A High-Level Language for Modeling Algorithms and their Properties

Sabina Akhtar Stephan Merz Martin Quinson

LORIA – INRIA Nancy Grand Est and Nancy University, Nancy, France

SBMF 2010

Outline

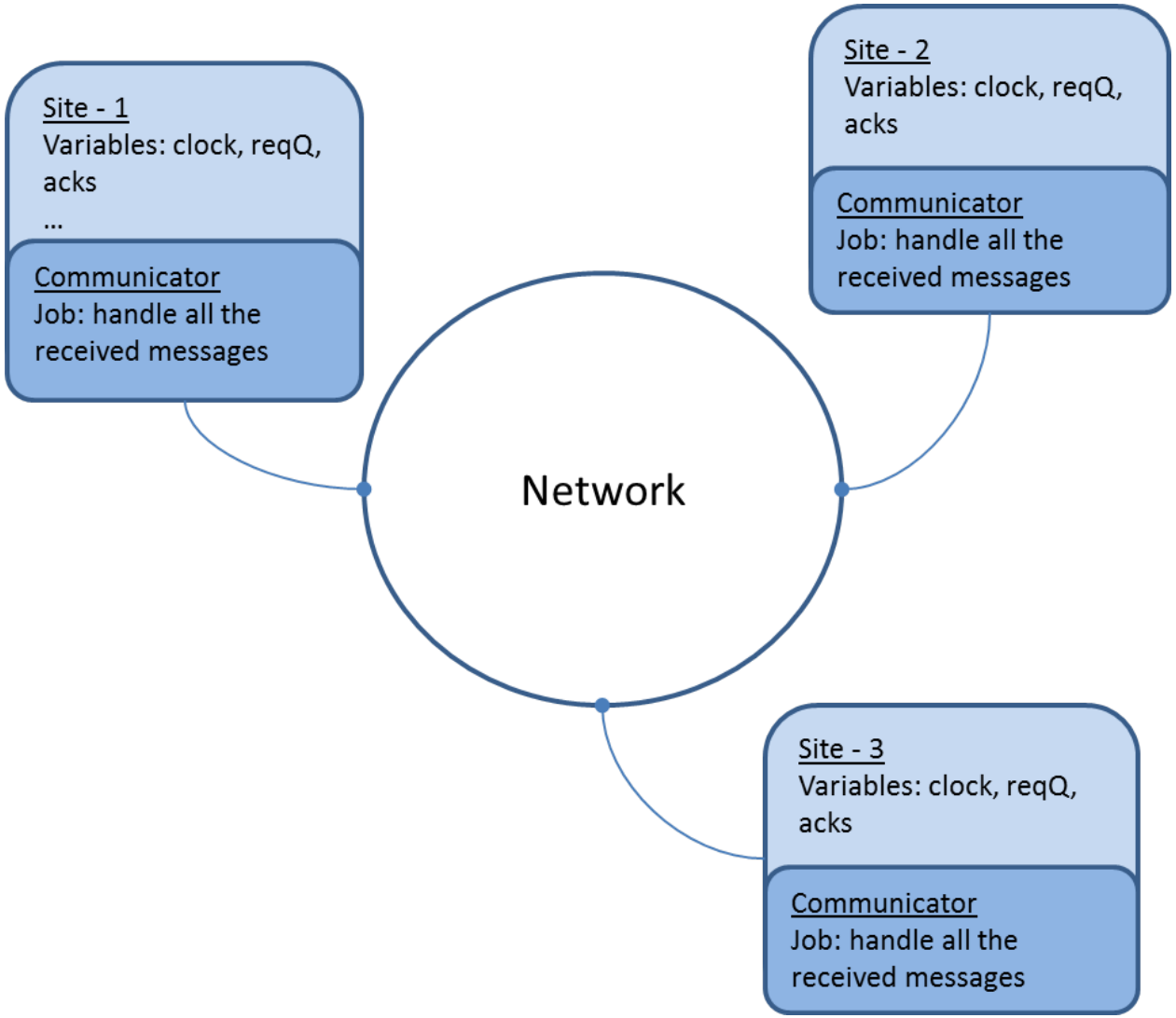
Background

- Formal verification of concurrent and distributed systems
- Problems like deadlocks, race conditions,...
- TLA$^+$: Specification language
  - developed by Leslie Lamport
  - a language based on mathematical set theory
- TLC: Model checker
  - for verifying TLA$^+$ specifications

Leslie Lamport. Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, 2002.

