Scalable Verification-Aware Kernel P Systems: Indexed Modelling, Candidate Generation, and Trace-Level Verification

Doru Constantin[3], Cristina Tudose[3], Ștefan Iacob[1], Ionuț Mihai Niculescu[3], Florentin Ipate[1], and Marian Gheorghe[2]

[1] University of Bucharest, Faculty of Mathematics and Computer Science, 14 Academiei St., 010014 Bucharest, Romania
f.ipate@fmi.unibuc.ro, stefan.iacob@fmi.unibuc.ro
[2] University of Bradford, School of Computing and Engineering, West Yorkshire, Bradford BD7 1DP, UK
m.gheorghe@bradford.ac.uk
[3] National University of Science and Technology POLITEHNICA Bucharest, Department of Mathematics-Informatics, Pitești University Centre, 110040, Romania
doru.constantin0804@upb.ro, cristina.tudose0203@upb.ro, ionutmihainiculescu@gmail.com

Abstract. Kernel P systems provide an executable formalism for parallel rule-based computation with simulation and verification support. We present a verification-aware benchmark methodology combining source-level indexation and refinement, direct kPWorkbench simulation, external candidate generation, execution-derived representative selection, and temporal checking. The methodology is evaluated on Broadcasting, Partition/Subset Sum, and Binary Adder, covering communication, combinatorial branching, and bit-level arithmetic. Refined and indexed sources make completion, pruning, cleanup, and carry propagation more explicit and therefore more directly accessible to property-oriented analysis. At workflow level, generated candidates are materialized and executed in kPWorkbench before representative traces are selected for NuSMV checking. Across the 18 selected representatives, the campaign comprises 1203 real executions and 282 NuSMV properties, all of which hold. A complementary Subset Sum control audit further shows that selected and lower-ranked candidates can be distinguished by non-vacuous target-reachability and positive-verdict properties.

Keywords: Membrane computing · Kernel P systems · kP-Lingua · kP-Workbench · Model checking · Indexation · Genetic algorithms

1 Introduction

Membrane computing studies computational models inspired by cellular organization and by the evolution and communication of multisets of objects [1–4]. Kernel P systems provide a graph-based membrane-computing formalism

combining typed compartments, object multisets, rewriting and communication rules, guards, and execution strategies [5]. Their executable realization through kP-Lingua and kPWorkbench supports modelling, simulation, and verification-oriented analysis [6]. Recent kP-Lingua preprocessing and indexation mechanisms further allow parameterized families of rules, objects, guards, and links to be expressed compactly [7].

Current membrane-computing research continues to develop both the underlying formalisms and their software, application, and optimization perspectives. Recent studies include tool- and model-oriented developments for neural P systems, evolutionary and membrane-inspired optimization strategies, and new formal variants of P systems [31–33]. Related P-Lingua software resources also support the executable-model perspective adopted here [8]. Formal verification provides a complementary foundation: temporal logics and explicit-state or symbolic model checking make it possible to state and evaluate safety, liveness, response, and structural requirements [12–16]. SPIN and NuSMV are used as verification back ends in the experiments reported below [17–19]. A remaining methodological issue is how to connect these capabilities without conflating their roles. A compact indexed specification, a promising descriptor-level candidate, a successful concrete simulation, and a satisfied temporal property constitute different forms of evidence. This paper therefore investigates how source-level indexation and refinement, external candidate generation, real kP-Workbench execution, execution-derived selection, and trace-level verification can be organized as consecutive and reproducible stages. Three complementary benchmarks are used: Broadcasting for communication and cleanup, Partition/-Subset Sum for combinatorial generation and pruning, and Binary Adder for indexed arithmetic and carry propagation. The complete executable sources and the Python/Jupyter implementations of the external layer are distributed with the experimental resources in the public kPWorkbench repository. The paper makes four contributions. First, it defines a benchmark methodology for comparing standard kP sources with indexed or refined specifications while keeping source-level and workflow-level evidence separate. Second, it reports simulation and property-checking results for Broadcasting, Partition/Subset Sum, and Binary Adder. Third, it defines and implements a benchmark-specific external candidate-generation layer in which descriptor-level proxy ranking is followed by source materialization, real execution, and execution-derived representative selection. Fourth, it evaluates selected representatives through finite NuSMV

Table 1. Role of the three benchmarks in the comparative study.

Benchmark	Main modelling issue	Refined/indexed evidence	Experimental focus
Broadcasting	communication under a threshold	direct return rules, stopped markers, and cleanup	threshold reachability and absence of residual messages
Partition/Subset Sum	combinatorial branch growth	target-sum pruning and explicit verdict structure	branch reduction and verdict consistency
Binary Adder	arithmetic representation	explicit sum bits and carry propagation	canonical output and arithmetic agreement

trace abstractions and reports both the main verification campaign and a complementary Subset Sum control experiment. The external search follows standard evolutionary-computation principles [25, 26, 28–30], but the formal evidence reported in this study is obtained only from materialized executable sources and their associated verification artefacts.

