ABSTRACT

EARTHPERSON, ARJUN. A Data-Parallel, Hardware-Accelerated Monte Carlo Framework for Quantifying Risk using Probabilistic Circuits. (Under the direction of Dr. Mihai A. Diaconeasa).

Quantitative risk assessment traditionally relies on exhaustive enumeration of minimal cut sets, a process that grows combinatorially with model size and therefore struggles with the multi-hundred–component systems encountered in contemporary nuclear safety studies. This dissertation introduces a Monte-Carlo paradigm that sidesteps explicit state-space exploration by evaluating the underlying probabilistic directed acyclic graph directly on modern data-parallel hardware, treating the full set of inter-linked event trees, fault trees, and their dependencies as a single unified computation graph. The proposed solver, mcSCRAM, is accompanied by a suite of theoretical and algorithmic advances. First, we formalize hardware-native gates for threshold and cardinality voting, prove their estimator equivalence to classical AND/OR expansions, and demonstrate exponential savings in graph size and kernel launch overhead for gates with high fan-in. Second, we develop a knowledge compilation pipeline, specifically tailored to Monte Carlo workloads. Third, we derive first-order importance measures and extensions for common-cause failure analysis within the Monte Carlo paradigm. Extensions include a composite convergence criterion that fuses frequentist, Bayesian, and information-theoretic diagnostics, guaranteeing statistical accuracy under user-specified budgets. Formal proofs establish unbiasedness, variance preservation, and convergence rates for all estimators. Comprehensive benchmarks on the 43-model Aralia fault tree dataset show that the compiled graphs, no longer subject to exact inference constraints, can be compressed by a median factor of 1.3×. During simulation, the solver achieves fully saturated device throughput, converging to sub-percent relative error on graphs with a few thousand events in under five seconds—even on entry-level consumer GPUs. The rare-event regime remains challenging but is mitigated by the proposed importance-sampling extension. Sensitivity studies confirm that randomness quality and convergence diagnostics

jointly prevent premature termination. The work demonstrates that data-parallel Monte-Carlo techniques can match the accuracy of exact solvers while scaling to models previously considered intractable. Limitations include residual inefficiency for events with probabilities below 10^{-8}, reliance on specialized hardware for peak performance, and the need for further validation on full-scale industry benchmarks. Nevertheless, the methodological foundation laid here opens avenues for stochastic, gradient-based optimization, adaptive variance reduction, and real-time risk monitoring in future research.

An Informal Overview

Why another PRA solver? Probabilistic Risk Assessment (PRA) for large nuclear systems asks a simple question: *how likely is an accident, given thousands of component failures, recoveries, and human actions?* The textbook solution (build a decision diagram, or enumerate every failure combination) scales exponentially and quickly becomes impractical. Over the years analysts have tamed that combinatorial surge with gate restrictions, bounding tricks, and rare-event approximations; the price has been either coarse error margins or hours-long runtimes.

A different stance: sample first, ask questions later. Instead of traversing every Boolean state, we draw random global scenarios and evaluate the entire logic graph in one pass. The key enabler is hardware parallelism: by packing 64 Bernoulli trials into a 64-bit word we turn Boolean gates into bitwise instructions and let modern Graphics Processing Unit (GPU)s churn through billions of scenarios per second. The solver, mcScram, views the complete PRA model (nested event trees, fault trees, and their cross-links) as one unified Probabilistic Directed Acyclic Graph (PDAG) that is sampled and tallied en bloc.

What falls out of that design?

- **Flexibility.** Any gate expressible in Boolean logic (NOT, k-out-of-n voting, XOR, Cardinality, ...) is handled without extra coding.

- **Speed.** On a mid-range laptop GPU the method converges to sub-percent error for graphs with a few thousand events in $\lesssim 5$ seconds.

- **Extensibility.** Because sampling is the only primitive, common-cause failures, importance measures, and importance sampling plug in naturally.

What it is *not*. The approach is still Monte-Carlo: for events rarer than 10^{-8} additional variance-reduction techniques are required, and maximum throughput depends on access to a GPU or other Single Instruction, Multiple Data (SIMD) hardware. It is also *not* an exact Boolean enumerator. By design we trade exactness for tractability and streamability: outputs are stochastic estimators with on-line confidence intervals that tighten as sample size grows. Practitioners must therefore choose a target error band and be comfortable with the residual uncertainty when stopping.

Road map.

- **Chapter 4 (Part I).** Formalizes PRA models as probabilistic DAGs.
- **Chapters 7–11 (Part III).** Present the data-parallel Monte-Carlo engine—sampling, gate kernels, tallying, and backend scalability.
- **Chapters 13–18 (Part IV).** Detail refinements: randomness guarantees, composite convergence policy, common-cause failure handling, importance measures, importance sampling, and hardware-native voting gates.
- **Chapters 12 and 19.** Benchmark accuracy, runtime, and structural compression on the Aralia suite.
- **Part V.** Outlines future directions such as inverse problems and parameter fitting.