An Example

- Lamport's Mutual Exclusion Algorithm



TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"}\quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \ \land\ \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"}\quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \land \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"}\quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \land \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"}\quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \land \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"}\quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \land \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
Init & \triangleq & \land\ clock = 0 \\
 & & \land\ \ldots \\
 & & \land\ ProcSet = SiteIDs \cup CommunicatorIDs \\
 & & \land\ pc = [self \in ProcSet \mapsto \text{CASE } self \in SiteIDs \to \text{"}ncrit\text{"} \\
 & & \qquad\qquad \square self \in CommunicatorIDs \to \text{"}chkMsg\text{"}\,] \\
ncrit(self) & \triangleq & \land\ pc[self] = \text{"}ncrit\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ pc' = [pc \text{ EXCEPT!}[self] = \text{"}try\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } vars \backslash \{pc\} \\
try(self) & \triangleq & \land\ pc[self] = \text{"}try\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
Site(self) & \triangleq & ncrit(self) \lor try(self) \lor enter(self) \lor crit(self) \lor exit(self) \\
\ldots & & \\
Communicator(self) & \triangleq & chkMsg(self) \\
Next & \triangleq & \lor\ \exists self \in SiteIDs : Site(self) \\
 & & \lor\ \exists self \in CommunicatorIDs : Communicator(self) \\
 & & \lor\ (\land\ \forall self \in ProcSet : pc[self] = \text{"}Done\text{"}\ \ ) \\
 & & \qquad \land\ \text{UNCHANGED } vars \\
 & & \\
Spec & \triangleq & Init\ \land\ \square[Next]_{vars}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"}\quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \land \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"}\quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \land \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

TLA+ Specifications

$$
\begin{array}{lcl}
\mathit{Init} & \triangleq & \land\ \mathit{clock} = 0 \\
 & & \land\ \ldots \\
 & & \land\ \mathit{ProcSet} = \mathit{SiteIDs} \cup \mathit{CommunicatorIDs} \\
 & & \land\ \mathit{pc} = [\mathit{self} \in \mathit{ProcSet} \mapsto \text{CASE } \mathit{self} \in \mathit{SiteIDs} \to \text{"}\mathit{ncrit}\text{"} \\
 & & \qquad\qquad \square \mathit{self} \in \mathit{CommunicatorIDs} \to \text{"}\mathit{chkMsg}\text{"}\,] \\
\mathit{ncrit}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{ncrit}\text{"} \\
 & & \land\ \ldots \\
 & & \qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT!}[\mathit{self}] = \text{"}\mathit{try}\text{"}\,] \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \backslash \{\mathit{pc}\} \\
\mathit{try}(\mathit{self}) & \triangleq & \land\ \mathit{pc}[\mathit{self}] = \text{"}\mathit{try}\text{"} \\
 & & \land\ \ldots \\
\ldots & & \\
\mathit{Site}(\mathit{self}) & \triangleq & \mathit{ncrit}(\mathit{self}) \lor \mathit{try}(\mathit{self}) \lor \mathit{enter}(\mathit{self}) \lor \mathit{crit}(\mathit{self}) \lor \mathit{exit}(\mathit{self}) \\
\ldots & & \\
\mathit{Communicator}(\mathit{self}) & \triangleq & \mathit{chkMsg}(\mathit{self}) \\
\mathit{Next} & \triangleq & \lor\ \exists \mathit{self} \in \mathit{SiteIDs} : \mathit{Site}(\mathit{self}) \\
 & & \lor\ \exists \mathit{self} \in \mathit{CommunicatorIDs} : \mathit{Communicator}(\mathit{self}) \\
 & & \lor\ (\land\ \forall \mathit{self} \in \mathit{ProcSet} : \mathit{pc}[\mathit{self}] = \text{"}\mathit{Done}\text{"} \quad) \\
 & & \qquad \land\ \text{UNCHANGED } \mathit{vars} \\
 & & \\
\mathit{Spec} & \triangleq & \mathit{Init} \ \land\ \square[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

PlusCal: A high-level language

- TLA$^+$: Specification language
  - requires specifications in the form of formulas
  - difficult to write for algorithm designers
- PlusCal: Algorithmic Language
  - proposed by Leslie Lamport for algorithm designers
  - a language for modeling algorithms
  - generates TLA$^+$ specifications for a given model
- Features
  - allows writing informal description of algorithms
  - no complicated concepts
  - constructs for expressing non-determinism

Leslie Lamport. The +CAL Algorithm Language.
Theoretical Aspects of Computing-ICTAC 2009, number 5684, pp. 36-60.