2 Benchmark Problems and Study Scope

Table 1 summarizes the computational role, refined evidence, and experimental focus associated with each benchmark. The study considers three benchmark problems that exercise complementary parts of the kP modelling and verification workflow. The first problem, *Broadcasting*, models the controlled circulation of message families between a central compartment and a set of receivers. It supports the analysis of communication, threshold reachability, post-threshold cleanup, and the distinction between a blocked terminal configuration and a completed execution with explicit stopping evidence. The initial source provides a compact communication baseline, whereas the refined source introduces direct receiver-side return rules and stopped markers. The second problem, *Partition/Subset Sum*, represents a classical combinatorial search based on include/exclude decisions [23, 24]. The initial source provides an exhaustive partition-style baseline, while the pruning-aware source represents target-sum reachability and avoids extending branches that cannot reach a valid target under positive weights. This benchmark supports the analysis of branch growth, pruning, verdict consistency, and source-level safety properties. The third problem, *Binary Adder*, compares arithmetic expressed through object multiplicities with an indexed full-adder representation. The symbolic source preserves numerical value at the multiset level, whereas the indexed source makes input bits, sum bits, and carry objects explicit. It is used to analyse carry propagation, input cleanup, canonical output, and arithmetic agreement. Together, the three benchmarks cover communication behaviour, combinatorial branching, and arithmetic structure within a common comparative workflow. Broadcasting is parameterized by the number of receiver families k and a threshold θ. The central compartment emits message objects b_i as active objects a_i, receives acknowledgements, and accumulates a counter object c. The main source-level questions are whether the threshold is reached, whether active messages are removed after completion, and whether the final configuration records completion explicitly. For Partition/Subset Sum, let $V = \{v_1, \ldots, v_n\}$, with positive integer weights $w(v_i) = k_i$. Partition asks whether there exist V_1, V_2 such that $V_1 \cap V_2 = \emptyset, V_1 \cup V_2 = V, \sum_{v_i \in V_1} w(v_i) = \sum_{v_i \in V_2} w(v_i)$. For a prescribed positive target value K, the Subset Sum variant asks whether there exists a subset $V_1 \subseteq V$ such that $\sum_{v_i \in V_1} w(v_i) = K$. Thus, K is part of the problem instance and denotes the target sum against which the generated subsets are evaluated. The initial source exposes the generated subset space, whereas the pruning-aware source uses the target condition to avoid extending branches that cannot reach K under positive weights. For Binary Adder, the symbolic baseline uses multi-

Table 2. Main notation.

Symbol	Meaning	Symbol	Meaning
Π, A, μ, C_i, i_0	kP system components	d	number of compartments
k, θ	Broadcasting size and threshold	n, K	Subset Sum size and target
$v_i, w(v_i), k_i$	item and positive weight	m_{bit}	Binary Adder bit width
a_i, b_i, c	active, returned, and counter objects	$carry_i$	carry object at position i
F^{proxy}	proxy-fitness score	V, H	validity and reachability indicators
B_{term}	retained terminal branches	S_{est}	estimated execution stages

plicity rewriting, $2a_i \to a_{i+1}$, whereas the indexed full-adder source represents each bit position through input, sum, and carry objects. The relevant properties concern both the value represented by the final multiset and the canonical form of the binary output. Throughout the paper, k and θ denote the Broadcasting size and threshold, n and K denote the Subset Sum size and target, and m_{bit} denotes the Binary Adder bit width.

3 Background, Benchmarks, and Source Variants

Following the standard kernel P-system formalization [5], a kernel P system of degree d is written as $\Pi = (A, \mu, C_1, \ldots, C_d, i_0)$, where A is a finite alphabet of objects, μ is the graph describing the communication links between compartments, $C_1, \ldots, C_d$ are the compartments of the system, and i_0 identifies the output compartment. Each compartment has a type, a current multiset of objects, and an associated rule set together with an execution strategy. Objects represent the local state of a compartment through their multiplicities. Kernel P rules can rewrite objects locally, communicate objects between linked compartments, divide or dissolve compartments, or modify the communication graph. A rule may be guarded by conditions over object multiplicities; such a rule is applicable only when its guard is satisfied in the current configuration. At each computation step, applicable rules are selected according to the strategy associated with the compartment type. Strategies such as *choice* select among enabled alternatives, whereas maximal parallel strategies apply a maximal compatible collection of rules. The simultaneous effects of the selected rules determine the object multisets and, when structural rules are present, the compartment graph of the next configuration. This operational view is the one used by the executable kP-Lingua sources and by the kPWorkbench simulations considered in the experiments. The notation used throughout the source-level and workflow-level analyses is summarized in Table 2. kP-Lingua indexation is used as a syntactic macro mechanism that compactly generates families of rules, objects, guards, and links without changing the underlying operational semantics. Indexed and non-indexed sources are therefore comparable when instantiated with the same parameters. Formal verification is considered at two complementary levels. At source level, KPQ/LTL formulas are checked through the kPWorkbench–SPIN/NuSMV toolchain whenever the required predicates

Table 3. Source variants and associated source-level property families.

Problem	Variants	Main refinement	Property families
Broadcasting	B-NI, B-I, B-R	direct return and stopped markers	threshold, cleanup, residual messages
Subset Sum	P-NI, P-I, P-PR	pruning-aware target-sum source	reachability, exclusivity, pruning
Binary Adder	BS-NI, BS-I, BA-I	explicit full-adder carry structure	sum bits, carry, cleanup, agreement

are directly available. At workflow level, selected concrete simulation traces are translated into finite NuSMV abstractions and checked against temporal properties. Safety properties express the absence of undesirable states, liveness properties capture eventual events, and structural invariants describe branch, object, or arithmetic relations. The study uses three benchmarks with complementary computational structures. *Broadcasting* models messages transmitted from a central compartment to receiver compartments under a threshold counter. The refined source introduces direct receiver-side return rules and explicit stopped markers, providing a clear representation of completed execution. *Partition/Subset Sum* starts from exhaustive include/exclude branch generation and then considers a pruning-aware target-sum source in which branches that cannot reach the target under positive weights are not extended. *Binary Adder* starts from symbolic addition based on multiplicity rules $2a_i \to a_{i+1}$ and moves to an indexed full-adder representation with explicit input, sum, and carry objects. These benchmarks jointly cover communication, combinatorial branching, and arithmetic computation. The variant suffixes denote non-indexed (NI), indexed (I), refined (R), and pruning-aware (PR) sources. Accordingly, B, P, BS, and BA denote Broadcasting, Partition/Subset Sum, Binary Symbolic, and Binary Adder sources, respectively. Table 3 identifies the variants considered for each benchmark and their associated source-level property families. The executable source files are available in the public repository cited in the introduction. The following analysis compares their modelling choices, source-level properties, and workflow-level verification evidence.

