Automatic Distributed Code Generation from Formal Models of Asynchronous Concurrent Processes

Hugues Evrard and Frédéric Lang

INRIA and LIG – Team CONVECS

http://convecs.inria.fr
Motivation

Usage of formal methods
- specification
- verification
- hand-written implementation: time consuming, discrepancies with the specification?

Distributed implementation generation
- automatically generates a distributed program
CADP verification toolbox

http://cadp.inria.fr

Explicit state techniques: LTS (Labeled Transition System) exploration

50+ tools for LTS manipulation and verification

– model checking, compositional verification, test generation...

Main specification language: LNT

– LNT semantics formally defined in LTS

EXEC/CAESAR compiler: LNT to sequential C
This talk

Aim: Generate a distributed implementation of a concurrent system from its formal specification (LNT)

Contributions

- DLC (*Distributed LNT Compiler*)
- from LNT to C and TCP sockets
- inter process communication: **multiway rendezvous**
- interaction with environment: **hook functions**
- experimentations
Distributed LNT Compiler overview

- LNT specification
- DLC
- C distributed implementation
- Formal Verification
- CADP
A process performs actions on gates

Process interaction by **multiway rendezvous** on gates
- Gates are named synchronization points
- Multiway: \( n \) processes synchronize \((n \geq 2)\)
- Possibly exchange data by offers

```plaintext
process P1 [A,B] is
  A (1); B
end process

process P2 [A,C] is
  var n:nat in
    A (?n); C(n)
  end var
end process

(* Parallel composition *)
process MAIN [A,B,C] is
  par A in
    P1 [A,B]
  || P2 [A,C]
end par
end process
```

---

**LNT (1)**

PDP(4PAD) – Turku – March 4-6, 2015

Hugues Evrard, Frédéric Lang – Team CONVECS

---
LNT (2)

**Non deterministic choice:** a process is ready on several actions

```
select
  A; ... 
[] B; ... 
[] C; ... 
end select
```

**General synchronization:** multiple rendezvous per gate, \(m\)-among-\(n\)

```
par A, B#2 in
  C -> P1 [A,B,C]   \(\rightarrow\) A: (P1,P2,P3)
|| P2 [A,B]         \(\rightarrow\) B: (P1,P2), (P1,P3), (P2,P3)
||| C -> P3 [A,B,C]  \(\rightarrow\) C: (P1, P3)
end par
```

**Value-matching data exchange:** send/receive freely combined

```
par A in
  A (1, 2, \(?x\):nat, \(?b1\):bool)  A !1 !2 !3 !true
||| A (1, \(?y\):nat, 3, \(?b2\):bool)  A !1 !2 !3 !false
end par
```
Distributed LNT Compiler overview

LNT specification

Task || Task || ... || Task

DLC

EXEC/CAESAR

rendezvous protocol

distributed code generator

C implementation

Gate

Gate

Task

Task

Task
LNT to sequential C

EXEC/CAESAR [Garavel-Viho-Zendri-01]
- C program that explores an execution path inside the LTS
- list possible actions from current state
- still requires code to decide between actions

DLC automatically completes the C program of each task
- task may synchronize with others for actions
- adds code to interface with the rendezvous protocol
Multiway rendezvous protocol

Role: synchronize task actions
- a task can be ready on different actions: \texttt{select A [] B [] ...}
- all tasks of a rendezvous must commit to the \textit{same} action

Requirement: \textbf{distributed} implementation
- avoid unique central synchronizer (performance bottleneck)
- protocol relies on \textit{asynchronous message passing} (TCP: reliable, ordered)

Solution: based on \cite{Parrow-Sjodin-96}
- One process per \textit{gate}, one process per \textit{task}
- extended: \textit{data exchange, general synchronization}
par A, B in
T1 [A,B]
|| T2 [A,B,C]
end par

process T1 [A,B] is
    select
    A
    [] B
    end select
end process

process T2 [A,B,C] is
    select
    A
    [] B
    [] C; B
    end select
end process

