



Instituto Superior Técnico

The Model Checker SPIN

Presentation by Group 10

Daniel Da Palma Pereira 99194

Matheus Trindade 105471

Pedro Gomes 96902

Overview

01 Introduction

02 Proposed Solution

03 Evaluation

04 Related Work

05 Conclusion

Introduction

What does the article study?

Practical Constraints:

- Physical limitations:
 - problem size;
 - machine memory;
 - maximum runtime;
- Neglected issue of these constraints in formal verification.
- Handle problems outside the normal domain of exhaustive proof.

What is SPIN?

- Efficient verification system
- Supports the design of asynchronous
- Distinguishes by focus on asynchronous control in software

Goals of SPIN

What does SPIN provide?

1. Notation for specifying design choices unambiguously, without implementation detail,
2. A powerful, concise notation for expressing general correctness requirements
3. Establishing the logical consistency of the design choices from 1) and the matching correctness requirements from 2).

01

Language and Specifications

PROMELA (Process Meta Language) is a verification language which SPIN uses to accept design specifications.

LTL Parsers: Stands for Linear Temporal Logic and SPIN uses to accept correctness claims.

Correctness claims: desired properties that must hold true for system to be considered correct.

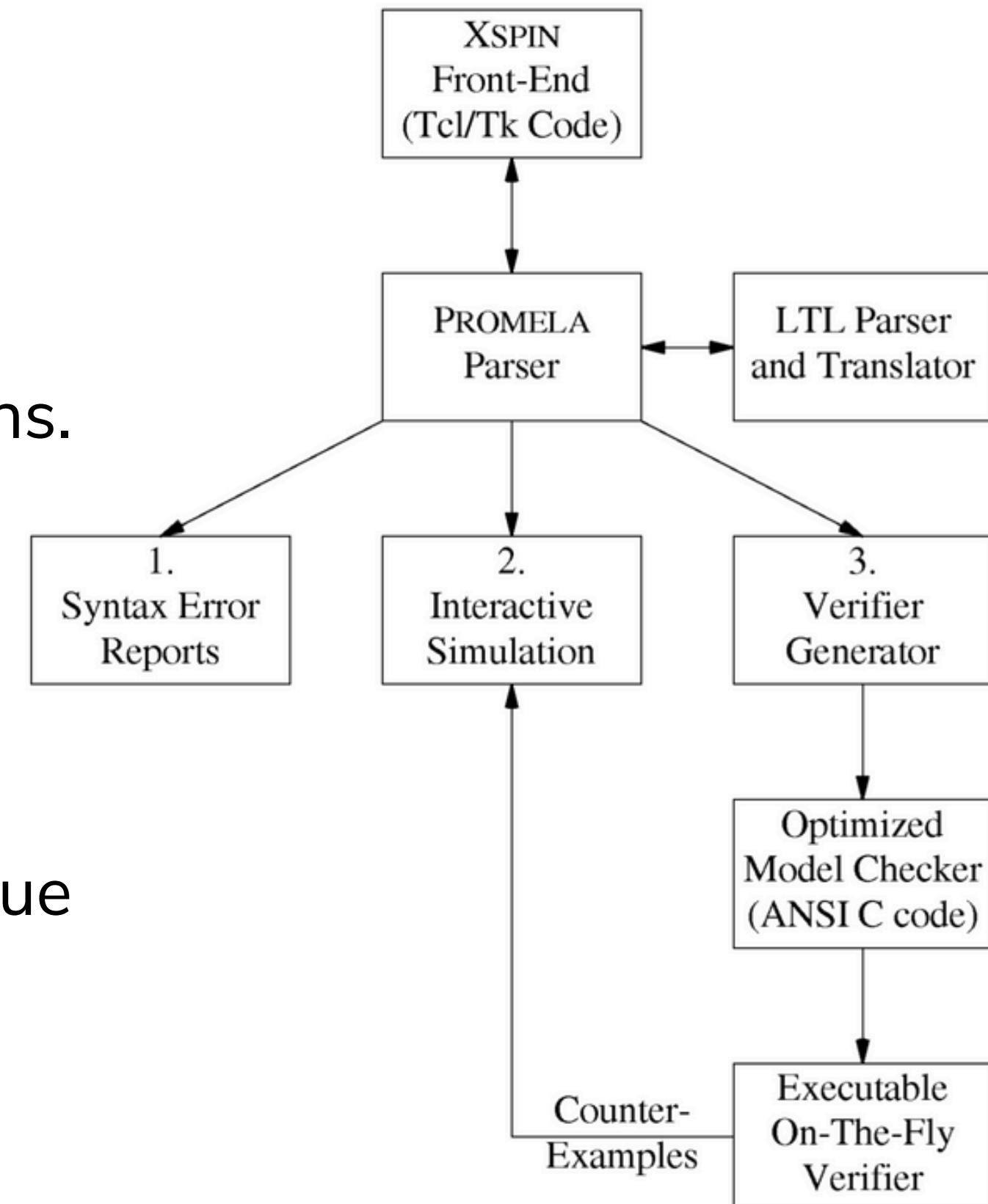


Fig. 1. The structure of SPIN simulation and verification.

- **XSPIN Step:** Specification of a model of a concurrent system, or distributed algorithm, using SPIN's graphical front-end XSPIN.
- **First Step:** Fix syntax errors
- **Second Step:** Interactive simulation is performed.
- **Third Step:** Generate an optimized on-the-fly verification program from the high level specification.

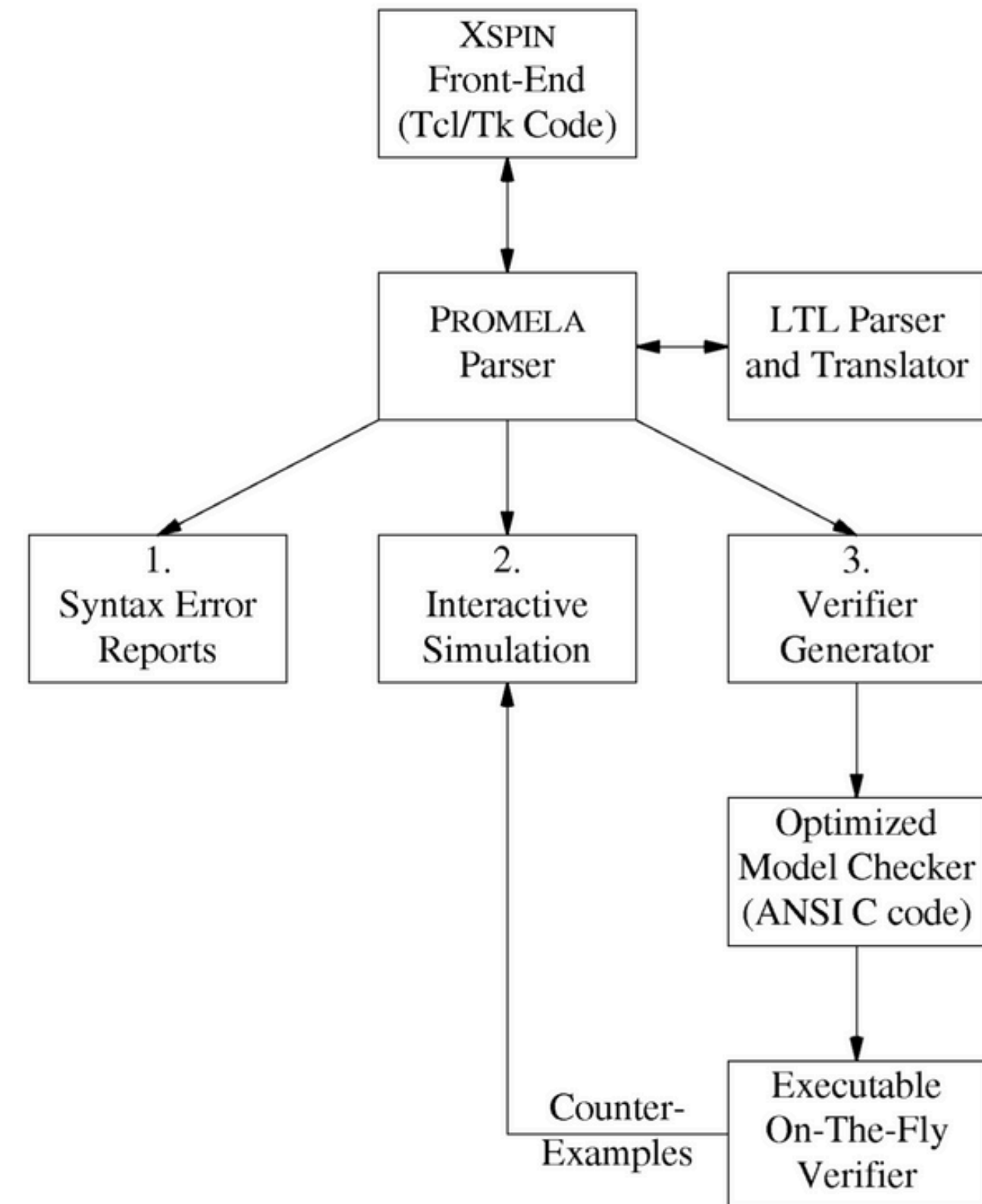
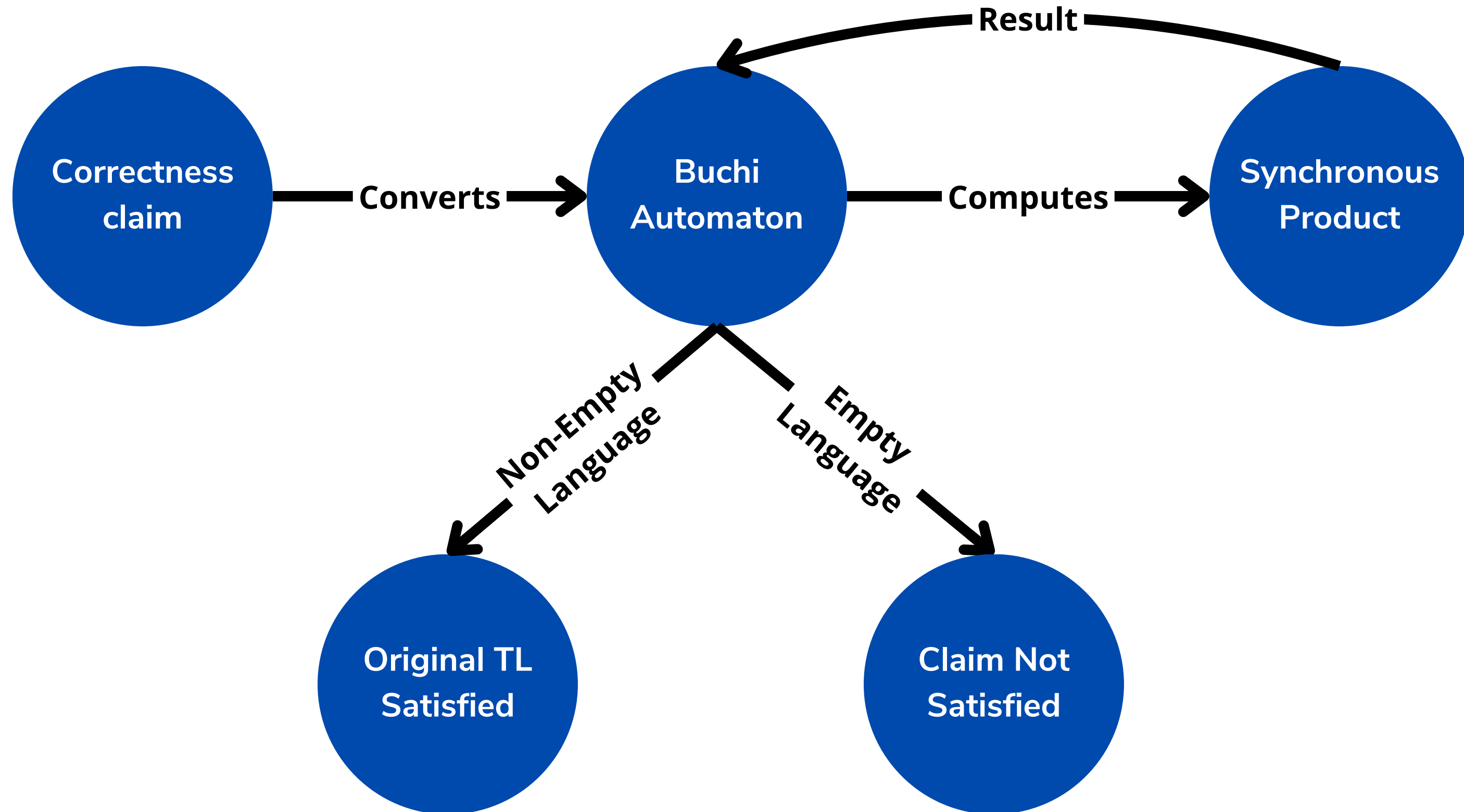


