument.
- Implicit arguments are omitted, provided they can be uniquely determined by the type checker.
- We can make a hidden argument explicit by writing for instance zero $\{n\}$.

Agda

- The above definition introduces a new type Fin : $\mathbb{N} \to$ Set where (Fin n) is a type with n elements.
- The elements of (Fin n) are those constructed from applying these constructors.

Agda

Therefore we can define functions by case distinction on these constructors using pattern matching, e.g.

$$
\begin{aligned}
&\mathsf{to}\mathbb{N} : \forall \{n\} \rightarrow \mathsf{Fin}\ n \rightarrow \mathbb{N} \\
&\mathsf{to}\mathbb{N}\ \ \mathsf{zero} \qquad\ \ = \ 0 \\
&\mathsf{to}\mathbb{N}\ \ (\mathsf{suc}\ n) \ = \ \mathsf{suc}\ (\mathsf{to}\mathbb{N}\ n)
\end{aligned}
$$

Agda

There are two approaches of defining coinductive types in Agda.

- The older approach is based on the notion of codata types.
- The newer one is based on coalgebras given by their observations or eliminators

We will follow the newer one, pioneered by Setzer, Abel, Pientka and Thibodeau.

Why Agda?

Why Agda?

- Agda supports induction-recursion.

  Induction-Recursion allows to define universes.
- Agda supports definition of coalgebras by elimination rules and defining their elements by combined pattern and copattern matching.
- Using of copattern matching allows to define code which looks close to normal mathematical proofs.

Overview Of Process Algebras

Overview Of Process Algebras

- "Process algebra" was initiated in 1982 by Bergstra and Klop [1], in order to provide a formal semantics to concurrent systems.
- Baeten et. al. Process algebra is the study of distributed or parallel systems by algebraic means.
- Three main process algebras theories were developed.
  - Calculus of Communicating Systems (CCS).
    Developed by Robin Milner in 1980.
  - Communicating Sequential Processes (CSP).
    Developed by Tony Hoare in 1978.
  - Algebra of Communicating Processes (ACP).
    Developed by Jan Bergstra and Jan Willem Klop, in 1982.
- Processes will be defined in Agda according to the operational behaviour of the corresponding CSP processes.

Example Of Processes

CSP

- CSP considered as a formal specification language, developed in order to describe concurrent systems.

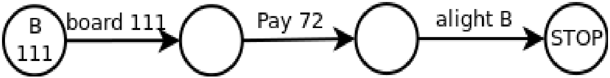
  By identifying their behaviour through their communications.
- CSP is a notation for studying processes which interact with each other and their environment.
- In CSP we can describe a process by the way it can communicate with its environment.
- A system contains one or more processes, which interact with each other through their interfaces.

CSP Syntax

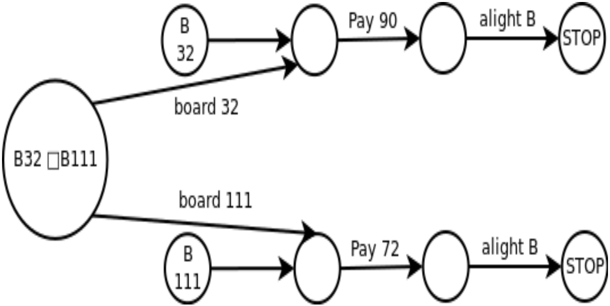
In the following table, we list the syntax of CSP processes:

Q ::= STOP	$STOP$
| SKIP	$SKIP$
| prefix	$a \rightarrow Q$
| external choice	$Q \Box Q$
| internal choice	$Q \sqcap Q$
| hiding	$Q \setminus a$
| renaming	$Q[R]$
| parallel	$Q \;_X\|_Y\; Q$
| interleaving	$Q \; \|\|\| \; Q$
| interrupt	$Q \triangle Q$
| composition	$Q \, ; Q$

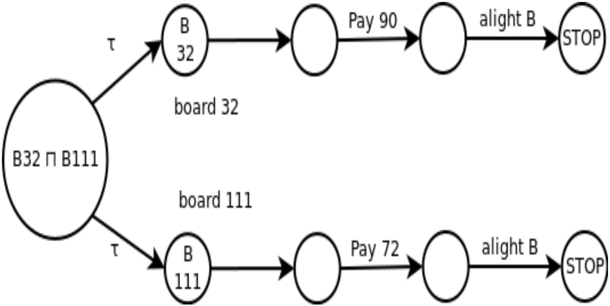
Example Of Processes



Example Of Processes



Example Of Processes



CSP-Agda

CSP-Agda

- We will represent the process algebra CSP in a coinductive form in dependent type theory.
- Implement it in Agda.
- can proceed at any time with labelled transitions (external choices), silent transitions (internal choices), or $\checkmark$-events (termination).
- Therefore, processes in CSP-Agda have as well this possibility.