- **Three phases**
  1. **Announces**: ready
  2. **Negotiations**: lock (ordered to avoid deadlock)
  3. **Results**: commit/abort
Rendezvous protocol illustration

\begin{verbatim}
par A, B in
  T1 [A,B]
|| T2 [A,B,C]
end par

process T1 [A,B] is
  select
    A
    [] B
  end select
end process

process T2 [A,B,C] is
  select
    A
    [] B
    [] C; B
  end select
end process
\end{verbatim}

- **Three phases**
  1. **Announces**: ready
  2. **Negotiations**: lock (ordered to avoid deadlock)
  3. **Results**: commit/abort
**Rendezvous protocol illustration**

```plaintext
par A, B in
  T1 [A,B]
|| T2 [A,B,C]
end par

process T1 [A,B] is
  select
    A
  [] B
end select
end process

process T2 [A,B,C] is
  select
    A
  [] B
  [] C; B
end select
end process
```

- **Three phases**
  1. **Announces**: ready
  2. **Negotiations**: lock (ordered to avoid deadlock)
  3. **Results**: commit / abort
Rendezvous protocol illustration

\[
\text{par } A, B \text{ in } \\
\text{T1 } [A,B] \\
|| \text{T2 } [A,B,C] \\
\text{end par}
\]

\[
\text{process } T1 \ [A,B] \text{ is} \\
\quad \text{select} \\
\quad \quad A \\
\quad \quad \begin{array}{c} [\ ] \ B \\
\quad \text{end select} \\
\quad \text{end process}
\]

\[
\text{process } T2 \ [A,B,C] \text{ is} \\
\quad \text{select} \\
\quad \quad A \\
\quad \quad \begin{array}{c} [\ ] \ B \\
\quad \quad [\ ] \ C; \ B \\
\quad \text{end select} \\
\quad \text{end process}
\]

Three phases

1. Announces: ready
2. Negotiations: lock (ordered to avoid deadlock)
3. Results: commit / abort
par A, B in
    T1 [A,B]
|| T2 [A,B,C]
end par

process T1 [A,B] is
    select
    A
    [ ] B
end select
end process

process T2 [A,B,C] is
    select
    A
    [ ] B
    [ ] C; B
end select
end process

Three phases

1. Announces: ready
2. Negotiations: lock (ordered to avoid deadlock)
3. Results: commit / abort
Rendezvous protocol illustration

par A, B in
  T1 [A,B]
|| T2 [A,B,C]
end par

process T1 [A,B] is
  select
    A
  [] B
end select
end process

process T2 [A,B,C] is
  select
    A
  [] B
  [] C; B
end select
end process

Three phases
1. Announces: ready
2. Negotiations: lock (ordered to avoid deadlock)
3. Results: commit / abort
Rendezvous protocol protocol illustration

```
par A, B in
  T1 [A,B]
|| T2 [A,B,C]
end par

process T1 [A,B] is
  select
    A
  [] B
  end select
end process

process T2 [A,B,C] is
  select
    A
  [] B
  [] C; B
  end select
end process
```

- **Three phases**

  1. **Announces:** ready
  2. **Negotiations:** lock (ordered to avoid deadlock)
  3. **Results:** commit / abort
Protocol correctness

- Formal verification with LNT and CADP [Evrard-Lang-13]
  - model checking: absence of protocol livelock/deadlock
  - equivalence checking between implementation & specification

  "The devil is in the detail"
  - [Parrow-Sjodin-96] can deadlock with asynchronous messages, fixed version in [Evrard-Lang-13]
  - [Perez-04] can deadlock, fixed version in [Katz-Peled-10]

- DLC version of the protocol
  - protocol logic isolated in a library
  - only glue code that calls protocol is generated for each specification
Interaction with environment: hook functions

- Optional user-defined C function
  - side effects, call external software, ...
- Triggered by actions
Three types of hook functions

- **pre-negotiation hook** (one per gate)
  - called *before* starting a negotiation
  - return bool: authorize negotiation or not
  - role: prevent useless negotiation to hamper others

- **post-negotiation hook** (one per gate)
  - called on *negotiation success* (tasks are locked, no commit yet)
  - return bool: authorize action or not
  - role: final decision to realize action