Fig. 1. The structure of SPIN simulation and verification.

02

Proposed Solution Verification Process



Foundation

Domain of Application

What SPIN design focus on?

Efficient verification of asynchronous software systems

This focus **affects** many central tool characteristics:

- specification language design
- logic
- verification procedure,
- reduction techniques,
- state encoding methods

Proposed Solution

Algorithms Requirements

- Spin uses cycles to **identify** violations in the correctness claims
- Cycle detection method **compatible** to all modes of verification including:
 - Exhaustive search
 - Bit-state hashing
 - Partial order reduction techniques
- Tarjan's is the best but doesn't fit, lowlink numbers not compatible to bit-state hashing.

Proposed Solution

Algorithms

- **Nested depth first search**
- Acceptance cycle exist:
 - Reachable from the initial state (1st run)
 - Reachable from itself (2nd run)
- $\text{Result} = \{\text{dfs1}\} + \{\text{dfs2}\}$
- May not detect all cycles but detect at least one cycle if any exists.

Proposed Solution

Algorithms

- Partial Order Reduction

What is this technique implemented for?

This technique makes verifying **more efficient** by reducing the number of system states it needs to check.

How does it work?

Only checks **one representative sequence** from each group of equivalent sequences. (sequences that lead to the same outcome in terms of the system's behavior).

Static reduction technique - identifies where it can safely apply these rules before verification starts.

Proposed Solution

Algorithms

- **Partial Order Reduction**

Online booking system where users can book tickets for events. Two main actions can occur independently: **User A books a ticket** or **User B books a ticket**.

Correctness property: The system **updates the number of available tickets after each booking**.

Proposed Solution

Algorithms

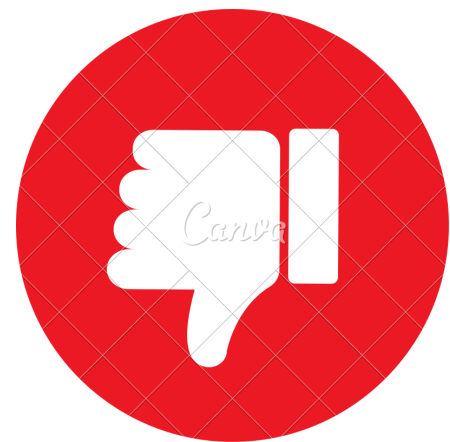
- **Partial Order Reduction**

Online booking system where users can book tickets for events. Two main actions can occur independently: **User A books a ticket** or **User B books a ticket**.

Correctness property: The system **updates the number of available tickets after each booking**.

Without partial order reduction, you would need to consider all possible orders of these actions, even if they don't affect each other:

- **Order 1:** User A books a ticket, then User B books a ticket.
- **Order 2:** User B books a ticket, then User A books a ticket.



Proposed Solution

Algorithms

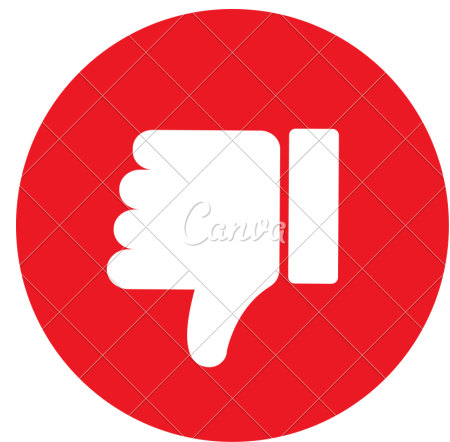
- **Partial Order Reduction**

Online booking system where users can book tickets for events. Two main actions can occur independently: **User A books a ticket** or **User B books a ticket**.

Correctness property: The system **updates the number of available tickets after each booking**.

Without partial order reduction, you would need to consider all possible orders of these actions, even if they don't affect each other:

- **Order 1:** User A books a ticket, then User B books a ticket.
- **Order 2:** User B books a ticket, then User A books a ticket.



With partial order reduction, only a representative order from the group of equivalent sequences is checked:

- **Order 1:** User A books a ticket, then User B books a ticket.