A Data-Parallel, Hardware-Accelerated Monte Carlo Framework for Quantifying Risk using Probabilistic Circuits

by
Arjun Earthperson

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Nuclear Engineering

Raleigh, North Carolina
2025

APPROVED BY:

Dr. Aydin Aysu

Dr. Yousry Azmy

Dr. Nam Dinh

Dr. Mihai A. Diaconeasa
Chair of Advisory Committee

DEDICATION

To my earliest teachers – my parents, Ajay Verma and Sunita Bainsla; my grandparents, aunts and uncles – who taught me curiosity, kindness, discipline, and the joy of learning long before I had words for any of it.

To the career mathematicians who were, above all, mentors – Mr. Biren Chamola and Mr. Lalit Dwivedi, who kept my curiosity for math alive by meeting me where I was: in philosophy, stories, and humor[1].

To those at UCLA who believed in me over the years – Dean Richard Wesel, Dr. Saeromi Kim, Dr. Chip McDaniels, and Dennis Briggs – for advocating for me and showing me how to advocate for myself.

To Ms. Elizabeth Bell, Mr. Angel Viramontes, and the dedicated Math Department at WLAC, where I learned how to learn, and found my footing again.

To my sister, RUC, who, while I was an ocean away and unable to travel home for more than a decade, quietly shouldered our shared family responsibilities without ever asking for anything in return. I owe you more than I can say.

To my thoughtful, patient, hopeful wife, Halle Hampy, who puts out fires nearly every day, sometimes just to get me out the door and to my desk; who cared for the nine cats we accidentally adopted last year; who tells me she loves me for my idealism, not despite it; and who gives me the space to chase abstract problems by keeping real ones at bay. This is for you.

[1] Mr. Dwivedi always says: "यह मैथ्स क्या है? कुछ भी तो नहीं!"

BIOGRAPHY

On a December morning in 1992, as the Babri Masjid fell and riots rippled across cities in India, a baby in Ghaziabad arrived with exactly one name: Arjun. No surname. In his revisionist retellings, twelve-year-old Arjun decided it was noble, modern, progressive, maybe all of the above - that his parents skipped the last name as a quiet act of secular defiance.

In boarding school, once, during roll call, Ms. Purnima Dutta (PDT) called out: ”Arjun?” ”Arjun what?” ”Just Arjun, ma’am.” ”Just Arjun? Fullstop?” She smirked. ”Fullstop” stuck for a few years – or so he liked to think – rather than the other, more colorful names his ever-changing group of bullies called him.

Years later, the US consulate in New Delhi stamped ”FNU” on his visa, which eventually rolled over into official documents, taking increasingly elaborate forms, ”No Name Given Arjun” being one.

”Arjun Arjun” didn’t bother him; ”fullstop” didn’t bother him; ”FNU” didn’t bother him. Nothing bothered him more than constantly wrangling with poorly implemented web form validation. So, in 2019, he decided it was time to adopt a last name. After years of not having one, Arjun Earthperson decided to cast the widest net, claiming shared lineage with every person on Earth.

ACKNOWLEDGEMENTS

I am grateful to the students in the PRA Group at NC State for their companionship and friendship. I thank Priyanka Pandit for working patiently with me on developing methods for building causal models of digital instrumentation & control failures, and for her grace in accepting my topic-switch so late into the program.

I thank Asmaa Farag, Egemen Aras, and Hasibul Rasheeq for tirelessly documenting their benchmarking, profiling, and optimization efforts for the solvers referenced throughout this work. Their extensive contributions allowed me to pick up where they left off.

I also thank Rawan Mostafa, Mostafa Hamza, and Yahya Alzahrani for helping me take the event tree inverse quantification problem from conceptualization to a working prototype, for using it in their work, and for providing feedback on its merits and limitations. Their candid feedback helped me think beyond methods and implementations.

In addition, I thank Mario Milev for going above and beyond in his administrative duties, for offering a sense of peace and calm, because I knew ”Mario is on top of it”.

Finally, I am deeply grateful to my dear friend Kirill Shumilov, who manifested, from nothing, the physical substrate on which we grounded our numerous, long, deep technical discussions, whimsies, and flights of the imagination. I could not have asked for a more enthusiastic, encouraging, analytical, and supportive work partner.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

AFM	Alpha Factor Model.
AFW	Auxiliary Feedwater.
AIG	And-Inverter Graph.
ANF/RNF	Algebraic/Ring Normal Form.
ANS	American Nuclear Society.
ARS	Advanced Reactor Safety.
ASME	American Society of Mechanical Engineers.
BCF	Blake Canonical Form.
BDD	Binary Decision Diagram.
BE	Basic Event.
BFM	Beta Factor Model.
BFR	Binomial Failure Rate Model.
BICS	Boolean Indicated Cut Sets.
BPM	Basic Parameter Model.
CAFTA	Computer Aided Fault Tree Analysis System.
CCCG	Common Cause Component Groups.
CCF	Common-Cause Failure.
CCW	Component Cooling Water.
CDF	Cumulative Distribution Function.
CI	Continuous Integration.
CLI	Command-Line Interface.
CLT	Central Limit Theorem.
CNF	Conjunctive Normal Form.

