Dynamic Symbolic Verification of MPI Programs

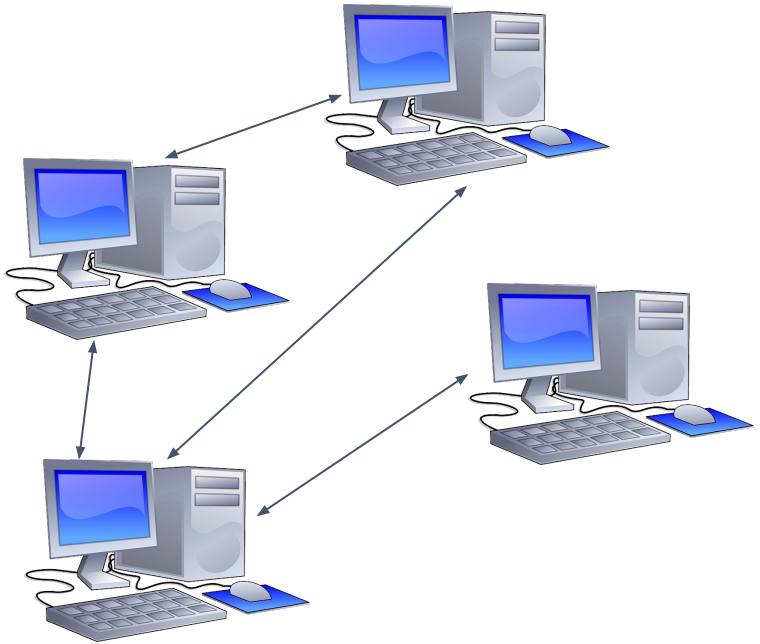
Dhriti Khanna[*], Subodh Sharma[#], César Rodríguez[$], Rahul Purandare[*]

International Symposium on Formal Methods
July 2018
Oxford, UK

* IIIT Delhi | # IIT Delhi | $ University of Paris

Motivation

- Key computing trend
- Communication mechanism
- Non-determinism
- **Deadlocks**
- **Hard problem**



Distributed Computing

Motivation

Process 1

```
Recv(*, x); //R1
if (x==10)
  Recv(*, y); //R2
else if (x==20)
  Recv(*, y); //R3
else
  { }
Recv(*, z); //R5
```

Process 2

```
Send(P1, 10);
```

Process 3

```
Send(P1, 10);
```

Process 4

```
Send(P1, 30);
```

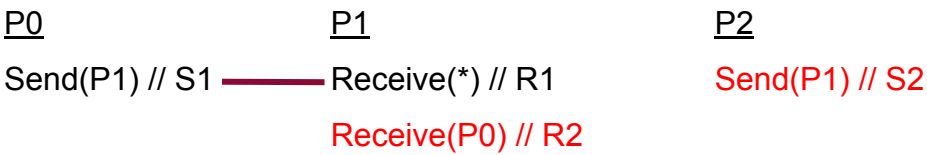
Aim of the work: to find deadlocks in message passing programs

Challenging because: non-determinism which leads to explosion in interleavings

MPI: Message Passing Interface

- Parallel programs consist of separate processes, each with its own address space
- Data sent explicitly between processes
- Point-to-point operations or Collective operations

```
MPI_SEND (start, count, datatype, dest, tag, comm)
MPI_RECV (start, count, datatype, source / *, tag, comm, status)
```



Automated verification of multi-path MPI programs

Process 1

```
Recv(*, x); //R1
if (x==10)
  Recv(*, y); //R2
else if (x==20)
  Recv(*, y); //R3
else
  { }
Recv(*, z); //R5
```

Process 2

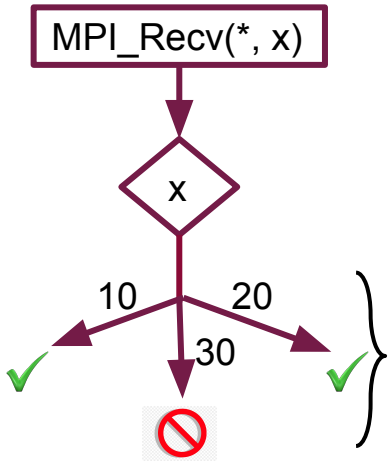
```
Send(P1, 10); //S1
```

Process 3

```
Send(P1, 10); //S2
```

Process 4

```
Send(P1, 30); //S3
```



Multi-path program:

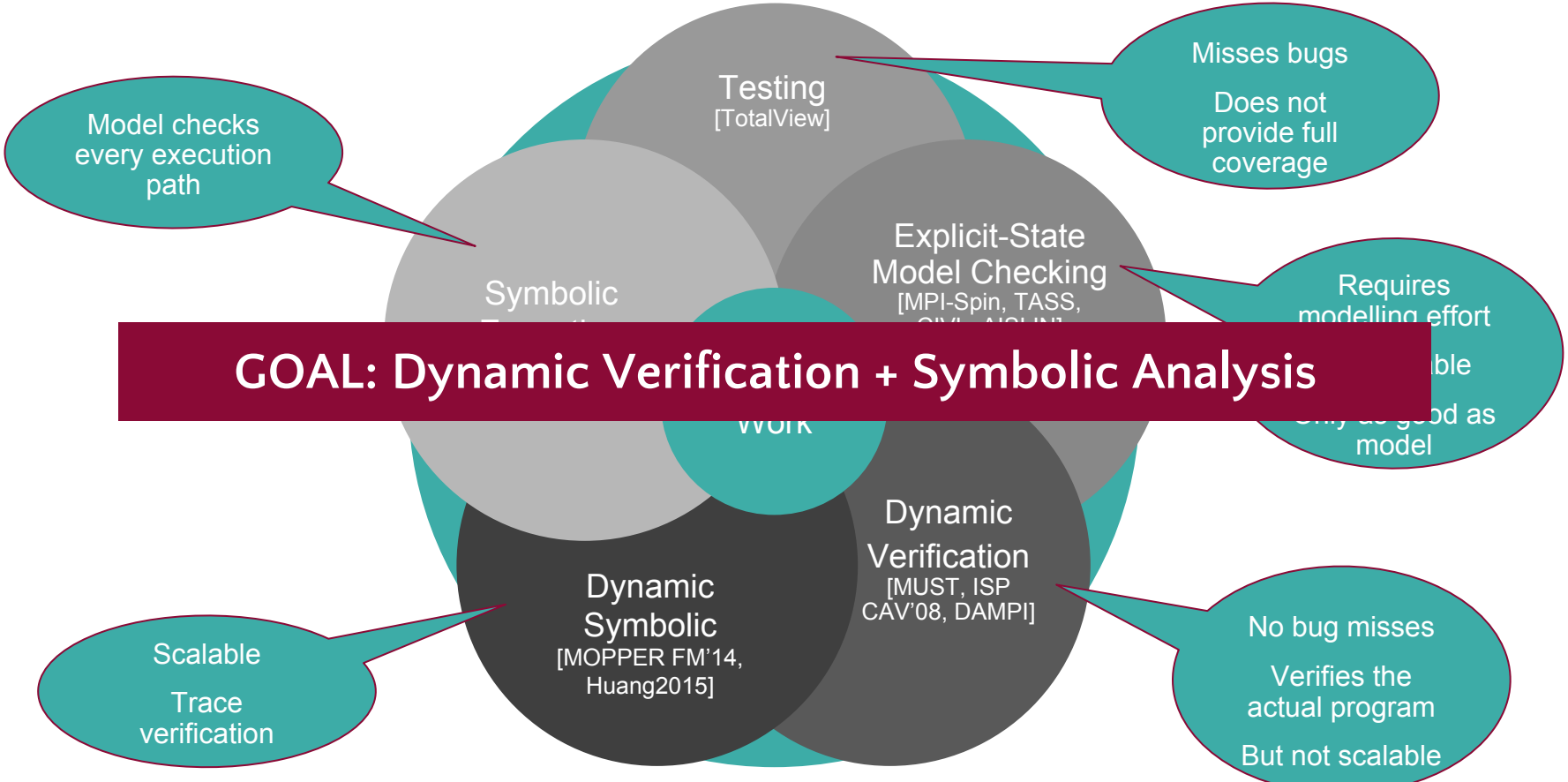
Multiple control flows based upon the data received in a prior wildcard Receive call

Non-determinism can exist because of:

1. Concurrency
2. Data
   - Control

Aim

Current approaches in MPI landscape



Example

Process 1

```
Recv(*, x); //R1
if (x==10)
  Recv(*, y); //R2
else if (x==20)
  Recv(*, y); //R3
else
  Recv(*, y); //R4
Recv(*, z); //R5
```