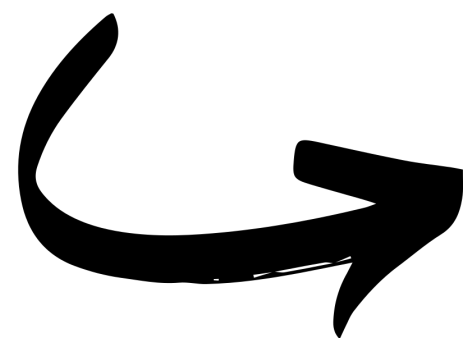
Proposed Solution

Algorithms

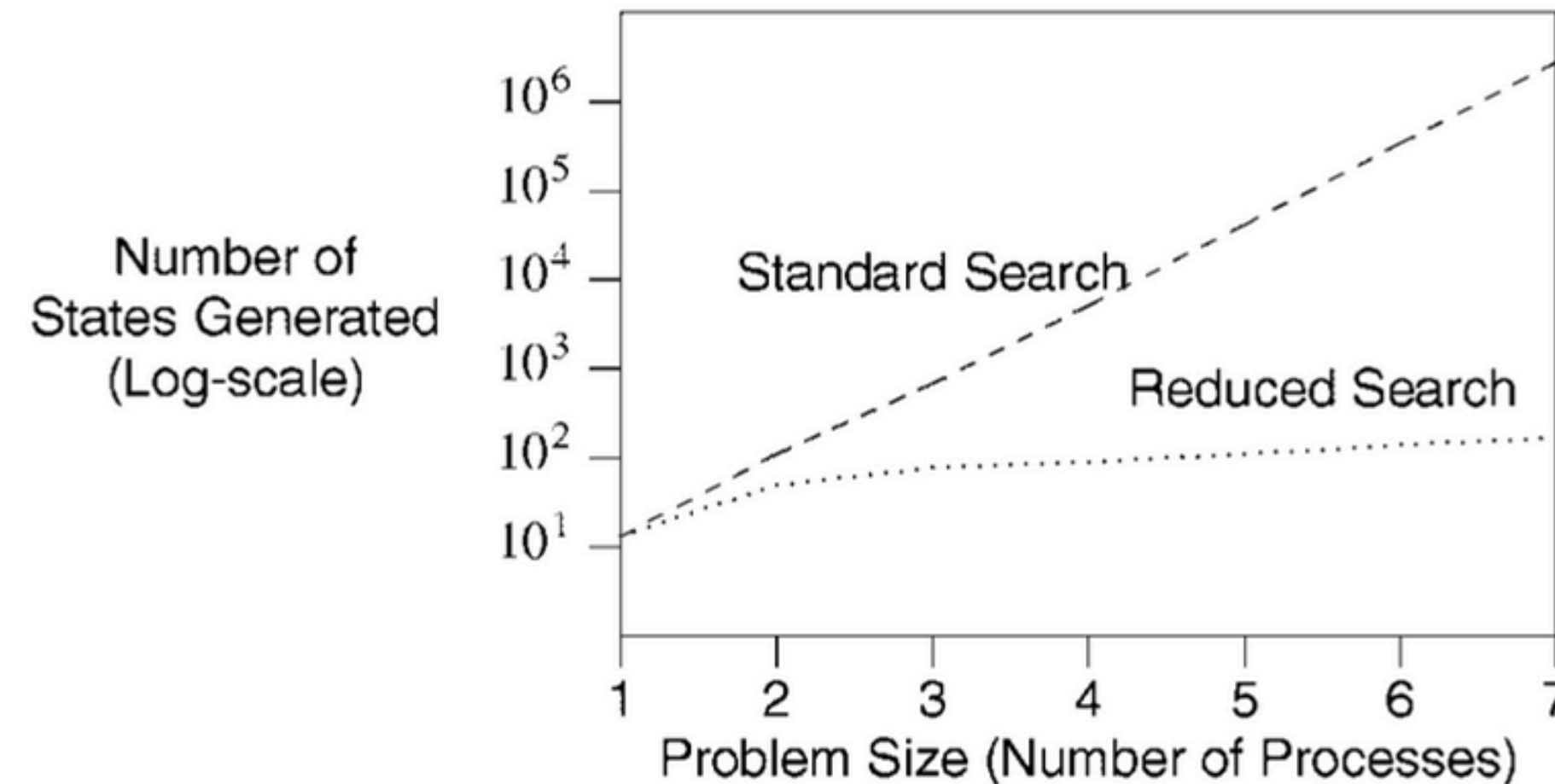
- Partial Order Reduction

Benefits from the method?

By ignoring redundant states, instead of the **number of states** growing exponentially with more processes, it **grows linearly**, which is much more manageable.



Saves between 10 to 90 percent of memory and runtime !!!



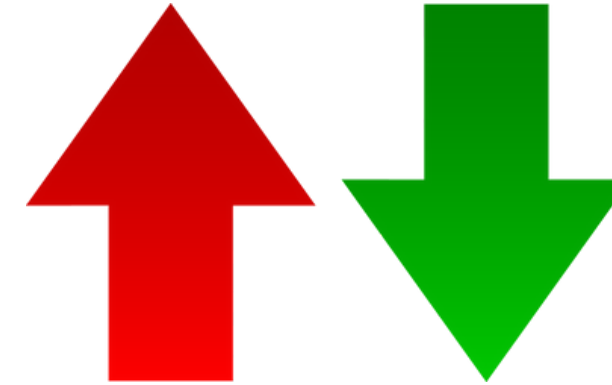
Proposed Solution

Algorithms

- Memory Management

What is the problem?

Run Time



Memory Usage

Memory is a **limited resource** in any system. Usually model checkers that uses small amount of memory take much longer to run.

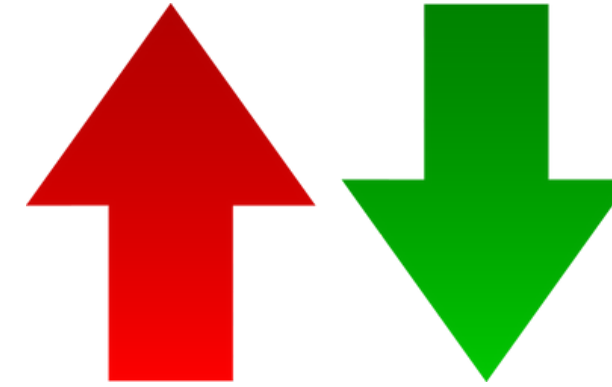
Proposed Solution

Algorithms

- Memory Management

What is the problem?

Run Time



Memory Usage

Memory is a **limited resource** in any system. Usually model checkers that use small amount of memory take much longer to run.

What are the proposed solutions?



- State Compression
- Bit-State Hashing

Proposed Solution

Algorithms

- State Compression

Context:

- In a PROMELA model, each process and communication channel typically has a small number of **unique states**.
- However, the **global state space**, which represents all possible combinations of these local states, can be very large.

What is the solution proposed?

Stores the local states only once, separately, and uses unique indices to refer to these states within the global state table

Proposed Solution

Algorithms

- State Compression - Example

Instead of storing the **complete concatenation of all local state** descriptors for the variables, the compression algorithm now stores each separable element alone, and uses **unique indices** to the local descriptors in the global state vector.

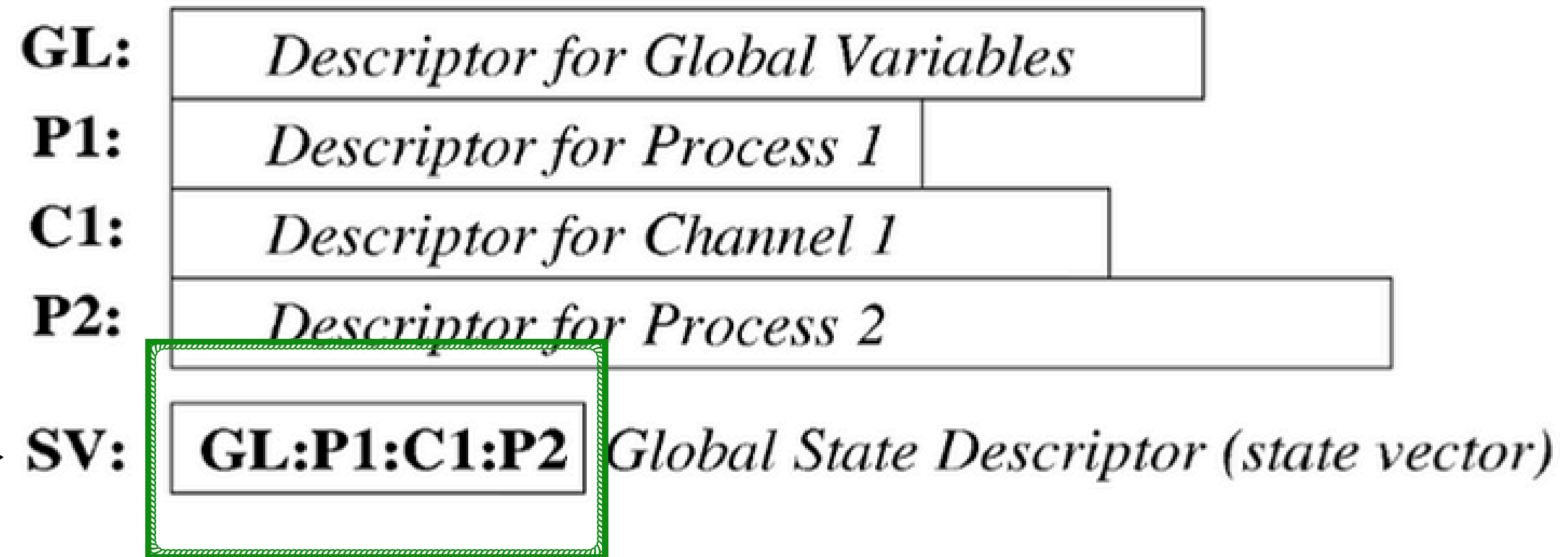


Fig. 6. State compression algorithm—indexing method.

Proposed Solution

Algorithms

- State Compression

Benefits from the method?

For 256 or fewer local states, we can use an 8-bit index (1 byte) to refer to these states.

TABLE 1
EFFECT OF COMPRESSION

Type of Run	No. States	Memory (Mb)	Time (sec.)
Standard	2,435,220	156.59	107.56
Compressed	2,435,220	59.57	123.46

The local states are stored only once but can be referred to many times, each referral using only **1 byte**. **With that memory usage can be reduced by 60 to 80%.**

Proposed Solution

Algorithms

- **Bit-State Hashing**

Stores each reachable state using only **two bits of memory**.

The bit-addresses are computed with **two statistically independent hash functions**.