Source-Level Modelling Choices. The source-level comparison focuses on the modelling decisions that determine both execution behaviour and verification readiness. In Broadcasting, the subscript i identifies a message family. The central compartment initially stores an object b_i, which becomes the active message a_i when it is sent to a receiver. Receiver i is associated with the local marker x_i, used by the initial source to identify the message family that the receiver is expected to acknowledge. Because the target receiver is selected nondeterministically among compartments of the same type, an a_i object can reach a receiver carrying x_j, with $j \neq i$; in that configuration, the corresponding guarded acknowledgement rule is not enabled and the active message may remain blocked. The refined source removes this mismatch by using direct receiver-side return rules and introduces explicit `stopped_i` markers to represent completion. The consequences of this modelling change are evaluated later by the source-level simulation and property results, where blocking-induced termination is distinguished from explicit completion.

Table 4. Main stages of the external candidate-generation and verification workflow.

Stage	Role	Evidence retained
Candidate generation	Build a static set of variants or a population of chromosomes	Candidate descriptors and proxy scores
Materialization	Export selected descriptors as executable `.kpl` sources	Concrete source files
Real execution	Run generated sources in kPWorkbench	Logs, halting status, steps, objects, verdicts
Metric selection	Rank candidates using real metrics	One representative per case
Trace checking	Check finite NuSMV abstractions of selected traces	LTL/CTL truth values

In Partition/Subset Sum, the initial source is retained as an exhaustive branch-generation baseline. It exposes the 2^n candidate space and provides a reference for source-level simulation. The pruning-aware source adopts the target-sum interpretation and stops extending branches whose partial sums cannot reach the target under positive weights. The associated checks do not treat the two sources as contradictory; instead, they examine whether directly expressible safety and verdict properties are preserved while the explored branching structure is reduced. In Binary Adder, the symbolic source is value-oriented: multiplicity rules of the form $2a_i \to a_{i+1}$ preserve the numerical value represented by the multiset, although a terminal multiset may not always be canonical. The indexed full-adder source is structure-oriented and represents input bits, sum bits, and carry objects explicitly. This representation may require more execution steps for some instances, but it supports local carry properties, explicit cleanup conditions, and canonical binary output. The comparison is therefore based on arithmetic agreement and verification readiness rather than on step count alone.

4 Candidate-Generation and Verification Workflow

The workflow integrates candidate generation, executable simulation, metric-based selection, and temporal verification. Its input consists of a benchmark configuration specifying the source template, case parameters, candidate representation, proxy-fitness components, and execution plan. Its output is a set of case-level representatives supported by real kPWorkbench metrics and temporal-property checks. Table 4 summarizes the main workflow stages and the evidence retained at each stage. The external layer is implemented in Python through the Jupyter notebooks distributed with the public experimental artefacts. Its role and configuration are benchmark-specific. Broadcasting uses a deterministic candidate-generation and proxy-ranking procedure rather than a generational genetic algorithm. Subset Sum and Binary Adder use genetic algorithms operating on descriptor-level chromosomes. In all cases, the external layer precedes kPWorkbench execution: proxy scores guide generation or preliminary ranking, whereas the representatives reported later are selected from metrics extracted from real executable runs. The descriptor-level choices and proxy indicators used for the three benchmarks are summarized in Table 5. For Broadcasting, the deterministic grid contains six cases and six candidate policies per case. The ordering policies are ascending, descending, odd–even, even–odd,

Table 5. Descriptor-level choices used by the external layer.

Benchmark	Candidate descriptor	Proxy-level indicators
Broadcasting	case parameters and refined source	threshold reachability, cleanup estimate, order smoothness
Subset Sum	item ordering and pruning trajectory	structural validity, target reachability, retained branches, estimated stages
Binary Adder	block size, carry strategy, and grouping	carry suitability, rule regularity, scalability, verification readiness

Table 6. External candidate-generation configurations used in the experiments.

Parameter	Broadcasting	Subset Sum	Binary Adder
Mode	deterministic ranking	genetic	genetic
Chromosome / descriptor	ordering and stop policy	item-weight permutation	block size, carry strategy, grouping
Population size	–	20	30
Generations	–	20	20
Elitism	–	2	3
Selection	–	tournament (3)	tournament (3)
Crossover	–	order, $p_c = 0.80$	uniform gene-wise, $p_c = 0.82$
Mutation	–	swap, $p_m = 0.15$	single-gene, $p_m = 0.18$
Random seed	deterministic per candidate	42	2026
Export per case	6 candidates	up to 10	6

center-out, and deterministically shuffled orderings, combined with benchmark-specific stopping policies. The pseudo-random ordering uses the deterministic seed $2026 + 6(c-1) + j$, where c and j denote the case and candidate indices, respectively. Consequently, regeneration of the same candidate grid does not depend on an uncontrolled random state.

For Subset Sum, a chromosome is a permutation $\chi_{\mathrm{SS}} = \left(w_{\pi(1)}, \ldots, w_{\pi(n)}\right)$ of the item weights. A population of 20 permutations is evolved for 20 generations. At each generation, the two highest-ranked chromosomes are copied unchanged to the next population. The remaining positions are generated from parents selected by tournaments of size three. Order crossover is applied with probability 0.80; otherwise the first selected parent is copied. Swap mutation, applied with probability 0.15, exchanges two randomly selected positions. The run is initialized with random seed 42, and up to the ten highest-ranked descriptors per case are exported for the subsequent executable workflow.