CSP-Agda

- In process algebras, if a process terminates, it does not return any information except for that it terminated.
- We want to define processes in a monadic way in order to combine them in a modular way.
- Therefore, if processes terminate, they should return some additional information, namely the result returned by the process.

CSP-Agda

In Agda the corresponding code is as follows:

```
mutual
  record Process∞ (i : Size) (c : Choice) : Set where
    coinductive
    field
      forcep : {j : Size< i} → Process j c
      Str∞   : String

  data Process (i : Size)  (c : Choice) : Set where
    terminate  : ChoiceSet  c    → Process i c
    node       : Process+   i c  → Process i c
```

CSP-Agda

In Agda the corresponding code is as follows:

```
record Process+ (i : Size) (c : Choice) : Set where
  constructor process+
  coinductive
  field
    E    : Choice
    Lab  : ChoiceSet E → Label
    PE   : ChoiceSet E → Process∞ i c
    I    : Choice
    PI   : ChoiceSet I → Process∞ i c
    T    : Choice
    PT   : ChoiceSet T → ChoiceSet c
    Str+ : String
```

So we have in case of a process progressing:

(1) an index set E of external choices and for each external choice e the Label (Lab e) and the next process (PE e);
(2) an index set of internal choices I and for each internal choice i the next process (PI i); and
(3) an index set of termination choices T corresponding to ✓-events and for each termination choice t the return value PT t : A.

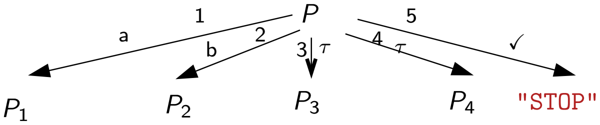
CSP-Agda

As an example the following Agda code describes the process pictured below:

$$
\begin{aligned}
P = \quad & \mathsf{node}\ (\mathsf{process+}\ E\ Lab\ PE\ I\ PI\ T\ PT\ \texttt{"P"}) \\
& : \mathsf{Process\ String} \quad \text{where}
\end{aligned}
$$

$$
\begin{aligned}
E = \quad & \text{code for } \{1, 2\} & \qquad I = \quad & \text{code for } \{3, 4\} \\
T = \quad & \text{code for } \{5\}
\end{aligned}
$$

$$
\begin{array}{llllllll}
Lab\,1 & = & a & \quad Lab\,2 & = & b & \quad PE\,1 & = & P_1 \\
PE\,2 & = & P_2 & \quad PI\,3 & = & P_3 & \quad PI\,4 & = & P_4 \\
PT\,5 & = & \texttt{"STOP"}
\end{array}
$$



Choices Set

Choices Set

- Choice sets are modelled by a universe.
- Universes go back to Martin-Löf in order to formulate the notion of a type consisting of types.
- Universes are defined in Agda by an inductive-recursive definition.

Choice Sets

We give here the code expressing that Choice is closed under fin, ⊎ and subset'.

```
mutual
data Choice : Set where
   fin     : ℕ → Choice
   _⊎'_    : Choice → Choice → Choice
   subset' : (E : Choice) → (ChoiceSet E → Bool)
     → Choice

ChoiceSet : Choice → Set
ChoiceSet (fin n)       =   Fin n
ChoiceSet (s ⊎' t)      =   ChoiceSet s ⊎ ChoiceSet t
ChoiceSet (subset' E f) = subset (ChoiceSet E) f
```

Interleaving operator

Interleaving operator

- In this process, the components P and Q execute completely independently of each other.
- Each event is performed by exactly one process.
- The operational semantics rules are straightforward:

$$\frac{P \xrightarrow{\checkmark} \bar{P} \qquad Q \xrightarrow{\checkmark} \bar{Q}}{P \mid\mid\mid Q \xrightarrow{\checkmark} \bar{P} \mid\mid\mid \bar{Q}}$$

$$\frac{P \xrightarrow{\mu} \bar{P}}{\begin{array}{c} P \mid\mid\mid Q \xrightarrow{\mu} \bar{P} \mid\mid\mid Q \\ Q \mid\mid\mid P \xrightarrow{\mu} Q \mid\mid\mid \bar{P} \end{array}} \; \mu \neq \checkmark$$

Interleaving operator

We represent interleaving operator in CSP-Agda as follows

$$
\begin{array}{l}
\_|||{+\!\!+}\_ : \{i : \mathsf{Size}\} \to \{c_0\ c_1 : \mathsf{Choice}\} \\
\quad \to \mathsf{Process{+}}\ i\ c_0 \to \mathsf{Process{+}}\ i\ c_1 \\
\quad \to \mathsf{Process{+}}\ i\ (c_0 \times' c_1)
\end{array}
$$