Process 2

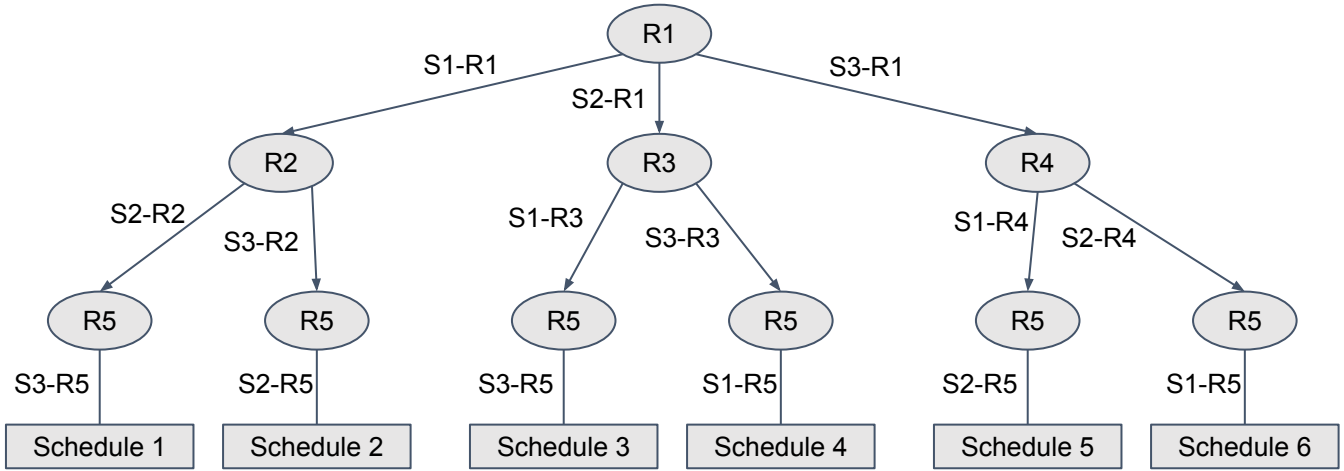
```
Send(P1, 10); //S1
```

Process 3

```
Send(P1, 10); //S2
```

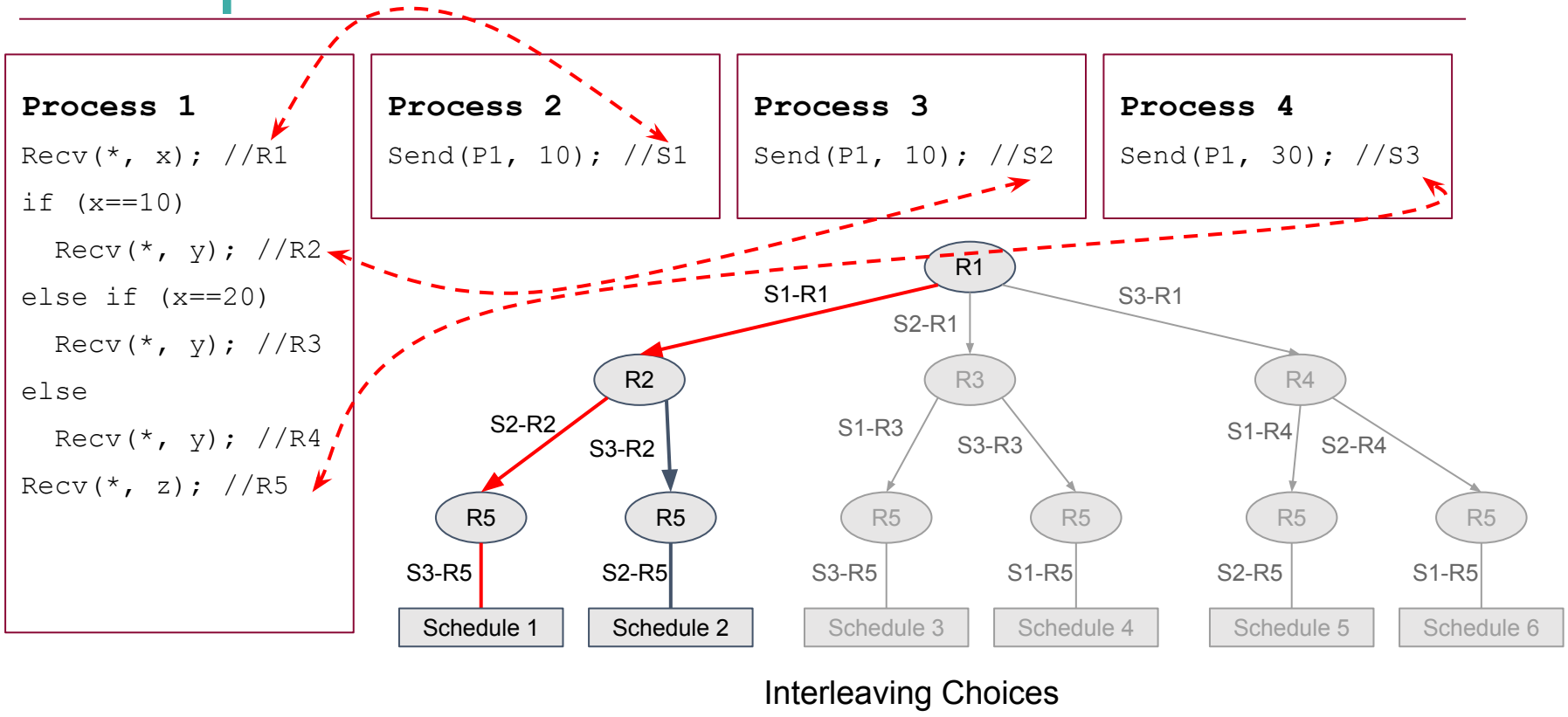
Process 4

```
Send(P1, 30); //S3
```



Interleaving Choices

Example



Interleaving Choices

Example

Process 1

```
Recv(*, x); //R1
if (x==10)
  Recv(*, y); //R2
else if (x==20)
  Recv(*, y); //R3
else
  Recv(*, y); //R4
Recv(*, z); //R5
```