For Binary Adder, a chromosome is $\chi_{\mathrm{BA}} = (\mathit{block_size}, \mathit{carry_strategy}, \mathit{grouping})$, where the first gene selects an admissible block size, the second selects one of the serial, block-carry, lookahead-like, or hybrid carry strategies, and the third specifies the ordering/grouping policy used when generating the indexed rule families. A population of 30 chromosomes is evolved for 20 generations. The three highest-ranked chromosomes are retained unchanged. The remaining population is created using tournaments of size three, uniform gene-wise crossover with probability 0.82, and single-gene mutation with probability 0.18. Uniform crossover independently takes each of the three genes from the second parent with probability 0.5; mutation selects one gene and replaces it with another admissible value. The implementation uses random seed 2026 and exports the

Table 7. The algorithmic steps of the verification-aware workflow.

Step	Operation	Scientific role
1	Instantiate benchmark case	fixes k, θ, n, K, or m_{bit} and the source template
2	Generate candidate descriptor	represents ordering, pruning, grouping, or carry-related design choices
3	Rank by proxy indicators	filters candidates before real execution
4	Materialize `.kpl` source	converts descriptors into executable kP-Lingua artefacts
5	Execute in kPWorkbench	obtains real steps, halting status, objects, verdicts, and rates
6	Select case representative	uses real metrics for the reported representative
7	Check NuSMV trace abstraction	evaluates LTL/CTL properties over selected real behaviours

six highest-ranked candidates per case. The associated execution plan contains ten kPWorkbench runs per exported candidate before real-metric aggregation. Table 7 presents the workflow as a compact sequence of candidate-generation, execution, selection, and verification steps.

The optimization objective of the external layer is to prioritize descriptors that are structurally admissible and operationally promising before the more expensive executable evaluation. The resulting values are therefore *proxy* scores and must not be interpreted as model-checking results. For Subset Sum, the implemented proxy score is $F_{\text{SS}}^{\text{proxy}} = 10000V + 500H - B_{\text{term}} - 0.1S_{\text{est}}$, where V indicates structural validity, H indicates whether the target is reachable in the pruned descriptor-level calculation, B_{term} is the retained terminal-branch count, and S_{est} is the estimated number of stages. The coefficients implement a lexicographic preference: structural validity dominates target reachability, while retained branching and estimated depth act as secondary efficiency criteria. For Binary Adder, the implementation combines normalized descriptor scores as $F_{\text{BA}}^{\text{proxy}} = 100\big(0.32C + 0.18B + 0.15G + 0.12R + 0.15S + 0.08Q\big)$, where C, B, G, R, S, and Q denote carry-strategy suitability, block-size suitability, grouping compatibility, rule regularity, scalability, and verification-readiness indicators, respectively. Their weights express the design-time emphasis of the descriptor ranking and are not learned from the final verification outcomes. After materialization, candidates are evaluated through real kPWorkbench executions. Halting status, execution steps, final objects, residual objects, verdicts, validity, success, cleanup, and benchmark-specific correctness indicators are extracted from the logs. Execution-derived metrics, rather than proxy fitness alone, determine the case representative subsequently submitted to NuSMV. Table 6 fixes the implementation parameters needed to reproduce the candidate-generation stage. In both genetic benchmarks, the replacement mechanism is generational with elitism: the configured number of highest-ranked chromosomes is first copied unchanged, after which tournament selection, probabilistic crossover, and mutation are repeatedly applied until the next population reaches its configured size. The evolutionary process terminates after the fixed 20-generation budget. These operations affect only candidate descriptors; they do not execute kPWorkbench and do not evaluate SPIN or NuSMV properties. Candidate descriptors become verification evidence only after materialization, executable simulation, real-metric extraction, and representative selection. After materialization, each

Table 8. Evidence preserved for each benchmark.

Benchmark	Source-level evidence	Workflow-level evidence	Main interpretation
Broadcasting	SPIN/NuSMV threshold and cleanup checks	78/78 NuSMV trace checks	explicit completion and cleanup
Subset Sum	SPIN safety checks and pruning comparison	66/66 NuSMV trace checks	reduced branching with preserved verdicts
Binary Adder	SPIN/NuSMV arithmetic checks	138/138 NuSMV trace checks	canonical output through indexed carry structure

source is evaluated through concrete kPWorkbench executions. The collected metrics include halting status, number of steps, final object multiplicities, residual active objects, threshold or target reachability, completion rate, validity rate, success rate, and, for Binary Adder, arithmetic correctness. Proxy indicators are therefore used for search and preliminary ranking, whereas execution metrics determine the reported case representative. In Broadcasting, the representative is selected according to completion and cleanup behaviour for the given case parameters. In Subset Sum, the selected source preserves the relation between the target state and the reported verdict after pruning. In Binary Adder, selection considers canonical sum bits, carry cleanup, and arithmetic agreement. The subsequent NuSMV checks are applied to these concrete representatives, linking workflow-level verification to executable outputs. The verification layer uses property families adapted to the observable structure of each benchmark. Broadcasting properties address threshold reachability, cleanup after threshold, stopped-family evidence, and the absence of residual active messages. Subset Sum properties address target reachability, agreement between target states and positive verdicts, output exclusivity, and cleanup of intermediate objects. Binary Adder properties address sum-bit completeness, final-carry availability, input cleanup, carry cleanup, and arithmetic agreement. Source-level and workflow-level evidence remain distinct throughout the evaluation. Source-level properties are checked on the available KPQ/SPIN/NuSMV translations of the initial and refined sources. Workflow-level properties are checked on finite NuSMV abstractions generated from selected simulation traces. Structural invariants describe stable relations among branches, objects, or arithmetic components, while LTL/CTL formulas express reachability, response, cleanup, and exclusivity conditions. The correspondence between source-level and workflow-level evidence is summarized in Table 8. Table 9 records the input, output, and purpose of each phase in the evaluation protocol. The two experimental levels are connected through the materialized executable sources. Source-level analysis compares the initial and refined or indexed specifications, whereas workflow-level analysis evaluates the representatives produced by candidate generation and selection. This organization relates modelling choices, execution evidence, and temporal verification within a single experimental workflow.