Sequential Bit State Hashing: Multiple runs with statistically independent hashing functions can be performed until the required coverage level is reached, in case one single run doesn't cover all states.

Proposed Solution

Algorithms

- Bit-State Hashing vs **Exhaustive Searches**

Example:

Memory Size: 100 million bytes (M).

Memory Required to Store Each State: 1.000 bytes (S).

100.000 states can be stored.

Total number of states to check: 1 million states.

Coverage through exhaustive search: $100.000 / 1.000.000 = 10\%$

Proposed Solution

Algorithms

- Bit-State Hashing

Benefits from the method?

With only **1% of the memory** required for an exhaustive search, the bit-state hashing technique can realize a problem **coverage close to 100 percent**.

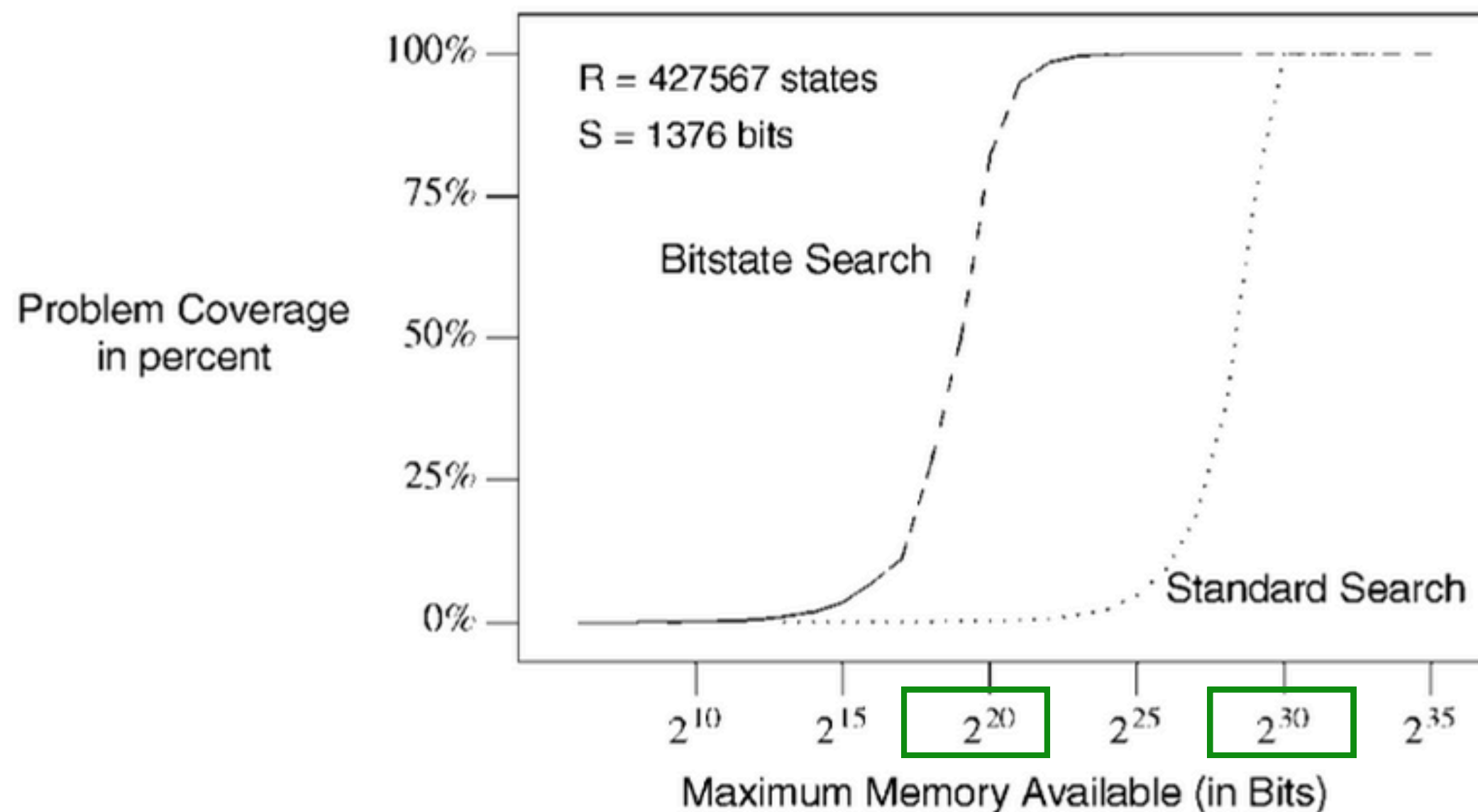


Fig. 7. Measured problem coverage [34], [42] effect of the optional bit-state hashing technique in SPIN.

03

Evaluation

Practical Applications

01 **Process Scheduling**

02 **Flow Control**

Evaluation

Practical Applications - Process Scheduling

- Process Scheduling

What is the process about and why does it need to be checked?

Scheduling process executions in a distributed operating system, which is **complex to solve correctly and efficiently**.

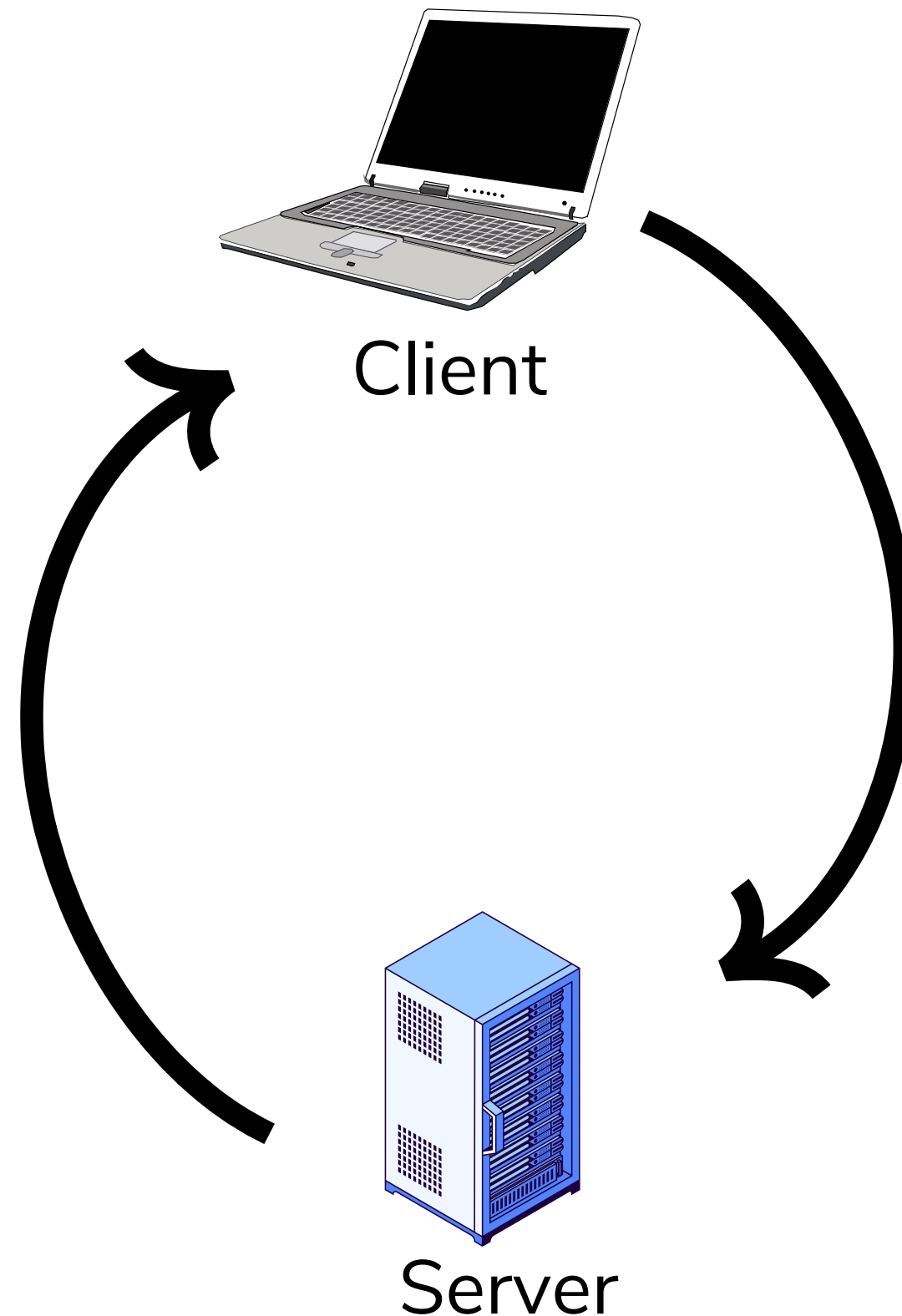
How does it work?

The application focuses on a **client-server interaction** where the client consumes resources provided by the server, and both processes use sleep and wakeup routines.