Process 2

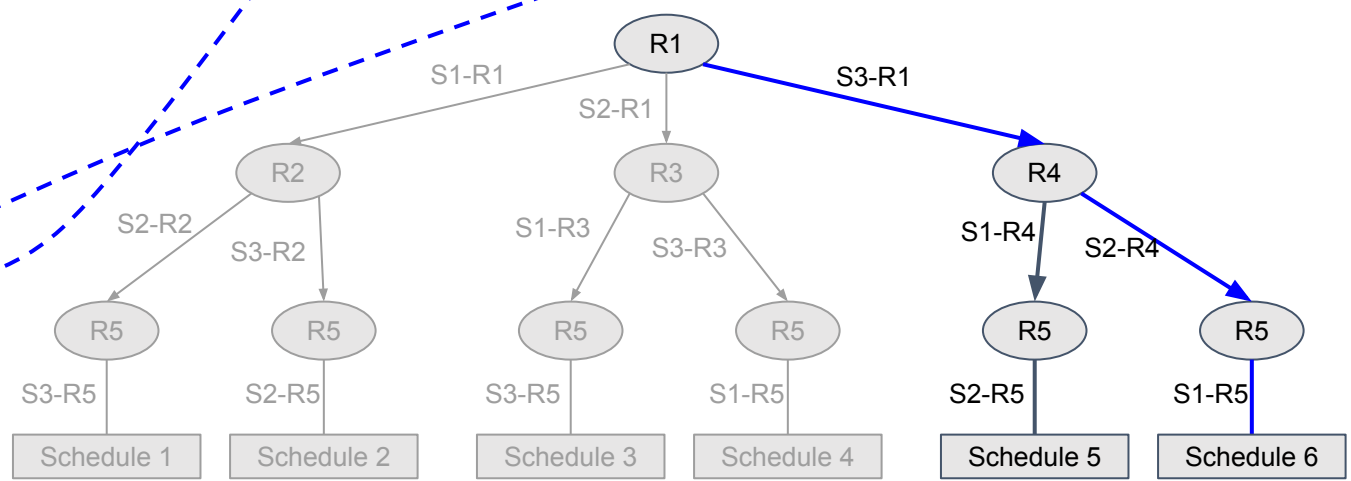
```
Send(P1, 10); //S1
```

Process 3

```
Send(P1, 10); //S2
```

Process 4

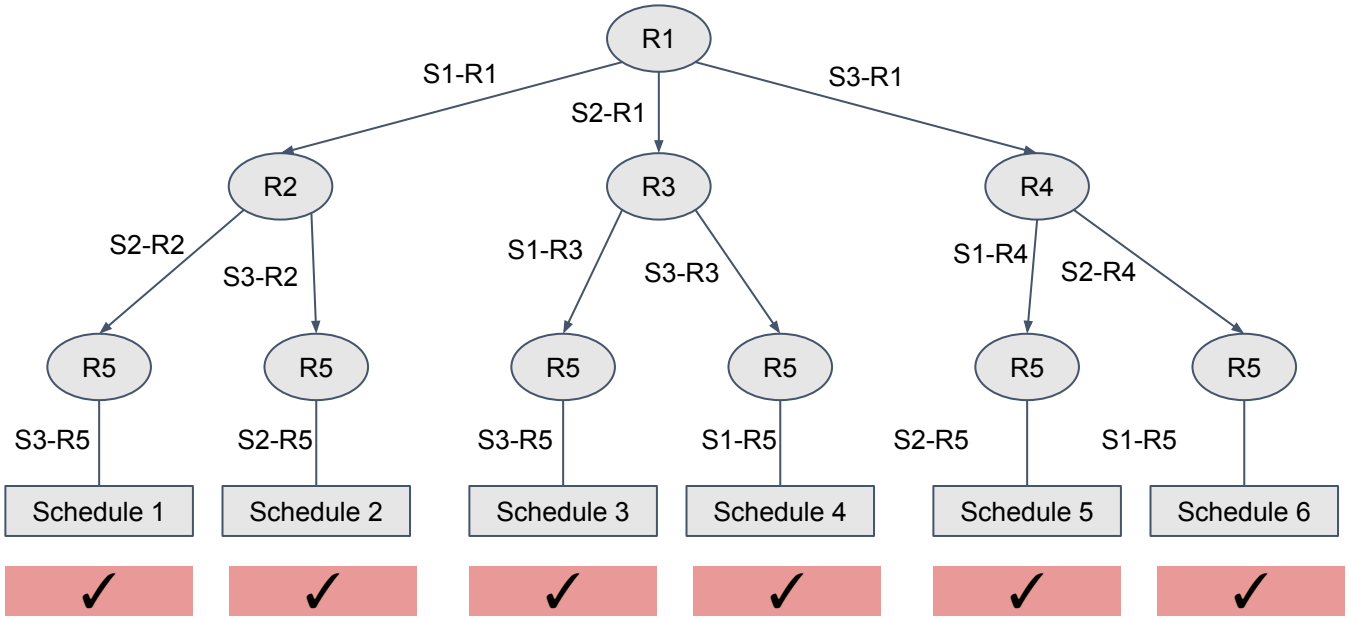
```
Send(P1, 30); //S3
```



Interleaving Choices

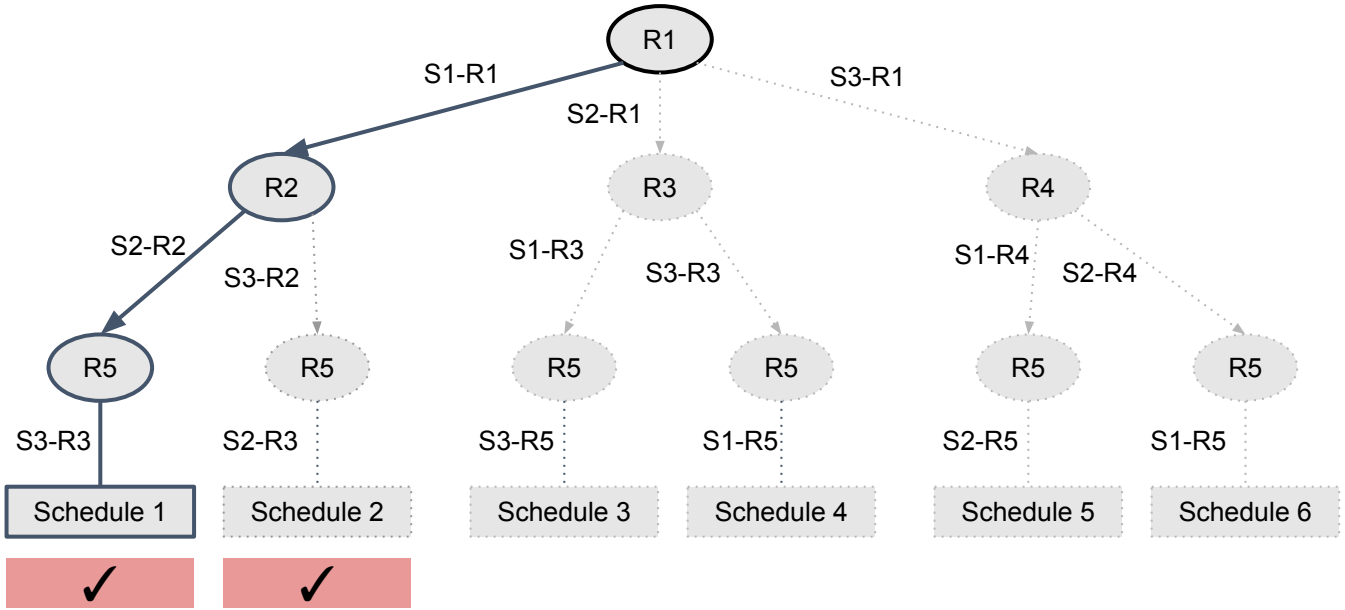
1. What dynamic verification does?

Analyzes all the interleavings one by one.



2. What trace verifiers do?

Analyzes two interleavings.

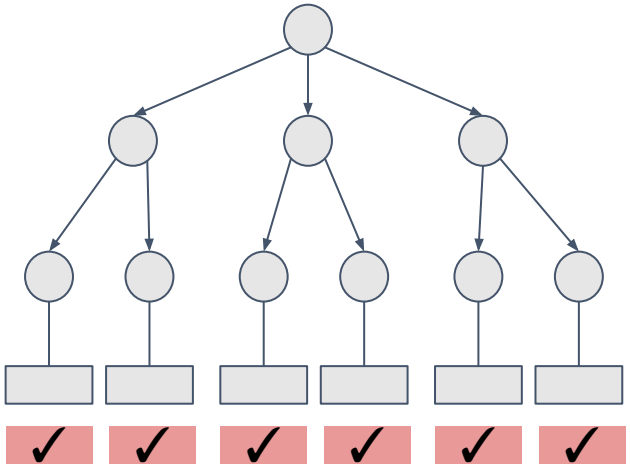


Our hybrid approach

Dynamic Verification

+

Constraint Solving



$$\bigwedge_{s \in S} clk_s < clk_{Imm(s)}$$

$$\bigwedge_{a:(a_1,a_2,\ldots,a_n) \in M^+} s_a \rightarrow (clk_{a_1} = clk_{a_2} = \ldots = clk_{a_n})$$