Table 9. Evaluation protocol retained across the three benchmarks.

Phase	Input	Output	Used for
Source simulation	initial/refined source	steps, final objects, residuals	behavioural comparison
Source checking	KPQ/LTL properties	SPIN/NuSMV outcomes	safety and liveness evidence
Candidate execution	generated `.kpl` sources	logs and real metrics	representative selection
Trace checking	selected traces	LTL/CTL truth values	workflow-level verification

Table 10. Selected source-level simulation results for the initial and refined Broadcasting sources. The two column groups list the available cases independently.

Initial source				Refined source			
Case	k	θ	Avg. steps	Case	k	Avg. steps	Final status
1	3	10	4.70	1	3	9.10	complete, 3/3 stopped
2	5	10	6.10	2	5	9.60	complete, 5/5 stopped
3	10	10	11.40	3	10	13.30	complete, 10/10 stopped
4	15	10	16.40	4	15	17.80	complete, 15/15 stopped
5	20	10	20.90	5	20	20.00	partial, 17–18/20 stopped
6	30	10	31.10	6	30	20.00	partial, 17–18/30 stopped
8	50	10	51.00	8	50	20.00	partial, 17–18/50 stopped
10	100	10	100.60	9	75	20.00	partial, 18/75 stopped

5 Source-Level Simulation and Property Results

This section reports the direct source-level campaign and compares the standard and indexed or refined sources before the external candidate-generation workflow is applied. Ten kPWorkbench runs were performed for each source-case pair in the source-level simulation campaign. The analysis combines execution metrics, source-level property checks, and representative trace-level property groups in order to relate observable behaviour to the corresponding verification evidence. Broadcasting presents a direct contrast between the initial communication structure and the refined completion mechanism. In the initial source, message delivery is combined with receiver-side matching conditions, while the refined source uses direct return rules and explicit stopped markers. Table 10 reports selected execution measurements for the initial and refined Broadcasting sources. The two column groups summarize the available cases independently and should not be interpreted as row-wise paired configurations. The results show that the refined representation records completion through stopped-family markers and supports direct inspection of cleanup behaviour. For larger instances, the fixed execution window produces partial stopped-family coverage, which is reported explicitly in the final status column. Subset Sum compares exhaustive branch generation with the pruning-aware target-sum source. The two variants represent related search tasks: the initial source exposes the complete partition-style branch space, whereas the pruned source evaluates target-sum reachability while limiting branch extension under positive weights. Table 11 reports the branch counts and result patterns for the exhaustive and pruning-

Table 11. Source-level comparison for Subset Sum.

Case	n	K	Initial	Pruned	Reduction	Pruned t/f	Verdicts
1	3	3	8	5	37.50%	2/3	yes/yes
2	4	5	16	9	43.75%	2/7	yes/yes
3	5	7	32	16	50.00%	3/13	no/yes
4	6	10	64	32	50.00%	5/27	no/yes
5	8	18	256	135	47.27%	14/121	yes/yes
6	10	27	1024	512	50.00%	40/472	no/yes

Table 12. Initial symbolic addition vs. indexed Binary Adder.

Case	input pos.	repr. value	BA steps	Initial final form	BA final form
1	1	2	2	canonical	canonical
2	3	3	4	canonical	canonical
3	3	8	4	non-canonical	canonical
4	3	16	4	non-canonical	canonical
5	7	255	8	canonical	canonical
6	7	256	8	non-canonical	canonical
7	15	256	16	non-canonical	canonical
8	31	2^{32}	32	non-canonical	canonical

aware Subset Sum sources. Here, t/f denotes the numbers of retained terminal branches associated with true and false target outcomes, respectively, while the two verdict entries refer to the initial and pruning-aware sources. Across the reported cases, the pruning-aware source retains approximately 37%–50% fewer terminal branches than the exhaustive source. The result pattern is interpreted with respect to the semantics of each variant, since the initial source provides a partition-style baseline and the pruned source evaluates target-sum reachability. Binary Adder compares symbolic addition through object multiplicities with an indexed full-adder source. The indexed representation exposes carry propagation and produces explicit binary output objects, whereas the symbolic source preserves value through rewriting rules of the form $2a_i \to a_{i+1}$. The symbolic and indexed Binary Adder executions are compared in Table 12. The indexed source may require additional execution steps because carry propagation is represented explicitly at each bit position. This structure also provides canonical binary output and supports local properties concerning sum bits, carries, and cleanup. The simulation campaign was complemented by source-level KPQ/LTL checking. The properties were translated through the available kPWorkbench verification workflow and evaluated with SPIN and, where the corresponding runs completed, NuSMV. Table 13 reports the source-level Broadcasting checks for the initial and refined sources. In this and the following source-level tables, N/A indicates that the corresponding source variant does not expose a direct object-level predicate for the property, whereas N.C. indicates that the NuSMV execution did not complete within the available run window. After correction of the verifier translations, both SPIN and NuSMV confirm threshold reachability for the initial and refined Broadcasting sources. The property outcomes distinguish the two completion interpretations: the initial source reaches sta-

Table 13. Source-level property checking for Broadcasting.