Evaluation

Practical Applications - Process Scheduling

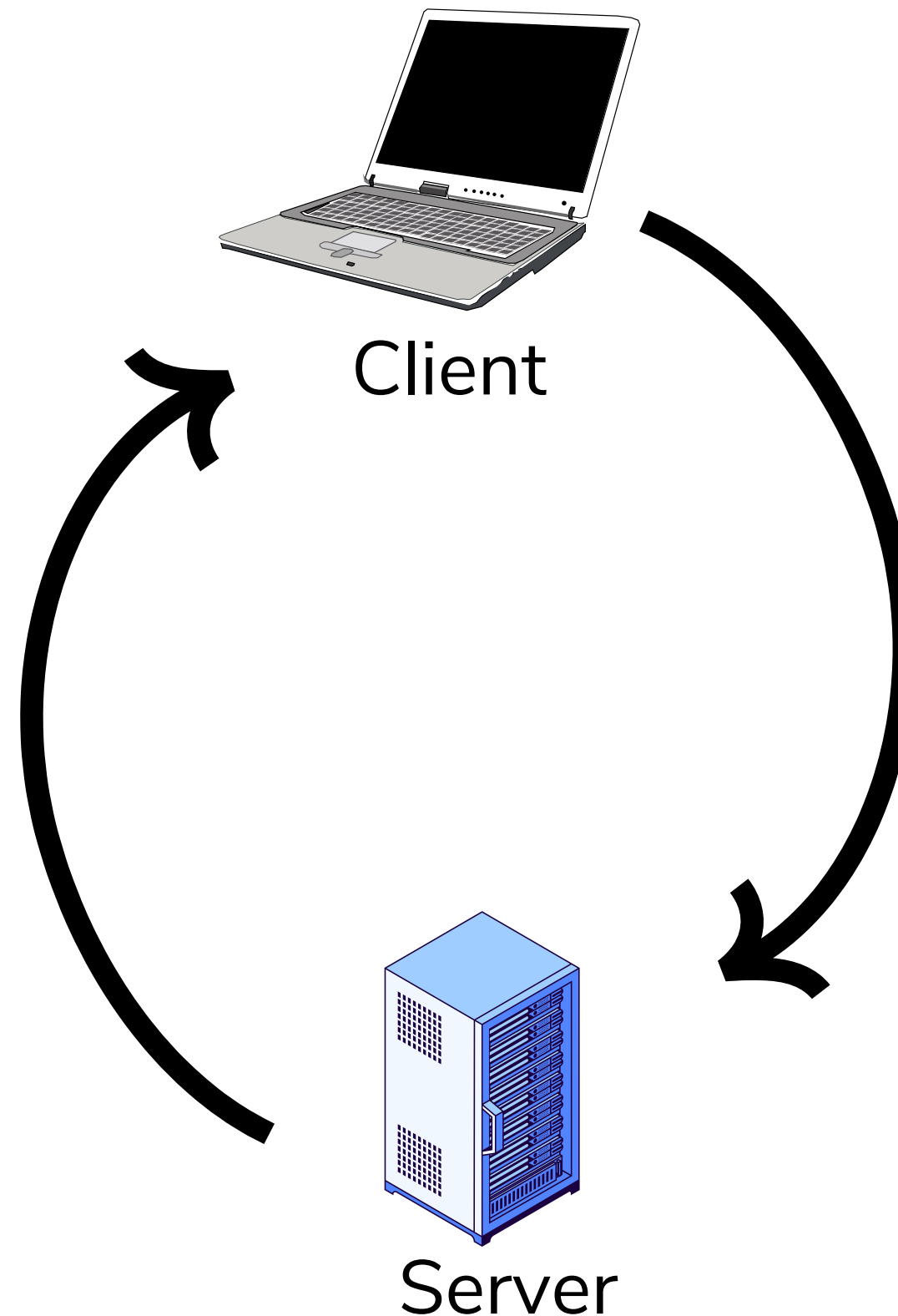
- Provides resources to client, one at a time.
- **Manages a variable `r_lock`**, that tells if a resource is available (0) or not (1).
- **Makes a resource available (0)** if it is not being used.
- **Reawakens the waiting client** when the desired resource is available.



Evaluation

Practical Applications - Process Scheduling

- Provides resources to client, one at a time.
- **Manages a variable `r_lock`**, that tells if a resource is available (0) or not (1).
- **Makes a resource available (0)** if it is not being used.
- **Reawakens the waiting client** when the desired resource is available.

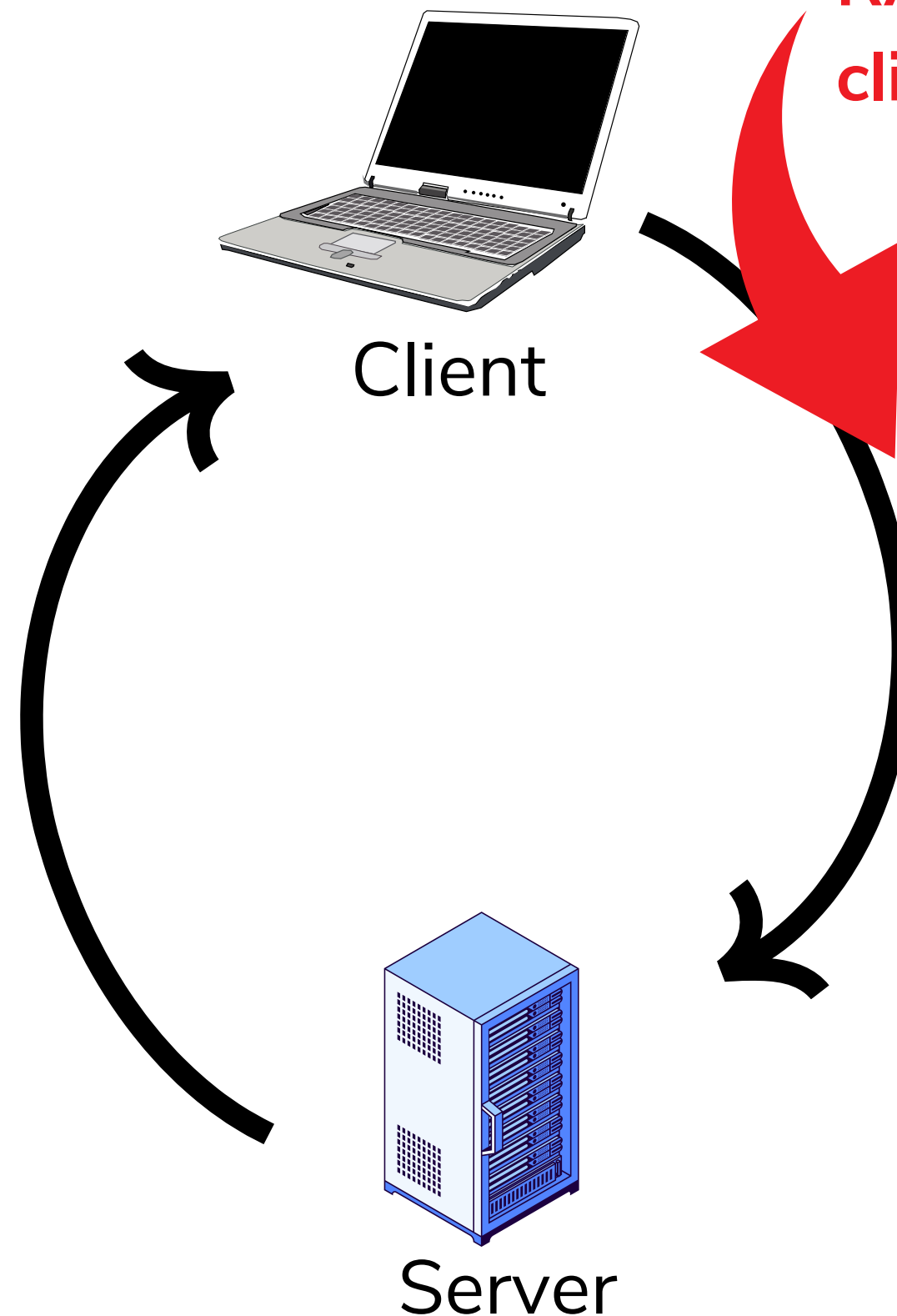


- Sets **`r_lock` to 1**, unavailable, when using a resource.
- Waits in **suspended mode** if resource isn't available, and **sets a flag** so the server knows about the waiting.
- Uses **spin lock** for exclusive access to code.

Evaluation

Practical Applications - Process Scheduling

- Provides resources to client, one at a time.
- **Manages a variable `r_lock`**, that tells if a resource is available (0) or not (1).
- **Makes a resource available (0)** if it is not being used.
- **Reawakens the waiting client** when the desired resource is available.



RACE CONDITION: Suspended client never reawakened.

- Sets **`r_lock`** to **1**, unavailable, when using a resource.

Waits in **suspended mode** if resource isn't available, and **sets a flag** so the server knows about the waiting.

- Uses **spin lock** for exclusive access to code.