$$\bigwedge_{(a,b) \in M^+} \bigwedge_{(a,c) \in M^+} s_{ab} \rightarrow !s_{ac}$$

$$\bigwedge_{(a,b) \in M^+} \bigwedge_{(c,b) \in M^+} s_{ab} \rightarrow !s_{cb}$$

$$\bigwedge_{(a,b) \in E} e_{ab}$$

$$\bigwedge_{(c,a);(c,b) \in M^+;(a,b) \in E} e_{ab} \rightarrow s_{ca} \wedge !s_{cb}$$

$$\bigwedge_{a \in r} (m_a \rightarrow \bigvee_{(b,a) \in M^+} (s_{ba})) \wedge \bigwedge_{a \in s} (m_a \rightarrow \bigvee_{(a,b) \in M^+} (s_{ab}))$$

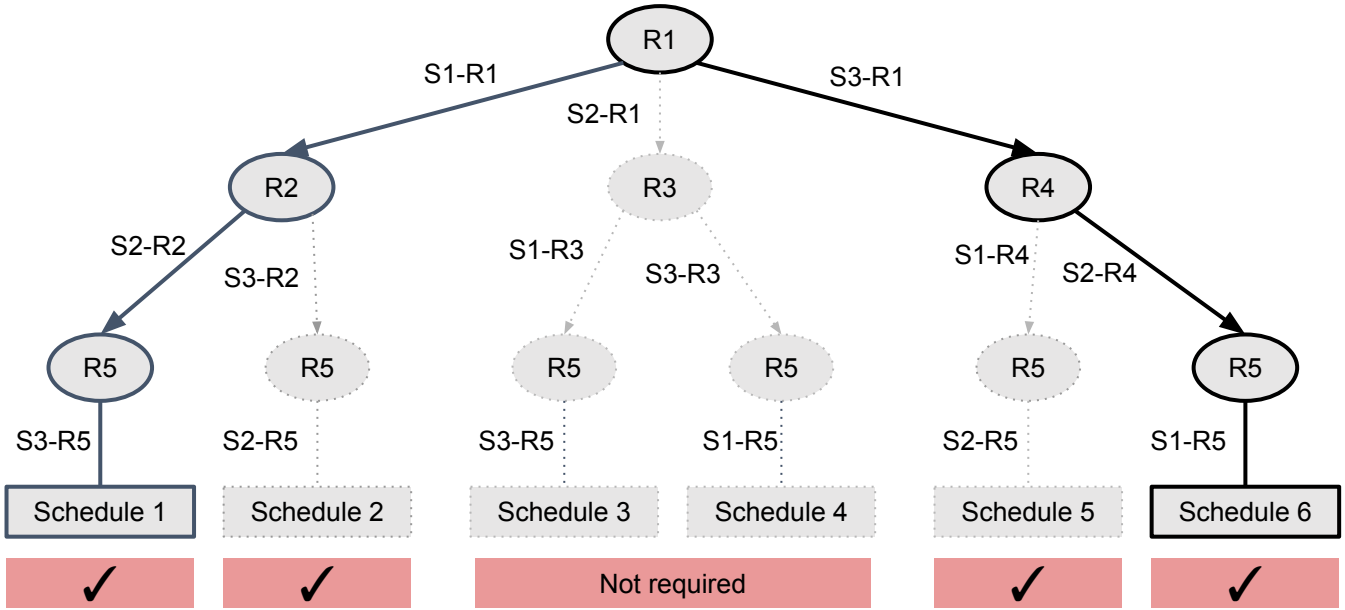
$$\bigwedge_{a:(a_1,a_2,\ldots,a_n) \in M^+} s_a \rightarrow \bigwedge_{a_i \in a} m_{a_i}$$