Property	Source-level check	Init. SPIN	Init. NuSMV	Ref. SPIN	Ref. NuSMV
Threshold reachability	`F(c1.c >= 10)`	TRUE	TRUE	TRUE	TRUE
Response of a_1 by b_1	`G(c21.a1>0 -> F c1.b1>0)`	TRUE	TRUE	N/A	N/A
Cleanup after threshold	`G(c1.c>=10 -> c2i.ai=0)`	FALSE	FALSE	TRUE	TRUE
Stopped-family evidence	`F(all stopped_i)`	N/A	N/A	TRUE	TRUE
No residual messages	`G(completion -> no active a_i)`	FALSE	FALSE	TRUE	TRUE

Table 14. Source-level property checking for Subset Sum.

Property	Source-level check	Initial SPIN	Pruned SPIN
Target/success reachability	`F(t1.t > 0)` or equivalent marker	TRUE	TRUE
Positive-verdict agreement	`G(t0.yes>0 -> t1.t>0)`	TRUE	TRUE
Terminal marker reachability	`F(t2.x > 0)`	FALSE	FALSE
Output exclusivity	`G !(yes_out & no_out)`	TRUE	TRUE
Residuals, halting, and steps	trace-level metadata	N/A	N/A

ble blocked configurations with residual active objects in some runs, whereas the refined source records completion through stopped markers and satisfies the corresponding cleanup properties. The source-level SPIN outcomes for the initial and pruning-aware Subset Sum sources are reported in Table 14. For Subset Sum, SPIN confirms the directly expressible source-level relations, including target reachability, positive-verdict agreement, and output exclusivity. The pruning-aware source retains these checked relations while reducing the explored branch structure. NuSMV checking of the partition source remained outside the completed execution window and is therefore reported separately from the available SPIN evidence. Table 15 reports the available checks for the symbolic and indexed Binary Adder sources. For Binary Adder, the indexed source exposes object-level predicates for sum-bit completion, carry availability, carry cleanup, and arithmetic agreement. The initial symbolic source remains value-oriented, and the SPIN and NuSMV outcomes for sum-bit completeness reflect the different asynchronous translations used by the two verification back ends. After correction of the verifier mappings, the indexed-source results agree for the directly expressible properties and support property-oriented analysis of the full-adder structure. The workflow-level NuSMV abstractions use Boolean pred-

Table 15. Source-level property checking for Binary Adder.

Property	Source-level check	Init. SPIN	Init. NuSMV	Indexed SPIN	Indexed NuSMV
Natural halting	explicit halt predicate	N/A	N/A	N/A	N/A
Sum-bit completeness	`F(output sum objects)`	TRUE	TRUE	TRUE	TRUE
Final-carry availability	`F(carry4_0 > 0)`	N/A	N/A	TRUE	N.C.
Input cleanup	`F(no input bit objects)`	TRUE	TRUE	TRUE	TRUE
Carry cleanup	`F(no intermediate carry)`	N/A	N/A	TRUE	TRUE
Arithmetic agreement	expected output-bit encoding	N/A	N/A	TRUE	TRUE

Table 16. Representative workflow-level temporal property groups.

Problem	Property group	Representative formula
Broadcasting	natural halting	`LTLSPEC F halted`
Broadcasting	cleanup after threshold	`G(threshold_reached -> F no_active_messages)`
Broadcasting	no residual messages	`G(halted -> no_residual_messages)`
Subset Sum	target reachability	`LTLSPEC F target_reached`
Subset Sum	positive-verdict agreement	`G(positive_verdict -> target_reached)`
Subset Sum	output exclusivity	`G !(yes_out & no_out)`
Binary Adder	sum-bit completeness	`G(halted -> all_sum_bits_ready)`
Binary Adder	carry cleanup	`G(halted -> no_intermediate_carry)`
Binary Adder	arithmetic agreement	`G(halted -> arithmetic_ok)`

icates derived from concrete simulation traces. Table 16 summarizes the main property groups evaluated after representative selection, while the complete generated artefacts contain the expanded declarations and formula instances. The source-level tables describe the behaviour observed before the external workflow is applied. Broadcasting distinguishes stable blocking from completion recorded through explicit markers, Subset Sum quantifies the effect of target-aware pruning, and Binary Adder distinguishes value preservation from canonical bit-level output. The property tables complement these observations by identifying the predicates that are directly available for each source variant and verification back end. The workflow-level representatives are selected according to the structure of each benchmark. Broadcasting uses static candidate ranking because the refined source and the case parameters already define the relevant instances. For Subset Sum, descriptor choices concern item ordering and pruning trajectories, which influence retained branches and execution stages. For Binary Adder, the candidate descriptors concern grouping and carry organization, which influence the regularity of the generated source and its property structure. The reported fitness values are computed from real kPWorkbench executions after the generated sources have been materialized and their logs parsed. Proxy scores are used to organize candidate search, while real execution metrics determine the reported representatives. NuSMV verification is subsequently applied to the selected concrete traces. Table 17 aggregates the primary verification campaign over the selected represen-

Table 17. Workflow-level summary across selected representatives.

Problem	Cases	Real runs	NuSMV properties	False	Status
Broadcasting	6	375	78	0	PASS
Subset Sum	6	468	66	0	PASS
Binary Adder	6	360	138	0	PASS
Total	18	1203	282	0	PASS

Table 18. Subset Sum control audit using non-vacuous success properties.