Practical Applications - Process Scheduling

- Process Scheduling - SPIN verification

Critical Lock



After waking up, the client process can execute the code between lines 12 and 15 without having first obtained the critical lock from line 10, it can set **r_want** again and go back to sleep, and never break the loop!

```

7  active proctype client()
8  {
9  sleep:                                /* the sleep routine */
10     atomic { (lk == 0) -> lk = 1};    /* SPINlock(&lk) */
11     do                                /* while r.lock is set */
12         :: (r_lock == 1) ->          /* r.lock == 1 */
13             r_want = 1;              /* set the want flag */
14             State = Wakeme;          /* remember State */
15             lk = 0;                  /* frelock(&lk) */
16             (State == Running);      /* wait for wakeup */
17         :: else ->                   /* r.lock == 0 */
18             break                    /* break from do-loop */
19     od;
20 progress:                             /* progress label */
21     assert(r_lock == 0);              /* should still be true */
22     r_lock = 1;                       /* consumed resource */
23     lk = 0;                           /* frelock(&lk) */
24     goto sleep
25 }
```

Evaluation

Practical Applications - Process Scheduling

• Process Scheduling - SPIN verification

SPIN generates an **execution scenario** showing that a process can remain indefinitely suspended.

The problem happens when the client is reawakened by the server and checks resource availability without reclaiming the lock first.

Client Process	Server Process	Line No.	Local States	Comment
[r_lock = 1]		22	16,15	Consume Resource
[lk = 0]		23	3,15	Release the lock
[lk==0]		10	2,15	Test the lock
[lk = 1]		10	11,15	Set the lock
[r_lock==1]		12	5,15	Resource not available
[r_want = 1]		13	6,15	Set want flag
	[r_want==1]	33	6,6	Server checks flag
	[sleep_q==0]	35	6,5	Wait lock on sleep_q
	[sleep_q = 1]	35	6,7	Set it
	[r_want = 0]	37	6,11	Resets the flag
	[else]	44	6,13	No process sleeping!
	[sleep_q = 0]	46	6,1	Release the lock
	[r_lock = 0]	30	6,2	Provide new resource
[State = Wakeme]		14	7,2	Client goes to sleep!
[lk = 0]		15	8,2	Releases the lock
	[lk==0]	31	8,15	Server checks the lock
	[else]	47	8,1	No flag is set
Non-Progress Cycle:				
	[r_lock = 0]	30	8,2	The server repeats this forever, while the client remains suspended.
	[lk==0]	31	8,15	
	[else]	47	8,1	

Fig. 9. An error scenario generated by SPIN (annotated).

Evaluation

Practical Applications - Process Scheduling

- **Process Scheduling - SPIN results**
- SPIN's SPIN shows that the client can skip locking and get stuck. The problem is subtle enough to challenge experienced designers but can be solved quickly with SPIN.
- The verification task involves no more than 300 reachable states and runs efficiently, taking **less than 0.1 CPU seconds** on an average workstation.
- **Proposed solution:** Adding a jump to ensure the client reclaims the lock every time it is reawakened.

Practical Applications - Flow Control

- Standard go-back-n flow control protocol
 - technique for reliable data transmission over an unreliable network connection.
- Steps to Model Flow Control with SPIN:
 - Create a verification model
 - Translation into PROMELA
 - Check correctness with SPIN
 - Simulation
 - Debugging and Testing
 - print statements and inline assertions to help understand and fix protocol issues.
- **Goal:** verify the safety and liveness properties of this protocol with SPIN.

Evaluation

Flow Control - Safety

- We can perform an exhaustive verification run with SPIN to prove some basic safety properties:
 - absence of deadlock, unreachable code, unspecified receptions, etc.
- This involves generating and compiling code for the specific protocol model, and then running the model checker with various reduction techniques and memory management options enabled.

TABLE 3
EFFECT OF COMPLEXITY CONTROL MEASURES (MAXSEQ = 3)

No. States	Memory (Mb)	Time (sec.)	Compression	Partial Order	Coarsening
402,997	26.7	25.8			
402,997	13.8	23.7	+		
182,735	12.5	6.5		+	
16,441	2.5	1.4			+
14,645	2.0	1.2	+	+	+


Evaluation

Flow Control - Liveness

- In contrast to safety properties, liveness is concerned with ensuring that certain desirable events will **eventually** occur during the execution of the system.
 - In a communication protocol, a liveness property might be that every message sent will eventually be received.
- The verification model is extended to include **two** new environment processes that simulate the ongoing operation of the system.
 - Helps in ensuring that messages are neither lost nor received out of order.

Evaluation

Flow Control - Liveness

- A practical approach to proving liveness properties involves using Linear Temporal Logic (LTL) formulae to express correctness requirements.
- The input data sequence consists of one red and one blue message inserted randomly in an infinite sequence of white messages.
 - ...  ...
- The liveness properties are:
 - No message loss: $! [] (sr \rightarrow < > rr)$
 - every sent red message (sr) should eventually be received (rr).
 - No out-of-order delivery: $(! rr \text{ U } rb)$
 - a red message should not be received (rr) after a blue message (rb).
- These properties can be translated into Büchi automata for verification purposes.

01

Articles about **logic model checking techniques**, from the pioneers of the technology, that laid the foundation for a new generation of model checking tools.

02

Studies on other **well-known approaches to model checking**.

03

Studies on validation of systems that review the SPIN model checker as a verification system for asynchronous processes.

- [12] E.M. Clarke, E.A. and Emerson, "Synthesis of Synchronization Skeletons for Branching Time Temporal Logic," *Proc. Logic of Programs: Workshop*, Yorktown Heights, N.Y., *Lecture Notes in Computer Science 131*. Springer-Verlag, May 1981.
- [61] J.P. Queille and J. Sifakis, "Specification and Verification of Concurrent Systems in Cesar," *Proc. Fifth Int'l. Symp. Programming*, pp. 195-220, *Lecture Notes In Computer Science 137*. Springer-Verlag, 1981.
- [12] E.M. Clarke, E.A. and Emerson, "Synthesis of Synchronization Skeletons for Branching Time Temporal Logic," *Proc. Logic of Programs: Workshop*, Yorktown Heights, N.Y., *Lecture Notes in Computer Science 131*. Springer-Verlag, May 1981.
- [49] R.P. Kurshan, *Computer-Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.
- [53] K.L. McMillan, *Symbolic Model Checking*. Boston: Kluwer Academic, 1993.
- [36] G.J. Holzmann, *Design and Validation of Computer Protocols*. Englewood Cliffs, N.J.: Prentice Hall, 1991.
- [38] G.J. Holzmann, P. Godefroid, and D. Pirotin, "Coverage Preserving Reduction Strategies for Reachability Analysis," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV92)*, pp. 349-364, Orlando, Fla., North-Holland, June 1992.

04

Vardi and Wolper extended the original logic model checking techniques with an automata theoretic model that became the formal basis for temporal logic model checking in the SPIN system.

05

Article about the partial order reduction method that SPIN uses, discussed in this presentation.

06

Some examples of successful implementation of the bit-state hashing technique in large-scale industrial applications of formal verification.

[77] M.Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Automatic Program Verification," *Proc. First IEEE Symp. Logic in Computer Science*, pp. 322-331, 1986.

[57] D. Peled, "Combining Partial Order Reductions with On-The-Fly Model-Checking," *Proc. Sixth Int'l Conf. Computer Aided Verification (CAV94)*, pp. 377-390, Stanford, Calif., *Lecture Notes In Computer Science 818*. Springer-Verlag, 1994.

[11] J. Chaves, "Formal Methods at AT&T, An Industrial Usage Report," *Proc. Fourth FORTE Conf. Formal Description Techniques*, pp. 83-90, Sydney, Australia, 1991.

[40] G.J. Holzmann, "The Theory and Practice of a Formal Method: NewCoRe," *Proc. 13th IFIP World Computer Congress*, pp. 35-44, Hamburg, Germany, North-Holland, Aug. 1994.

Conclusion

- Early Development of concurrent systems in the time of this paper
- The **design methodology** that is supported by SPIN can be summarized as follows:
 - Distinction between behavior and requirements on behavior
 - The prototype is verified using the model checker SPIN
 - The design is revised until its critical correctness properties can successfully be proven
- Future outlook of the paper
 - The increasing number of applications of SPIN, and the growing acceptance of formal methods in general
 - Hopeful signs that this paradigm of design is maturing

05

Conclusion

SPIN nowadays

SPIN 2024

30th International Symposium on Model Checking Software

10-11 April 2024, co-located with **ETAPS 2024** in Luxembourg City, Luxembourg

The 30th International SPIN symposium on Model Checking of Software (SPIN 2024) will be held in Luxembourg on 10 and 11 April 2023. SPIN 2024 is the latest in a successful series of workshops and symposia for practitioners and researchers interested in symbolic and state space-based techniques for the validation and analysis of software systems. Techniques and empirical evaluations based on explicit representations of state spaces, as implemented in the SPIN model checker or other tools, or techniques based on the combination of explicit representations with other representations, are the focus of this symposium.

SPIN 2024 will be colocated with the 27th European Joint Conferences on Theory and Practice of Software (ETAPS 2024).

Published March 15, 2024 | Version SPIN24-proceedings Software Open

Reproduction Package for SPIN 2024 Article 'Verification Witnesses Version 2'

Ayaziová, Paulina¹ ; Beyer, Dirk² ; Lingsch-Rosenfeld, Marian² ; Spiessl, Martin² ; Strejček, Jan¹ Show affiliations

Abstract

This artifact is a reproduction package for the paper Software Verification Witnesses 2.0 which has been accepted at SPIN 2024.

It consists of all executables, input data and results required to reproduce the experiments and see the raw data from which the results presented in the paper were extracted. Specifically, it contains the tools Symbiotic, CPAchecker, UAutomizer, witness-lint and the input data sv-benchmarks as used in SV-COMP24.