$$\bigwedge_{b \in S} (r_b \leftrightarrow \bigwedge_{a = Imm(b)} m_a)$$

$$\bigwedge_{a \in S} (m_a \rightarrow r_a)$$

Our approach

Executes the program twice but verifies all interleavings.



Example

Process 1

```
Recv(*, x); //R1
if (x==10)
  Recv(*, y); //R2
else if (x==20)
  Recv(*, y); //R3
else
  Recv(*, y); //R4
Recv(*, z); // R5
```

Process 2

```
Send(P1, 10); //S1
```

Process 3

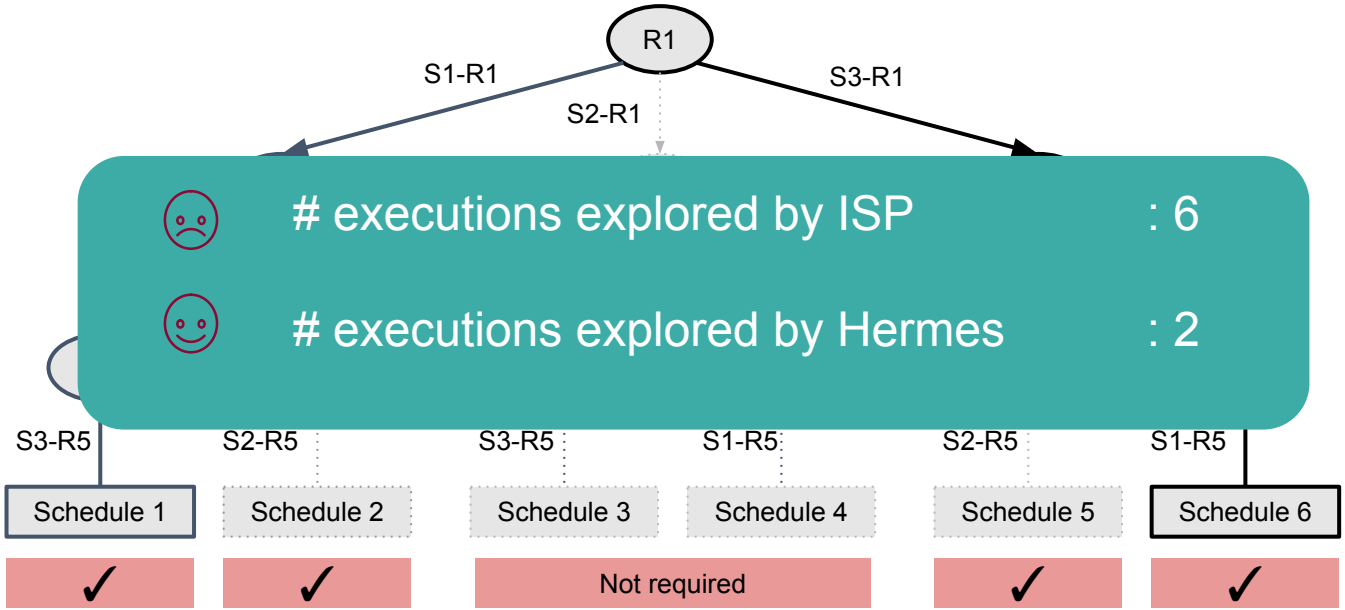
```
Send(P1, 10); //S2
```

Process 4

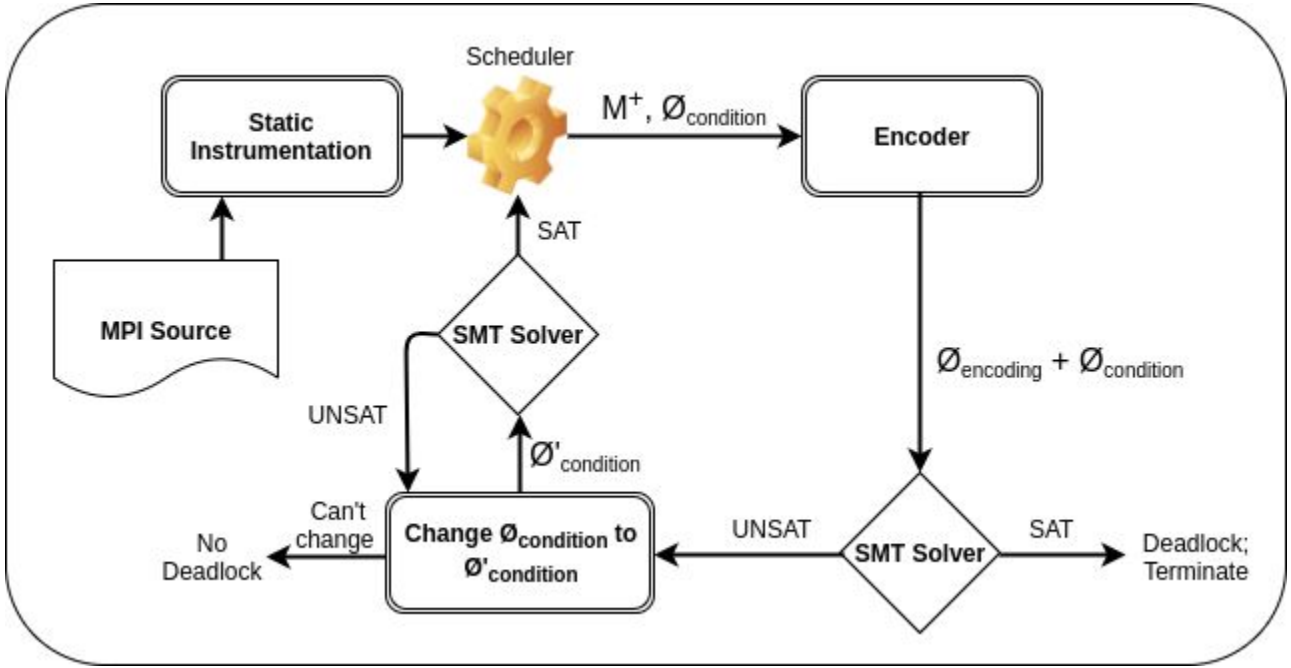
```
Send(P1, 30); //S3
```

Our approach

Executes the program twice but verifies all interleavings.

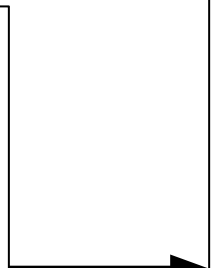


Overview of our approach



1. Program instrumentation

```
1.   Recv(*, x);
2.   if (x==10)
3.     Recv(*, y);
4.   else if (x==20)
5.     Recv(*, y);
6.   else
7.     Recv(*, y);
8.   Recv(*, z);
```



```
1.   Recv(*, x);
2.   if (x==10) {
3.     toScheduler('x', '==', '10', '1');
4.     Recv(*, y);
5.   }
6.   else if (x==20) {
7.     toScheduler('x', '==', '20', '1');
8.     Recv(*, y);
9.   }
10.  else {
11.    toScheduler('x', '!=', '20', '1');
12.    toScheduler('x', '!=', '10', '1');
13.    Recv(*, y);
14.  }
15.  Recv(*, z);
```

2. Trace, $<_{mo}$, and M^+ extraction

```
Recv(*, x);          /*** R1 ***/

if (x==10) {
  toScheduler('x', '==', '10', '1');
  Recv(*, y);        /*** R2 ***/
}
else if (x==20) {
  toScheduler('x', '==', '20', '1');
  Recv(*, y);
}
else {
  toScheduler('x', '!=', '20', '1');
  toScheduler('x', '!=', '10', '1');
  Recv(*, y);
}

Recv(*, z);          /*** R3 ***/
```

```
Send(P1,10);
/*** S1 ***/
```

```
Send(P1,10);
/*** S2 ***/
```

```
Send(P1,30);
/*** S3 ***/
```