Property	Selected	Unselected controls
F *reached_target*	TRUE	FALSE
AF *reached_target*	TRUE	FALSE
F *positive_global_verdict*	TRUE	FALSE
AF *positive_global_verdict*	TRUE	FALSE

tatives. A separate Subset Sum control audit, reported below, is not included in the 282-property total. For Subset Sum, the six selected representatives were additionally checked against four non-vacuous success properties: F *reached_target*, AF *reached_target*, F *positive_global_verdict*, and AF *positive_global_verdict*. All four properties hold for every selected representative. The same checks were applied to three lower-fitness unselected candidates: `case02_cand001`, `case03_cand001`, and `case04_cand004`; all four properties are false for each control candidate. Together with the 11 standard properties, this gives 15/15 true outcomes for each selected Subset Sum representative. Each unselected control satisfies the 11 standard structural and consistency properties but fails the four non-vacuous success properties, yielding 11/15 true outcomes. Overall, the section connects direct source-level simulation, translated-source checking, and temporal verification of concrete traces. The additional Subset Sum controls show that the non-vacuous reachability and positive-verdict properties distinguish the selected representatives from sampled lower-fitness candidates, while both groups retain the standard structural properties.

6 Discussion, Conclusions, and Future Work

The results should be interpreted primarily as a comparison of modelling and verification choices rather than as a uniform comparison of execution speed. The refined Broadcasting source separates threshold reachability from cleanup and records completion through explicit stopped markers. The pruning-aware Subset Sum source reduces the explored branch structure while retaining the directly checked safety and verdict relations. The indexed Binary Adder may require additional steps because carry propagation is represented explicitly, but this representation exposes the sum and carry structures needed for canonical output and local arithmetic properties. Refinement and indexation therefore contribute principally to semantic explicitness, parameterization, and verification accessibility; they do not remove the intrinsic computational complexity of the underlying benchmark. The source-level experiments also illustrate the importance of verifier-specific translations. During model development, differences

between SPIN and NuSMV were associated with object mappings, execution semantics, and finite bounds. For example, the Broadcasting counter bound must allow a valid threshold overshoot: when $\theta = 10$ and $k = 3$, a configuration containing $9c$ may receive three additional counter objects and therefore requires the finite abstraction to represent $12c$. In Binary Adder, output objects must likewise be mapped to the intended compartments and bit positions. These observations motivate keeping the executable source, its concrete simulation evidence, and the verifier-specific artefact as identifiable stages of the experimental record. The external layer has a complementary role. It explores or organizes descriptor-level alternatives before executable evaluation but does not itself provide formal correctness evidence. Broadcasting uses deterministic candidate generation, whereas Subset Sum and Binary Adder use the genetic configurations reported in Table 6. In all three cases, descriptor-level proxy ranking is followed by source materialization, kPWorkbench execution, metric extraction, and case-level representative selection. NuSMV is applied only after these stages. This separation prevents a high proxy score from being interpreted as either a successful simulation or a satisfied temporal property. The Subset Sum control audit gives a concrete example of the discriminating role of the temporal properties. Each selected representative satisfies all 15 examined properties, whereas the sampled lower-fitness candidates `case02_cand001`, `case03_cand001`, and `case04_cand004` satisfy the 11 structural and consistency properties but fail the four non-vacuous target-reachability and positive-verdict properties. The distinction is therefore not produced by changing the property set between selected and unselected candidates, but by applying the same success-oriented temporal requirements to behaviours with different observed outcomes. The verification claims remain deliberately trace-level. The workflow-level NuSMV models are finite abstractions constructed from concrete kPWorkbench simulation evidence and from the predicates required by the reported formulas. They validate the selected observed behaviours but do not constitute a complete semantic translation of the full kP-Lingua language. Source-level SPIN/NuSMV checks and workflow-level trace abstractions must consequently be read as complementary evidence rather than as interchangeable verification layers. In summary, the study provides an integrated verification-aware methodology connecting indexed kernel P-system modelling, benchmark-specific external candidate generation, real executable evaluation, execution-derived selection, and temporal checking across communication, combinatorial, and arithmetic benchmarks. Its principal contribution is the explicit separation of descriptor-level search, executable behavioural evidence, and formal property outcomes while retaining a reproducible link between them. Future work includes improving the Promela sources used for verification, examining query-based verification in `kPWorkbench` or kP-Query, developing a closer correspondence between indexed `kP-Lingua` specifications and verifier-specific temporal models, extending the control experiments to larger candidate samples, and exploring complementary probabilistic model-checking tools such as MC2 [27] for stochastic refined variants.

A Human-Readable kP Encodings of the Benchmarks

This appendix summarizes the computational interpretation of the three kP encodings independently of their executable kP-Lingua syntax. The purpose is to make explicit how the objects, compartments, and principal rule families represent each benchmark. The complete executable specifications remain available with the experimental artefacts.

Broadcasting. The Broadcasting system contains one central compartment and a family of receiver compartments. For each family i, the central compartment initially contains an object b_i. Below threshold θ, b_i can be activated as a_i and transmitted to a receiver. Receiver i is conceptually associated with marker x_i. In the initial source, receivers share a common type and target selection is nondeterministic; an active object a_i can therefore reach a receiver marked x_j, $j \neq i$, where the guarded acknowledgement rule for family i is unavailable. A successful acknowledgement returns b_i to the central compartment and contributes i copies of counter object c. Once the multiplicity of c reaches the threshold, the execution enters its cleanup regime. The refined encoding replaces the problematic receiver-side matching behaviour by direct return rules and records completed families explicitly through `stopped_i` markers. Consequently, threshold reachability, cleanup, and completed-family evidence can be expressed as separate properties.