The artifact is based on the SoSy-Lab Virtual Machine (Ubuntu 22.04 LTS) and has been configured such that no extra configuration is necessary and the artifact can run inside it without any problems.

By default, experiments are run with 2 cores and 15 GB of memory for at most 900 seconds. A full reproduction of the experiments requires roughly **8 months** of CPU time. For demonstration purposes, a subset of tasks has been selected which should take at most 1 day to complete and will likely finish sooner. To test that everything is working as intended, all experiments can be run for two files which should finish all the experiments in around 30 minutes.

Published January 27, 2024 | Version v1 Software Open

Replication Package for Paper "Test-Case Generation with Automata-based Software Model Checking" SPIN 24

Jakobs, Marie-Christine¹ ; Barth, Max² Show affiliations

Replication package for the paper:
"Test-Case Generation with Automata-based Software Model Checking" by Max Barth and Marie-Christine Jakobs.

Synchronisation in Language-level Symmetry Reduction for Probabilistic Model Checking

Valkov, I., Donaldson, A. and Miller, A. (2024) Synchronisation in Language-level Symmetry Reduction for Probabilistic Model Checking. In: 30th International SPIN symposium on Model Checking of Software (SPIN 2024), Luxembourg, 10-11 Apr 2024, (Accepted for Publication)