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .        (* Variables, define sections and macros *)
process Site ∈ Peers                                  (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
---------------------------MODULE LamportMutex---------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .        (* Variables, define sections and macros *)
process Site ∈ Peers                                 (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
=============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .          (* Variables, define sections and macros *)
process Site ∈ Peers                                    (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .        (* Variables, define sections and macros *)
process Site ∈ Peers                                  (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .        (* Variables, define sections and macros *)
process Site ∈ Peers                                    (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .          (* Variables, define sections and macros *)
process Site ∈ Peers                                    (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .          (* Variables, define sections and macros *)
process Site ∈ Peers                                    (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .          (* Variables, define sections and macros *)
process Site ∈ Peers                                    (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .          (* Variables, define sections and macros *)
process Site ∈ Peers                                    (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Lamport's Mutual Exclusion algorithm in PlusCal

```
------------------------------MODULE LamportMutex------------------------------
EXTENDS Naturals, Sequences                          (* Modules to be imported *)
CONSTANTS N, maxClock, Peers, Workers

(* - - algorithm LamportMutex
variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]
macro send(from, to, msg) begin . . .        (* Variables, define sections and macros *)
process Site ∈ Peers                                  (* Processes *)
   variables clock = 1, . . .
   begin
        start: skip;
        . . .
end process
process Communicator ∈ Workers
begin
   . . .
end process
end algorithm
*)
\* BEGIN TRANSLATION
                                   (* Compiler generates TLA+ formulas here. *)
\* END TRANSLATION
==============================================================================
```

Outline

Why change PlusCal?

- Need to understand TLA$^+$ and the compilation
  - cannot express properties in PlusCal algorithms
  - fairness assumptions should be added in generated TLA$^+$ specifications
- Lack of process hierarchy and scoping rules
  - impossible to express distributed algorithms naturally
  - all variables are considered as global variables
- Restrictions in specifying atomicity
  - labels define atomic blocks
  - restrictions on label placements
- Other technical limitations
  - no primitive for iterating over a set
  - restriction on multiple assignments to a variable in a block

Outline

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Lamport's Mutual Exclusion algorithm in PlusCal-2

```
algorithm LamportMutex
extends Naturals, Sequences                          (* Modules to be imported *)
constants N, maxClock

variable network = [from ∈ Site ↦ [to ∈ Site ↦ ⟨⟩]]   (* Variables and definitions *)
definition send(from, to, msg) ≜ . . .
process Site[N]                                      (* Processes *)
        variables clock = 1, . . .
        fair process Communicator[1]                 (* subprocess Communicator *)
        . . .
        end process
        . . .
end process

end algorithm
temporal ∀ s ∈ Site : Site[s]@enter ⇝ Site[s]@critsection
. . .
(* Finite instance for model checking *)
constants N = 3, maxclock = 5
constraint ∀ s ∈ Site : Site[s].clock ≤ maxClock
```

Outline

The PlusCal-2 Statements

- Assignment and skip statements
- Atomic construct

  atomic
  label1: x := 3;
  label2: y := 4;
  end atomic

- Non-deterministic choice construct: **either or**
- Conditional constructs
  - **if**, **when**, and **either** from previous PlusCal.
  - new construct **branch**, inspired by Dijkstra's guarded commands
- Iteration constructs
  - **while**, **loop** and **for** constructs

Outline

The PlusCal-2 Compiler

- PlusCal-2 Parser
- Translation to intermediate format

PlusCal-2 algorithm

$\underline{\lambda}$: **while** x $>$ 4 **do**
    x := x + 1;
$\underline{\mu}$: ...
end while
$\underline{\nu}$: ...