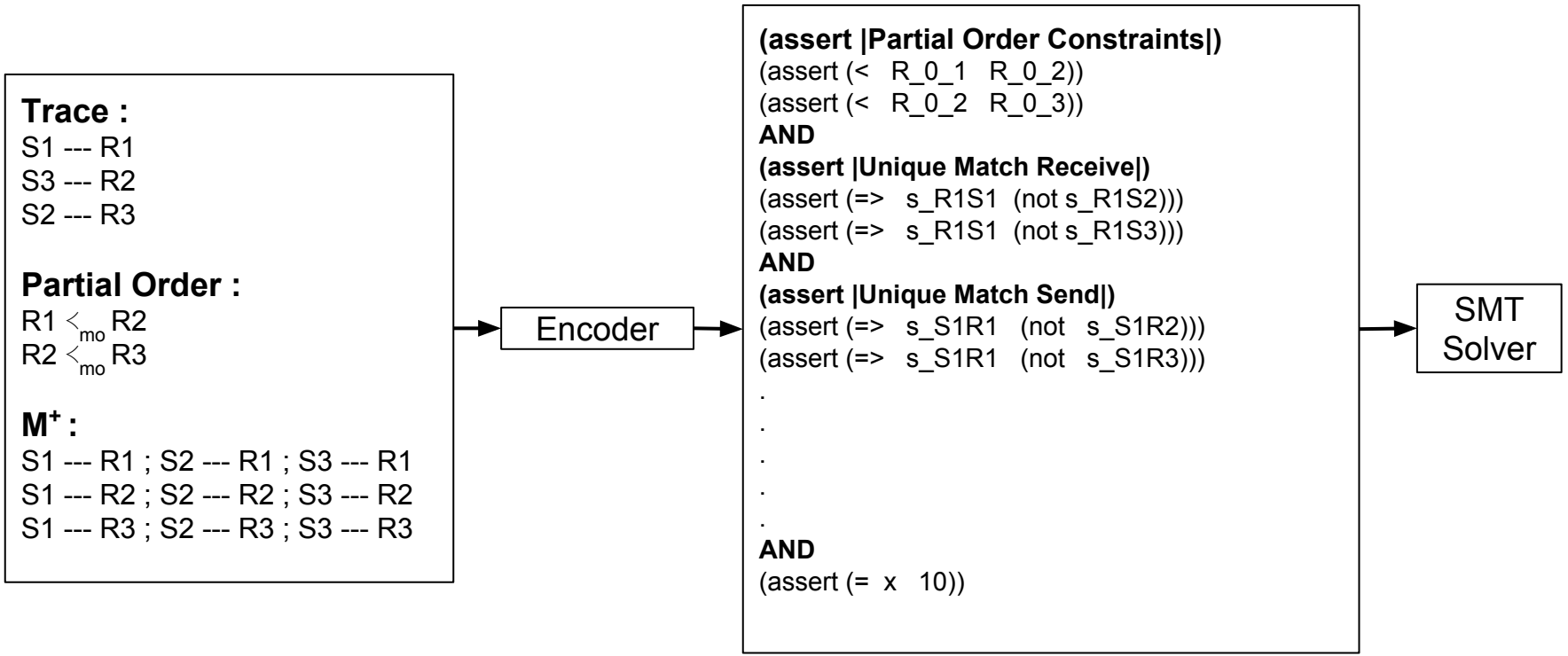
Scheduler

Trace :
S1 --- R1
S3 --- R2
S2 --- R3

Partial Order :
R1 $<_{mo}$ R2
R2 $<_{mo}$ R3

M^+ :
S1 --- R1 ; S2 --- R1 ; S3 --- R1
S1 --- R2 ; S2 --- R2 ; S3 --- R2
S1 --- R3 ; S2 --- R3 ; S3 --- R3

3. Encoding


Trace :
S1 --- R1
S3 --- R2
S2 --- R3
Partial Order :
R1 <mo R2
R2 <mo R3
M+ :
S1 --- R1 ; S2 --- R1 ; S3 --- R1
S1 --- R2 ; S2 --- R2 ; S3 --- R2
S1 --- R3 ; S2 --- R3 ; S3 --- R3
Encoder
(assert |Partial Order Constraints|)
(assert (< R_0_1 R_0_2))
(assert (< R_0_2 R_0_3))
AND
(assert |Unique Match Receive|)
(assert (=> s_R1S1 (not s_R1S2)))
(assert (=> s_R1S1 (not s_R1S3)))
AND
(assert |Unique Match Send|)
(assert (=> s_S1R1 (not s_S1R2)))
(assert (=> s_S1R1 (not s_S1R3)))
.
.
.
.
.
AND
(assert (= x 10))
SMT Solver

Evaluation

- **HERMES**: Dynamic Verification + Symbolic Analysis
  - https://github.com/DhritiKhanna/Hermes
- Compared Single-path and Multi-path programs with tools:
  - CIVL [Luo et al. EuroMPI'17] : Model Checker
  - ISP [Vakkalanka et al. CAV'07] : Dynamic Verification
  - Mopper [Foreijt et al. FM'14] : Trace Verification
  - Aislinn [Bohm et al. FM'16] : Dynamic Verification

Results: Single-path programs

B'mark	# Proc	ISP		Mopper	Aislinn Time (s)	CIVL Time (s)	HERMES Time (s)
		Runs	Time (s)				
Floyd	4	1	0.005	0.020	0.640	TO	**0.78**
	8	>1.6K	TO	0.116	1.391	TO	**0.218**
DTG	5	3	2.135	0.007	0.830	8.72	**0.036**
	8	3	2.220	0.011	1.135	8.78	**0.40**
GE	4	1	0.529	0.210	8.936	TO	**0.300**
	8	1	0.371	0.295	14.322	TO	**0.423**
	16	1	2.041	0.408	TO	TO	**0.659**
	32	1	5.457	0.856	TO	TO	**1.163**

* : Benchmark not supported
TO : Time out after 1800 secs

Results: Multi-path programs

B'mark	# Proc	ISP		Aislinn Time (s)	CIVL Time (s)	HERMES	
		Runs	Time (s)			Runs	Time (s)
Matrix Multiply	4	36	39.67	0.87	17.93	**1**	**7.25**
	5	144	163.28	1.10	25.31	**1**	**10.00**
	6	720	837.64	1.33	48.07	**1**	**12.93**
	8	~1.5K	TO	2.21	258.86	**1**	**19.67**
Monte	4	6	6.03	0.97	*	**3**	**2.33**
	5	24	28.35	1.67	*	**4**	**3.47**
	6	120	151.60	5.03	*	**5**	**4.82**
	8	>1.2K	TO	10.17	*	**7**	**7.43**

* : Benchmark not supported
TO : Time out after 1800 secs

Future work

- Handle input non-determinism
- Sequential analysis

Questions!

Thanks to TCS fellowship for sponsoring my Ph.D.

Thanks to ACM-India and FLoC 2018 for partly sponsoring this travel.