Text

321708.pdf - Accepted Version

Restricted to Repository staff only

617kB

Bibliography

REFERENCES

- [1] A. Agarwal, "A Unified Approach to Fault-Tolerance in Communication Protocols, Based on Recovery Procedures," PhD thesis, Computer Science Dept., Concordia Univ., Montreal, Canada, 1995.
- [2] M. Aljoudi, "On the Application of an Automated Validation Tool to Realistic Protocols," MSc thesis, INRS-Telecommunications, Univ. de Quebec, Canada, Aug. 1994.
- [3] F.R. D'Argenio, J.P. Katoen, T. Reys, and J. Trellmann, "Modeling and Verifying a Bounded Retransmission Protocol," *Proc. COST 247 Int'l Workshop Applied Formal Methods in System Design*, Maribor, Slovenia, June 1994.
- [4] A. Bass, M. Hayden, G. Morrisett, and T. von Eicken, "A Language-Based Approach to Protocol Construction," *Proc. ACM SIGPLAN Workshop Domain Specific Languages (DSL)*, Paris, Jan. 1997.
- [5] R. Bhawaji and C. Hameyer, "Verifying SCR Requirements Specifications Using State Exploration," *Proc. First ACM SIGPLAN Workshop Automatic Analysis of Software*, R. Chavevaland and D. Jackson, eds., pp. 9-24, Paris, Jan. 1997.
- [6] B. Boute and P. Godefroid, "Model Checking in Practice: An Analysis of the ACCESS Bus Protocol Using SPIN," *Proc. Formal Methods Europe (FME96)*, Oxford, England, *Lecture Notes in Computer Science 1151*, pp. 445-478, Springer-Verlag, Mar. 1996.
- [7] L. Bouvin, "Design of Validation Models in PROMELA for the Medium Access Protocol of the FTM Project," Report Royal Inst. of Technology, Stockholm, Sweden, Aug. 1991.
- [8] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic Model Checking: 10 States and Beyond," *Informatic Computing*, vol. 98, no. 2, pp. 142-170, June 1992.
- [9] T. Canet, "Modeling and Verification of a Multiprocessor RealTime OS Kernel," *Proc. Seventh Int'l Conf. Formal Description Techniques*, pp. 35-50, Bern, Switzerland, Oct. 1994.
- [10] T. Canet, "Using Concurrency and Formal Methods for the Design of Safe Process Control," *Proc. FDS/PCSENI Workshop*, Berlin, Mar. 1994.
- [11] J. Chaves, "Formal Methods at AT&T: An Industrial Usage Report," *Proc. Fourth FORTE Conf. Formal Description Techniques*, pp. 83-90, Sydney, Australia, 1991.
- [12] E.M. Clarke, E.A. Emerson, "Synthesis of Synchronization Skeletons for Branching Time Temporal Logic," *Proc. Logic of Programs Workshop*, Yorktown Heights, N.Y., *Lecture Notes in Computer Science 131*, Springer-Verlag, May 1991.
- [13] C.-E. Chou, and D. Peled, "Verifying a Model-Checking Algorithm," *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS96)*, Passau, Germany, *Lecture Notes in Computer Science 1155*, pp. 241-257, Springer-Verlag, Mar. 1996.
- [14] Y. Chevada, "Theories of Automata on Mega-Tapes: A Simplified Approach," *J. Computer and System Science*, vol. 8, pp. 117-141, 1974.
- [15] A. Cimatti, A. Giunchiglia, G. Mongardi, D. Romano, F. Torrelli, and P. Traverso, "Model Checking Safety Critical Software with SPIN: An Application to a Railway Interlocking System," *Proc. Third SPIN Workshop*, R. Langerak, ed., Twente Univ., The Netherlands, Apr. 1997.
- [16] J.C. Corbett, "Evaluating Deadlock Detection Methods for Concurrent Software," *IEEE Trans. Software Eng.*, vol. 22, no. 3, pp. 161-180, Mar. 1996.
- [17] C. Courcoubetis, M.Y. Vardi, P. Wolper, M. Yannakakis, "Memory Efficient Algorithms for the Verification of Temporal Properties," *Formal Methods in Systems Design*, vol. 1, pp. 275-288, 1992.
- [18] D. Dolev, M. Klawns, and M. Radoch, "An On (log n) Unidirectional Distributed Algorithm for Extrema Finding in a Circle," *J. Algorithms*, vol. 3, pp. 245-260, 1982.
- [19] G. Duval and J. Julliard, "Modeling and Verification of the RUBUS Micro-Kernel with SPIN," *Proc. First SPIN Workshop*, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.
- [20] P. Van Eijck, "Verifying Relay Circuits Using State Machines," *Proc. Third SPIN Workshop*, R. Langerak, ed., Twente Univ., The Netherlands, Apr. 1997.
- [21] H. Endegren, "Verifying Semantic Relations in SPIN," *Proc. First SPIN Workshop*, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.
- [22] M.J. Ferguson, "Validation of the Radio Link Protocol," Data Services Task Group of ANSI Accredited TIA TR45-3, Contribution TR45.3.2.5/93.08.25.02, Sept. 1993.
- [23] M.J. Ferguson, "Formalization and Validation of the Radio Link Protocol RLPL," *Computer Networks and ISDN Systems*, to appear, 1997.
- [24] F. Gagnon, "Bouclier, un validateur pour la language de specification Ganton," PhD thesis, Univ. de Quebec, Canada, July 1995.
- [25] R. Gerb, D. Peled, M.Y. Vardi, and P. Wolper, "Simple On-The-Fly Automatic Verification of Linear Temporal Logic," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV95)*, pp. 3-18, Warsaw, Poland, Chapman & Hall, June 1995.
- [26] P. Godefroid, and G.J. Holzmann, "On the Verification of Temporal Properties," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV95)*, pp. 109-124, Liege, Belgium, North-Holland, June 1995.
- [27] P. Godefroid, "Partial Order Methods for the Verification of Concurrent Systems," *Lecture Notes in Computer Science 1102*, Springer-Verlag, 1996.
- [28] P. Godefroid, "Symbolic Protocol Verification with Queue BDDs," *Proc. Logic in Computer Science*, pp. 198-208, Rutgers Univ., New Brunswick, N.J., July 1996.
- [29] J.-C. Gregoire, "State Space Compression in SPIN with GETSs," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.
- [30] M. Griffioen, "Specification and Verification of a Wireless LAN Controller Chip Using PROMELA and SPIN," Technical Report, AT&T Network Wireless Systems, The Netherlands, 1996.
- [31] J. Hajek, "Automatically Verified Data Transfer Protocols," *Proc. Fourth ICCP*, pp. 749-756, Kyoto, Aug. 1978.
- [32] G.J. Holzmann, "PAN: A Protocol Specification Analyser," Technical Report TMR1-11271-3, AT&T Bell Laboratories, Mar. 1981.
- [33] G.J. Holzmann, "Tracing Protocols," *AT&T Technical J.*, vol. 64, pp. 2,413-2,434, Dec. 1983.
- [34] G.J. Holzmann, "An Improved Protocol Reachability Analysis Technique," *Software, Practice and Experience*, vol. 18, no. 2, pp. 137-161, Feb. 1988.
- [35] G.J. Holzmann, "Algorithms for Automated Protocol Verification," *AT&T Technical J.*, vol. 69, no. 1, pp. 32-44, Jan. 1990.
- [36] G.J. Holzmann, *Design and Validation of Computer Protocols*, Englewood Cliffs, N.J.: Prentice Hall, 1991.
- [37] G.J. Holzmann, "Protocol Design: Redefining The State of the Art," *IEEE Software*, pp. 17-22, Jan. 1992.
- [38] G.J. Holzmann, P. Godefroid, and D. Pinotin, "Coverage Preserving Reduction Strategies for Reachability Analysis," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV95)*, pp. 349-364, Orlando, Fla., North-Holland, June 1995.
- [39] G.J. Holzmann, "Design and Validation of Protocols: A Tutorial," *Computer Networks and ISDN Systems*, vol. 25, no. 9, pp. 981-1,017, 1993.
- [40] G.J. Holzmann, "The Theory and Practice of a Formal Method: NewCoRe," *Proc. 13th IFIP World Computer Congress*, pp. 33-44, Hamburg, Germany, North-Holland, Aug. 1994.
- [41] G.J. Holzmann and D. Peled, "An Improvement in Formal Verification," *Proc. Seventh FORTE Conf. Formal Description Techniques*, pp. 177-194, Bern, Switzerland, Oct. 1994.
- [42] G.J. Holzmann, "An Analysis of Bit-State Hashing," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV95)*, pp. 301-314, Warsaw, Poland, Chapman & Hall, June 1995.
- [43] G.J. Holzmann, D. Peled, and M. Yannakakis, "On Nested DepthFirst Search," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.
- [44] G.J. Holzmann, "Designing Bug-Free Protocols with SPIN," *The Computer Comm. J.*, to appear 1997.
- [45] G.J. Holzmann, "State Compression in SPIN: Recursive Indexing and Compression Training Runs," *Proc. Third SPIN Workshop*, Twente Univ., R. Langerak, ed., The Netherlands, Apr. 1997.
- [46] H.E. Jensen, K. Larsen, and A. Skov, "Modeling and Analysis of a Collision Avoidance Protocol Using SPIN and UPPAAL," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.
- [47] A. Jossang, "Security Protocol Verification Using SPIN," *Proc. First SPIN Workshop*, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.
- [48] P. Kars, "The Application of PROMELA and SPIN in the BOS Project," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.
- [49] R.P. Kurshan, *Computer-Aided Verification of Coordinating Processes*, Princeton Univ. Press, 1994.
- [50] F.J. Lin, "Two Applications of PROMELA/ SPIN," *Proc. First SPIN Workshop*, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.
- [51] S. Loeffler and A. Serbroucht, "Protocol Design: From Specification to Implementation," *Proc. Fifth Open Workshop for High Speed Networks*, Paris, Mar. 1996.
- [52] "IEEE Std. 802-2-1985, ISO DMS 802/ 2," IEEE Standards for Local Area Networks: Logical Link Control, Published by the IEEE Standards Board, 345 E. 47th Street, New York, NY 10017, USA, 111 pp., ISBN 471-82748-7, 1984. Revised as 802-2-1989 in Aug. 1989.
- [53] K.L. McMillan, *Symbolic Model Checking*, Boston: Kluwer Academic, 1993.
- [54] E. Najm and F. Olsen, "Reactive EFSMs, Reactive PROMELA/ RSPIN," *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS96)*, pp. 349-368, Passau, Germany, *Lecture Notes in Computer Science 1155*, Springer-Verlag, Mar. 1996.
- [55] T. Nakatani, "Verification of a Group Address Registration Protocol using PROMELA and SPIN," *Proc. Third SPIN Workshop*, R. Langerak, ed., Twente Univ., The Netherlands, Apr. 1997.
- [56] R.M. Needham and A.J. Herbert, *The Cambridge Distributed Computing System*, London: Addison-Wesley, 1982.
- [57] D. Peled, "Combining Partial Order Reductions with On-The-Fly Model-Checking," *Proc. Sixth Int'l Conf. Computer Aided Verification (CAV96)*, pp. 377-390, Stanford, Calif., *Lecture Notes in Computer Science 818*, Springer-Verlag, 1994.
- [58] D. Peled, "On Projective and Separable Properties," *Colloquium on Decis in Algebra and Programming*, pp. 291-307, Edinburgh, Scotland, *Lecture Notes in Computer Science 787*, Springer-Verlag, 1994.
- [59] R. Pike, D. Presotto, K. Thompson, and G.J. Holzmann, "Process Sleep and Wakeup on a Shared-Memory Multiprocessor," *Proc. the Spring: EuroOpen Conf.*, pp. 141-146, Tromsø, Norway, 1991.
- [60] A. Pnueli, "The Temporal Logic of Programs," *Proc. 18th IEEE Symp. Foundations of Computer Science*, Providence, R.I., pp. 46-57, 1977.
- [61] J.P. Quille and J. Sifakis, "Specification and Verification of Concurrent Systems in Cesar," *Proc. Fifth Int'l. Symp. Programming*, pp. 195-220, *Lecture Notes in Computer Science 117*, Springer-Verlag, 1981.
- [62] B. Rabarido, "SPIN as a Hardware Design Tool," *Proc. First SPIN Workshop*, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.
- [63] M. Raynal, *Distributed Algorithms and Protocols*, New York: John Wiley & Sons, 1992.
- [64] L.M. Rasm, "Process Synchronization in the UTS Kernel," *Computing Systems, Proc. Usenix Conf.*, vol. 3, no. 3, pp. 387-421, 1990.
- [65] T.C. Reys and R. Langerak, "Validation of Bosch's Mobile Communication Network Architecture with SPIN," *Proc. Third SPIN Workshop*, R. Langerak, ed., Twente Univ., The Netherlands, Apr. 1997.
- [66] T.S. Chan and I. Gorton, "Formal Validation of a High Performance Error Control Protocol Using SPIN," *Software, Practice and Experience*, vol. 26, no. 1, pp. 105-124, Jan. 1996.
- [67] S. Shukla, D.J. Rosenkrantz, and S.S. Ravi, "Simulation and Validation of Self-Stabilizing Protocols," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.
- [68] *Proc. First SPIN Workshop*, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.
- [69] *Proc. Second SPIN Workshop*, J.-Ch. Gregoire, G.J. Holzmann, and D. Peled, eds., Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.
- [70] *Proc. Third SPIN Workshop*, R. Langerak, ed., Twente Univ., The Netherlands, Apr. 1997.
- [71] M. Stankauskas, "Tales from the Front: Industrial Experience with Formal Validation," *Proc. First SPIN Workshop*, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.
- [72] W.T. Strayer, B.J. Dempsey, and A.C. Weaver, *ATP--The Xpress Transfer Protocol*, Reading, Mass.: Addison-Wesley, 1992.
- [73] A. Tanenbaum, *Computer Networks*, second edition, Englewood Cliffs, N.J.: Prentice Hall, 1989.
- [74] R.E. Tarjan, "Depth First Search and Linear Graph Algorithms," *SIAM J. Computing*, vol. 1, no. 2, pp. 146-160, 1972.
- [75] W. Thomas, "Automata on Infinite Objects," *Handbook of Theoretical Computer Science*, J. Van Leeuwen, ed., pp. 133-187, Elsevier Science, 1990.
- [76] S. Tripakis and C. Courcoubetis, "Extending PROMELA and SPIN for real-time," *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS96)*, pp. 329-348, Passau, Germany, *Lecture Notes in Computer Science 1155*, Springer-Verlag, Mar. 1996.
- [77] P. Tullmann, J. Turner, J. McCompsdale, J. Lepreau, A. Chmura, and G. Back, "Formal Methods: A Practical Tool for OS Implementers," *Proc. HaeOS-92, Sixth IEEE Workshop Hot Topics in Operating Systems*, Cape Cod, Mass., May 1997.
- [78] R. Nishimura and G. Gopalakrishnan, "Explicit Enumeration Based on Verification Made Memory Efficient," *Proc. Conf. Computer Hardware Description Languages (CHDL'95)*, pp. 417-422, Chiba, Japan, 1995.
- [79] M.Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Automatic Program Verification," *Proc. First IEEE Symp. Logic in Computer Science*, pp. 322-331, 1986.
- [80] M.Y. Vardi and P. Wolper, "Reasoning About Infinite Computations," *Information and Computation*, vol. 113, pp. 1-37, 1994, appeared as a conference paper in 1983.
- [81] W. Vlasar, "Memory Efficient Storage in SPINs," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.
- [82] A.S. Wirth, J.W. O'Leary, and G.M. Brown, "Codeligns of Communication Protocols," *Computer*, vol. 26, no. 12, pp. 46-52, Dec. 1993.
- [83] C.H. Wiest, "General Technique for Communications Protocol Validation," *IBM J. Research and Development*, vol. 22, no. 3, pp. 393-404, 1978.
- [84] P. Wolper, "Expressing Interesting Properties of Programs in Propositional Temporal Logic," *Proc. 13th ACM Symp. Principles of Programming Languages*, pp. 140-153, St. Petersburg, Fla., Jan. 1986.



Gerard J. Holzmann received an MSc degree in electrical engineering in 1978 and a PhD degree in technical sciences in 1979 from the University of Technology in Delft, The Netherlands. He joined the Computing Sciences Research Center at Bell Laboratories, Murray Hill, New Jersey, in 1980, where today he is a distinguished member of technical staff (DMTS) in the Computing Principles Research Department. In 1980, Holzmann wrote one of the first automated protocol verification systems, called *Pro*. Since then, he has written several other on-the-fly verification systems for a variety of applications. His latest system, *Spin*, is considered to be one of the most efficient and most widely used LTL model checking systems. Dr. Holzmann has written books on digital image editing, on the history of communications, and on protocol verification. He serves as an editor for the journal, *Formal Methods in Systems Design (FOUwer)*.



Instituto Superior Técnico

Thank You