Intermediate format

$\underline{\lambda}$: **branch**
    x $>$ 4 **then**
        x := x + 1; pc[self] := μ;
    $\neg$(x $>$ 4) **then**
        pc[self] := ν;
end branch
$\underline{\mu}$: ...
pc[self] := λ;

The PlusCal-2 Compiler

- PlusCal-2 Parser
- Translation to intermediate format

PlusCal-2 algorithm

λ: **while** x $>$ 4 **do**
    x := x + 1;
μ: ...
end while
ν: ...

Intermediate format

λ: **branch**
    x $>$ 4 **then**
        x := x + 1; pc[self] := μ;
    $\neg$(x $>$ 4) **then**
        pc[self] := ν;
end branch
μ: ...
pc[self] := λ;

The PlusCal-2 Compiler

- PlusCal-2 Parser
- Translation to intermediate format

PlusCal-2 algorithm

λ: **while** x $>$ 4 **do**
    x := x + 1;
μ: ...
end while
ν: ...

Intermediate format

λ: **branch**
    x $>$ 4 **then**
        x := x + 1; pc[self] := μ;
    $\neg$(x $>$ 4) **then**
        pc[self] := ν;
end branch
μ: ...
pc[self] := λ;

The PlusCal-2 Compiler

- PlusCal-2 Parser
- Translation to intermediate format

PlusCal-2 algorithm

$\underline{\lambda}$: **while** x $>$ 4 **do**
    x := x + 1;
$\underline{\mu}$: ...
end while
$\underline{\nu}$: ...

Intermediate format

$\underline{\lambda}$: **branch**
    x $>$ 4 **then**
        x := x + 1; pc[self] := μ;
    $\neg$(x $>$ 4) **then**
        pc[self] := ν;
end branch
$\underline{\mu}$: ...
pc[self] := λ;

The PlusCal-2 Compiler

- PlusCal-2 Parser
- Translation to intermediate format

PlusCal-2 algorithm

$\underline{\lambda}$: **while** x $>$ 4 **do**
    x := x + 1;
$\underline{\mu}$: ...
end while
$\underline{\nu}$: ...

Intermediate format

$\underline{\lambda}$: **branch**
    x $>$ 4 **then**
        x := x + 1; pc[self] := μ;
    $\neg$(x $>$ 4) **then**
        pc[self] := ν;
end branch
$\underline{\mu}$: ...
pc[self] := λ;

The PlusCal-2 Compiler

- PlusCal-2 Parser
- Translation to intermediate format

PlusCal-2 algorithm

$\underline{\lambda}$: **while** x $>$ 4 **do**
    x := x + 1;
$\underline{\mu}$: ...
end while
$\underline{\nu}$: ...

Intermediate format

$\underline{\lambda}$: **branch**
    x $>$ 4 **then**
        x := x + 1; pc[self] := μ;
    $\neg$(x $>$ 4) **then**
        pc[self] := ν;
end branch
$\underline{\mu}$: ...
pc[self] := λ;

The PlusCal Compiler

- Generation of TLA$^+$ code
  - generates the actual TLA$^+$ model from the list of guarded commands

Intermediate format

$\underline{\lambda}$: **branch**
  x $>$ 4 **then**
    x := x + 1; pc[self] := μ;
  $\neg$(x $>$ 4) **then**
    pc[self] := ν;
end branch

TLA$^+$ code

$$\lambda(self) \triangleq \land pc[self] = \lambda \land \lor \land x > 4 \land x' = x + 1 \land pc' = [pc \text{ EXCEPT } ![self] = \mu] \land \text{UNCHANGED } vars \setminus \{x, pc\} \lor \land \neg(x > 4) \land pc' = [pc \text{ EXCEPT } ![self] = \nu] \land \text{UNCHANGED } vars \setminus \{pc\}$$

The PlusCal Compiler

- Generation of TLA$^+$ code
  - generates the actual TLA$^+$ model from the list of guarded commands

Intermediate format

λ: **branch**
  x $>$ 4 **then**
    x := x + 1; pc[self] := μ;
  $\neg$(x $>$ 4) **then**
    pc[self] := ν;
end branch

TLA$^+$ code

$$
\begin{aligned}
\lambda(self) \triangleq\ & \land\ pc[self] = \lambda \\
& \land\ \lor\ \land\ x > 4 \\
& \qquad\ \land\ x' = x + 1 \\
& \qquad\ \land\ pc' = [pc \text{ EXCEPT } ![self] = \mu] \\
& \qquad\ \land\ \text{UNCHANGED } vars \setminus \{x, pc\} \\
& \quad\ \lor\ \land\ \neg(x > 4) \\
& \qquad\ \land\ pc' = [pc \text{ EXCEPT } ![self] = \nu] \\
& \qquad\ \land\ \text{UNCHANGED } vars \setminus \{pc\}
\end{aligned}
$$