- **local hook** (one per task)
  - called by *each task* that realizes an action
  - role: local side effects at the task level
Exchange data with the environment

- Hook functions access data offers

- From **system to environment**
  - offer in **send** mode: \( A (x) \)

- From **environment to system**
  - offer in **receive** mode: \( A (\, ?x) \)
  - pre-/post-negotiation hook must **define** the offer

  ```cpp
  action->offer[0]->value = get_value(); // external function
  action->offer[0]->mode = DLC_MODE_SEND;
  ```

  - check at runtime: all offers in **send** mode at the time of negotiation commit
Experimentation: synchro. barrier

- Experimentations on Grid5000
- Compare inter-process interaction perf. with C, Java, ... 
  - They lack multiway rendezvous (neither built-in nor mature lib)
  - Use distributed synchronization barrier for comparison

![Graph showing comparison of different methods for synchronization](chart.png)

(smaller is better)
Experimentation: Raft consensus

Raft [Ongaro-Ousterhout-14] consensus algorithm (like Paxos)
- used to build fault tolerant service (replicated state machine)

Service: storage (read, write)

LNT Raft cluster
- use hook functions to implement access from external client
- 1000 writes replicated on 7 servers: 14000 actions in 5.5 sec

Comparison with Consul (in Golang, by Hashicorp)
- Consul is 10 times faster... batches requests!
- Raft-level optimization that DLC cannot handle yet
Conclusions

Generate a distributed implementation from an LNT specification
Reuse CADP tools for sequential, focus on distribution
Multiway rendezvous implemented by a verified protocol
Hook functions enable interaction with environment
Acceptable performances for rapid prototyping
new tool DLC: automatized compilation, “push button”
Future work

More case studies of distributed algorithms
- model in LNT, verify with CADP, compile with DLC

Improve rendezvous protocol performances (e.g. detect synchronization patterns that require less messages)

Handle more LNT constructions
- complex data types (record, list, array...) in action offers
- guarded actions: \( A (?x) \text{ where } x > 0 \)
“Designing a Multiway Synchronization Protocol”

Extract:

2 A Model of Multiway Synchronization

We assume that there are \( n \) processes, in the following called *client processes*, communicating over a communication network providing asynchronous point-to-point connections (see Fig. 1). Thus messages will arrive after an unknown and variable delay, and the network never loses messages.

We further assume that there are several multiway synchronization actions, denoted \( a, b \).
Experimentation: Raft consensus

Raft consensus algorithm [Ongaro-Ousterhout-14]
– similar to Paxos, but easier to understand, hence to implement
– used to build fault tolerant service (replicated state machine)

Service: storage (read, write)

LNT Raft cluster
– use hook functions to implement access from external client
– 1000 writes replicated on cluster of 7 servers: 5.5 sec
– triggers 14000 actions: approx. 0.4ms per action

Comparison with Consul (in Golang, by Hashicorp)
– Consul is 10 times faster... batches requests!
– Raft-level optimization that DLC cannot handle yet
Rendezvous protocol illustration

par A, B in
  T [A,B]
|| T [A,B]
end par

process T [A,B] is
  select A [ ] B end select
end process

Three phases

1. Announces
2. Negotiations (ordered lock to avoid deadlock)
3. Results
Three phases

1. **Announces**
2. **Negotiations** (ordered lock to avoid deadlock)
3. **Results**
Rendezvous protocol illustration

par A, B in
        T [A,B]
|| T [A,B]
end par

process T [A,B] is
        select A [] B end select
end process

Three phases

1. Announces
2. Negotiations (ordered lock to avoid deadlock)
3. Results
Rendezvous protocol illustration

par A, B in
 T [A,B]
|| T [A,B]
end par

process T [A,B] is
 select A [] B end select
end process

Three phases
1. Announces
2. Negotiations (ordered lock to avoid deadlock)
3. Results
Rendezvous protocol illustration

par A, B in 
  T [A,B]
|| T [A,B]
end par

process T [A,B] is 
  select A [] B end select
end process

Three phases

1. Announces
2. Negotiations (ordered lock to avoid deadlock)
3. Results