Partition/Subset Sum. The initial Partition/Subset Sum encoding represents the include/exclude decisions for n items through parallel branch generation. At stage i, a branch creates alternatives corresponding to inclusion or exclusion of item v_i; an object b_i records membership of v_i in the selected subset. After all n stages, the exhaustive construction can expose up to 2^n candidate branches. For Subset Sum, the instance additionally contains a prescribed target K. The pruning-aware encoding associates a partial selected sum with each active branch. Because the benchmark uses positive weights, an inclusion transition that would increase the partial sum beyond K cannot lead to a solution and is therefore not extended. A terminal branch whose accumulated value reaches K generates the positive target/verdict evidence; other retained terminal branches contribute negative evidence. The output rules enforce the intended yes/no interpretation and support exclusivity and verdict-consistency checks. Thus pruning changes the explored branch structure while preserving the target-sum semantics checked in the reported experiments.

Binary Adder. The symbolic Binary Adder baseline represents numerical value by object multiplicities. A rule $2a_i \rightarrow a_{i+1}$ implements a binary carry by replacing two objects of value 2^i with one object of value 2^{i+1}. This representation preserves numerical value but does not by itself expose a canonical sum bit at every position. The indexed full-adder variant instead represents both input bits explicitly. At position i, objects $x_{i,0}$ or $x_{i,1}$ encode the first input bit, $y_{i,0}$ or $y_{i,1}$ encode the second, and $carry_{i,0}$ or $carry_{i,1}$ encode the incoming carry. The eight combinations of these three Boolean inputs are represented by indexed full-adder rules that generate the corresponding $sum_{i,0/1}$ object and $carry_{i+1,0/1}$. Processing proceeds across the indexed bit positions until all sum

bits and the final carry have been determined. This explicit representation makes sum completeness, final-carry availability, input cleanup, working-carry cleanup, and arithmetic agreement directly available as verification predicates.

References

1. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1690
2. Păun, Gh.: Membrane Computing: An Introduction. Springer, Berlin (2002). https://doi.org/10.1007/978-3-642-56196-2
3. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010). https://global.oup.com/academic/product/the-oxford-handbook-of-membrane-computing-9780199556670
4. Rozenberg, G., Bäck, T., Kok, J.N. (eds.): Handbook of Natural Computing. Springer, Berlin (2012). https://doi.org/10.1007/978-3-540-92910-9
5. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., Lefticaru, R.: Kernel P systems: From modelling to verification and testing. Theoretical Computer Science 724, 45–60 (2018). https://doi.org/10.1016/j.tcs.2017.12.010
6. Kernel-P-Systems: kPWorkbench repository. https://github.com/Kernel-P-Systems/kPWorkbench
7. Iacob, S., Ipate, F., Niculescu, I.M., Konur, S.: Some new developments in kP-Lingua. Manuscript and kPWorkbench resources (2025). https://github.com/Kernel-P-Systems/kPWorkbench
8. Research Group on Natural Computing: P-Lingua and membrane computing software resources. https://www.gcn.us.es/
9. Ciobanu, Gh., Păun, Gh., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Berlin (2006). https://doi.org/10.1007/3-540-29937-8
10. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: A linear solution for QSAT using membrane creation. Theoretical Computer Science 371(1–2), 1–11 (2007). https://doi.org/10.1016/j.tcs.2006.11.007
11. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: P systems with limited resources. Theoretical Computer Science 529, 1–17 (2014). https://doi.org/10.1016/j.tcs.2014.01.005
12. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA (1999). https://mitpress.mit.edu/9780262032704/model-checking/
13. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge, MA (2008). https://mitpress.mit.edu/9780262026499/principles-of-model-checking/
14. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 46–57 (1977). https://doi.org/10.1109/SFCS.1977.32
15. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems 8(2), 244–263 (1986). https://doi.org/10.1145/5397.5399
16. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27, 333–354 (1983). https://doi.org/10.1016/0304-3975(82)90125-6

17. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on Software Engineering 23(5), 279–295 (1997). https://doi.org/10.1109/32.588521
18. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley (2004). https://spinroot.com/spin/Doc/Book_extras/
19. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: A new symbolic model checker. International Journal on Software Tools for Technology Transfer 2, 410–425 (2000). https://doi.org/10.1007/s100090050046
20. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
21. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H. et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer (2010). https://doi.org/10.1007/978-3-642-16612-9_11
22. Bobe, R.T., Konur, S.: Probabilistic verification of robot controllers using numerical P systems models. Manuscript (2025). https://github.com/Kernel-P-Systems/kPWorkbench
23. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
24. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979). https://dl.acm.org/doi/book/10.5555/578533
25. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975). https://mitpress.mit.edu/9780262581110/adaptation-in-natural-and-artificial-systems/
26. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989). https://dl.acm.org/doi/book/10.5555/534133
27. Gilbert, D., Donaldson, R.: A Monte Carlo model checker for probabilistic ltl with numerical constraints (2008)
28. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press (1998). https://mitpress.mit.edu/9780262631853/an-introduction-to-genetic-algorithms/
29. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing, 2nd edn. Springer (2015). https://doi.org/10.1007/978-3-662-44874-8
30. Whitley, D.: A genetic algorithm tutorial. Statistics and Computing 4, 65–85 (1994). https://doi.org/10.1007/BF00175354
31. Cabarle, F.G.C.: Thinking about spiking neural P systems: some theories, tools, and research topics. Journal of Membrane Computing 6, 148–167 (2024). https://doi.org/10.1007/s41965-024-00147-y
32. Andreu-Guzmán, J.A., Orellana-Martín, D., Valencia-Cabrera, L.: Dissecting OLMS membrane algorithms: understanding the role of communication and evolutionary operators in optimization strategies. Journal of Membrane Computing 7, 475–520 (2025). https://doi.org/10.1007/s41965-025-00200-4
33. Kuczik, A., Vaszil, Gy.: Non-cooperative polymorphic P systems and parallel communicating ET0L systems. Journal of Membrane Computing 7, 312–325 (2025). https://doi.org/10.1007/s41965-025-00185-0