The PlusCal Compiler

- Generation of TLA$^+$ code
  - generates the actual TLA$^+$ model from the list of guarded commands

Intermediate format

λ: **branch**
  x $>$ 4 **then**
    x := x + 1; pc[self] := μ;
  $\neg$(x $>$ 4) **then**
    pc[self] := ν;
end branch

TLA$^+$ code

$$
\begin{aligned}
\lambda(self) \triangleq\ & \land\ pc[self] = \lambda \\
& \land \lor \land\ x > 4 \\
& \quad \land\ x' = x + 1 \\
& \quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \mu] \\
& \quad \land \text{ UNCHANGED } vars \setminus \{x, pc\} \\
& \quad \lor \land\ \neg(x > 4) \\
& \quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \nu] \\
& \quad \land \text{ UNCHANGED } vars \setminus \{pc\}
\end{aligned}
$$

The PlusCal Compiler

- Generation of TLA$^+$ code
  - generates the actual TLA$^+$ model from the list of guarded commands

Intermediate format

λ: **branch**
  x $>$ 4 **then**
    x := x + 1; pc[self] := μ;
  $\neg$(x $>$ 4) **then**
    pc[self] := ν;
end branch

TLA$^+$ code

$$
\begin{aligned}
\lambda(self) \triangleq\ & \land\ pc[self] = \lambda \\
& \land \lor \land\ x > 4 \\
& \quad \land\ x' = x + 1 \\
& \quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \mu] \\
& \quad \land \text{ UNCHANGED } vars \setminus \{x, pc\} \\
& \quad \lor \land\ \neg(x > 4) \\
& \quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \nu] \\
& \quad \land \text{ UNCHANGED } vars \setminus \{pc\}
\end{aligned}
$$

The PlusCal Compiler

- Generation of TLA$^+$ code
  - generates the actual TLA$^+$ model from the list of guarded commands

Intermediate format

λ: **branch**
  x $>$ 4 **then**
    x := x + 1; pc[self] := μ;
  $\neg$(x $>$ 4) **then**
    pc[self] := ν;
end branch

TLA$^+$ code

$$
\begin{aligned}
\lambda(self) \triangleq\ & \land\ pc[self] = \lambda \\
& \land \lor \land\ x > 4 \\
& \quad \land\ x' = x + 1 \\
& \quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \mu] \\
& \quad \land \text{ UNCHANGED } vars \setminus \{x, pc\} \\
& \quad \lor \land\ \neg(x > 4) \\
& \quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \nu] \\
& \quad \land \text{ UNCHANGED } vars \setminus \{pc\}
\end{aligned}
$$

Outline

Verification results for PlusCal-2 algorithms

- No degradation found in the output of TLC model checker
  - Generated state space doesn't increase
- More natural representation of the algorithms in PlusCal-2
- Except for the TLA$^+$ specifications, they become less readable.
- Users are not supposed to read TLA$^+$ specifications

- Comparison between TLC output for PlusCal and PlusCal-2

Algorithm	# proc.	PlusCal	PlusCal-2
Peterson	2	37	23
FastMutex	2	2679	2679
Naimi-Trehel	3	111749	53905

Outline

Comparison with PlusCal

- Models have become self-contained
  - fairness assumptions, correctness properties or model checking constraints can be expressed within PlusCal-2 algorithm
- Nested processes and scoped declarations
  - represent the locality information
  - increase readability of algorithms
  - less errors are expected while modeling an algorithm
- Representation is more flexible, without losing any performance
  - new statements like **atomic**, **for**,...
  - multiple assignments to same variable in a block

Conclusions and work in progress

- Achievements
  - Easily accessible for algorithm designers
  - No need to read/modify TLA$^+$ specifications
- Ongoing/Future Work
  - Implementation of Partial order reduction for TLC geared towards PlusCal-2 algorithms
  - Implementation of the module for collecting locality information
  - Integration of the module in PlusCal-2

Questions!