$$
\begin{array}{llcl}
\mathsf{E} & (P \;|||{+\!\!+}\; Q) & = & \mathsf{E}\ P \uplus' \mathsf{E}\ Q \\
\mathsf{Lab} & (P \;|||{+\!\!+}\; Q)\ (\mathsf{inj}_1\ c) & = & \mathsf{Lab}\ P\ c \\
\mathsf{Lab} & (P \;|||{+\!\!+}\; Q)\ (\mathsf{inj}_2\ c) & = & \mathsf{Lab}\ Q\ c \\
\mathsf{PE} & (P \;|||{+\!\!+}\; Q)\ (\mathsf{inj}_1\ c) & = & \mathsf{PE}\ P\ c \;|||{\infty}{+}\; Q \\
\mathsf{PE} & (P \;|||{+\!\!+}\; Q)\ (\mathsf{inj}_2\ c) & = & P \;|||{+}{\infty}\; \mathsf{PE}\ Q\ c \\
\mathsf{I} & (P \;|||{+\!\!+}\; Q) & = & \mathsf{I}\ P \uplus' \mathsf{I}\ Q \\
\mathsf{PI} & (P \;|||{+\!\!+}\; Q)\ (\mathsf{inj}_1\ c) & = & \mathsf{PI}\ P\ c \;|||{\infty}{+}\; Q \\
\mathsf{PI} & (P \;|||{+\!\!+}\; Q)\ (\mathsf{inj}_2\ c) & = & P \;|||{+}{\infty}\; \mathsf{PI}\ Q\ c \\
\mathsf{T} & (P \;|||{+\!\!+}\; Q) & = & \mathsf{T}\ P \times' \mathsf{T}\ Q \\
\mathsf{PT} & (P \;|||{+\!\!+}\; Q)\ (c\ ,,\ c_1) & = & \mathsf{PT}\ P\ c\ ,,\ \mathsf{PT}\ Q\ c_1 \\
\mathsf{Str{+}} & (P \;|||{+\!\!+}\; Q) & = & \mathsf{Str{+}}\ P \;|||\mathsf{Str}\; \mathsf{Str{+}}\ Q
\end{array}
$$

Interleaving operator

- When processes P and Q haven't terminated, then $P \mathbin{|||} Q$ will not terminate.
  - The external choices are the external choices of P and Q.
  - The labels are the labels from the processes P and Q, and we continue recursively with the interleaving combination.
  - The internal choices are defined similarly.

Interleaving operator

- A termination event can happen only if both processes have a termination event.
- If both processes terminate with results a and b, then the interleaving combination terminates with result $(a ,, b)$.
- If one process terminates but the other not, the rules of CSP express that one continues as the other other process, until it has terminated.
  - We can therefore equate, if P has terminated, $P \mid\mid\mid Q$ with Q.
  - However, we record the result obtained by P, and therefore apply fmap to Q in order to add the result of P to the result of Q when it terminates.

A Simulator of CSP-Agda

A Simulator of CSP-Agda

We have written a simulator in Agda.

- It turned out to be more complicated than expected, since we needed to convert processes, which are infinite entities, into strings, which are finitary.
- The solution was to add string components to Process

A Simulator of CSP-Agda

The simulator does the following:

- It will display to the user
  - The selected process,
  - The set of termination choices with their return value
  - And allows the user to choose an external or internal choice as a string input.
- If the input is correct, then the program continues with the process which is obtained by following that transition,
- otherwise an error message is returned and the program asks again for a choice.
- $\checkmark$-events are only displayed but one cannot follow them, because afterwards the system would stop.

A Simulator of CSP-Agda

An example run of the simulator is as follows:

```
((b ⟶  (a ⟶  STOP)) □  (((c ⟶  STOP) ⊓ (a ⟶  STOP)) □  SKIP(STOP)))
Termination-Events: (inr (inr 0)):(inr (inr STOP))
Events: e-(inl 0):b i-(inr (inl 0)):τ i-(inr (inl 1)):τ
Choose Event
i-(inr (inl 0))
((b ⟶  (a ⟶  STOP)) □  ((c ⟶  STOP) □  SKIP(STOP)))
Termination-Events: (inr (inr 0)):(inr (inr STOP))
Events: e-(inl 0):b e-(inr (inl 0)):c
Choose Event
e-(inl 0)
(fmap inl (a ⟶  STOP))
Termination-Events:
Events: e-0:a
Choose Event
```

Future Work

- Looking to the future, we would like to model complex systems in Agda.
- Model examples of processes occurring in the European Train Management System (ERTMS) in Agda.
- Show correctness.

Conclusion

- A formalisation of CSP in Agda has been developed using coalgebra types and copattern matching.
- The other operations (external choice, internal choice, parallel operations, hiding, renaming, etc.) are defined in a similar way.
- A simulator of CSP processes in Agda has been developed.

Conclusion

- Define approach using Sized types.
- For complex examples (e.g recursion) sized types are used to allow application of functions to the co-IH.

[1] J. A. Bergstra and J. W. Klop. Fixed point semantics in process algebras. CWI technical report, Stichting Mathematisch Centrum. Informatica-IW 206/82, 1982.

The End