Communication Safe Parallel Programming with Session Types

Nicholas Ng and Nobuko Yoshida

Department of Computing, Imperial College London

Imperial College London

Motivation

- Parallel architectures
  - Utilise hardware resources well
  - Correct parallel programs difficult to write
- Common issues
  - Communication mismatch
    (i.e. send without matching receive or vice versa)
  - Lead to communication deadlocks
  - Difficult to debug and detect
- State of the art techniques
  - Model checking or symbolic execution [1]
  - Suffers from state explosion
  - Completeness: relies on heuristics to reduce state space

Our approach: Session Types

- Multiparty Session Types [2] (MPST)
  - Formal typing system for communication
  - Exploits duality between communication
  - Guarantees communication safety and deadlock freedom
- Seq. of communication abstracted as **sessions**

  Global types describe global interactions between participants interleaved with global control flow of program

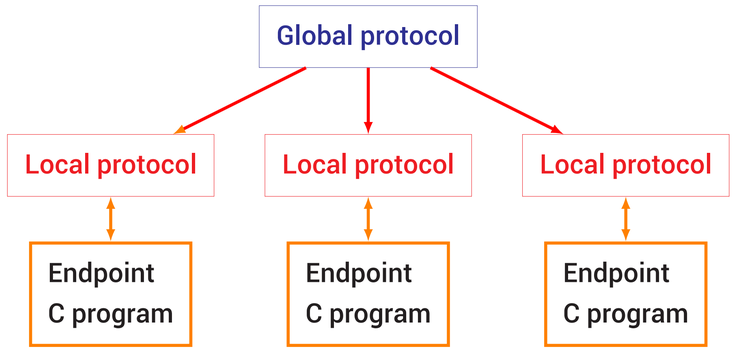
  Projection converts Global types to Endpoint types

  Endpoint types are localised types at endpoints
- **Static type checking**
  - Overcomes shortcomings of model checking techniques
  - Fully guarantees communication safety in all execution path

Session C programming framework

We introduce a programming framework [3] following closely the workflow of MPST:

1. Design global communication interaction
2. Project into local protocol for endpoints
3. Implement using local protocol as specification
4. Type check program against endpoint protocol



- Uses MPST-based protocol desc. language **Pabble**
- If type checking succeeds
  i.e. Endpoint program follows local protocol, then
  1. Communication between the programs are compatible
  2. There are no communication deadlocks

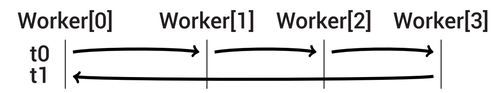
References

G. Gopalakrishnan, R. M. Kirby, S. Siegel, R. Thakur, W. Gropp, et al.
Formal analysis of MPI-based parallel programs.
Commun. ACM, 54(12):82–91, Dec. 2011.

K. Honda, N. Yoshida, and M. Carbone.
Multiparty asynchronous session types.
In *POPL'08*, volume 5201 of *LNCS*, pages 273–284, 2008.

N. Ng, N. Yoshida, and K. Honda.
Multiparty Session C : Safe Parallel Programming with Message Optimisation.
In *TOOLS 2012*, volume 7304 of *LNCS*, pages 202–218, 2012.

Example: Communication Protocol in Pabble




```
global protocol Ring (role Worker[0..3]) {
  rec LOOP {
    Data(int) from Worker[i:0..2] to Worker[i+1];
    Data(int) from Worker[3] to Worker[0];
    continue LOOP; }}
```

Convert to local protocol by Projection

```
local protocol Ring at Worker (role Worker[0..4]) {
  rec LOOP {
    if Worker[i:1..3] Data(int) from Worker[i-1];
    if Worker[i:0..2] Data(int) to Worker[i+1];
    if Worker[0]    Data(int) from Worker[3];
    if Worker[3]    Data(int) to Worker[0];
    continue LOOP; }}
```

Example: Endpoint MPI Source code to type check

```
while (i++<10) {
  if (rank < 3) MPI_Isend(rank+1); //This reordering valid
  if (rank == 3) MPI_Isend(0); // by 'subtyping relation'
  if (rank > 0) MPI_Recv(rank-1);
  if (rank == 0) MPI_Recv(3);
}
```

Key Challenges

- Extract protocols from common MPI coding patterns accurately
- Extending session typing system for practical use cases
- Inferring global protocol from extracted protocols