CPU	Central Processing Unit.
CRI	Criticality Importance Measure.
CuDD	Colorado University Decision Diagram Package.
DAG	Directed Acyclic Graph.
d-DNNF	Deterministic Decomposable Negation Normal Form.
DIF	Diagnostic Importance Factor.
DIM	Differential Importance Measure.
DNF	Disjunctive Normal Form.
DNNF	Decomposable Negation Normal Form.
d-NNF	Deterministic Negation Normal Form.
DPMC	Data-Parallel Monte Carlo.
DUCG	Dynamic Uncertain Causality Graph.
EIP	Essential Prime Implicant.
EPI	Essential Prime Implicate.
EPRI	Electric Power Research Institute.
ET	Event Tree.
f-BDD	Free/Read-Once Binary Decision Diagram.
FDS	Frontier Development of Science.
FE	Functional Event.
FHE	Fully Homomorphic Encryption.
f-NNF	Flat Negation Normal Form.
FPGA	Field-Programmable Gate Array.

FPRM	Fixed Polarity Reed-Muller.
FT	Fault Tree.
FTA	Fault Tree Analysis.
FTREX	Fault Tree Reliability Evaluation eXpert.
FV	Fussell–Vesely Importance Measure.
GB	Gigabyte.
GPL	GNU General Public License.
GPU	Graphics Processing Unit.
G–PWR	Generic Pressurized Water Reactor.
GUI	Graphical User Interface.
HCL	Hybrid Causal Logic.
HCLA	Hybrid Causal Logic Analyzer.
HE	Flag/House Event.
HPC	High Performance Computing.
HRA	Human Reliability Analysis.
i.i.d.	independent and identically distributed.
IMECE	International Mechanical Engineering Congress and Exposition.
INL	Idaho National Laboratory.
IP	Prime Implicant.
IRRAS	Integrated Reliability and Risk Analysis System.
ISLOCA	Interfacing Systems LOCA.

JIT	just-in-time.
JSCut	SAPHSOLVE MCS JSON.
JSInp	SAPHSOLVE Model Input JSON.
JSON	JavaScript Object Notation.
KAERI	Korea Atomic Energy Research Institute.
KIRAP	KAERI Integrated Reliability Analysis Code Package.
LCG	Linear Congruential Generator.
LERF	Large Early Release Frequency.
LLN	Law of Large Numbers.
LOCA	Loss of Cooling Accident.
LOOP	Loss of Offsite Power.
MAR–D	Models and Results Database.
MC	Monte Carlo.
MCS	Minimal Cut Sets.
MCUB	Min Cut Upper Bound.
MDP	Motor-Driven Pump.
MGL	Multiple Greek Letter Model.
MHTGR	Modular High Temperature Gas-cooled Reactor.
MICSUP	Minimal Cut Sets Upward.
MIT	Massachusetts Institute of Technology.
MOCUS	Method for Obtaining Minimal Cut Sets.

NASA	National Aeronautics and Space Administration.
NCSU	North Carolina State University.
NNF	Negation Normal Form.
NRC	US Nuclear Regulatory Commission.
OBDD	Ordered Binary Decision Diagram.
PC	Personal Computer.
PDAG	Probabilistic Directed Acyclic Graph.
PDF	Probability Density Function.
PGA	Peek Ground Acceleration.
PI	Prime Implicate.
PORV	Power-Operated Relief Valve.
PPRM	Positive Polarity Reed-Muller.
PRA	Probabilistic Risk Assessment.
PRNG	Pseudo Random Number Generator.
PSA	Probabilistic Safety Assessment.
PSDD	Probabilistic Sentential Decision Diagram.
PW	Prevention Worth.
PWR	Pressurized Water Reactor.
QRA	Quantitative Risk Assessment.
QRAS	Quantitative Risk Assessment System.
RAM	Random Access Memory.
RAW	Risk Achievement Worth.

REA	Rare-Event Approximation.
RegEx	Regular Expression.
RHR	Residual Heat Removal.
RNG	Random Number Generator.
RoBDD	Reduced Ordered Binary Decision Diagram.
RRW	Risk Reduction Worth.
SAPHIRE	Systems Analysis Programs for Hands-On Integrated Reliability Evaluations.
SAPHSOLVE	SAPHIRE Solve Engine.
SAT	Boolean Satisfiability.
SDD	Sentential Decision Diagram.
sd-DNNF	Smooth/Structured Deterministic Decomposable Negation Normal Form.
s-DNNF	Smooth/Structured Decomposable Negation Normal Form.
SIMD	Single Instruction, Multiple Data.
SIS	Safety Injection System.
SoP	Sum-of-Products/Sum-Product.
SP	Sum-Product/Sum-of-Products.
SPAR	Standardized Plant Analysis Risk.
SWS	Service Water System.
SYCL	Formerly known as SYstem-wide Compute Language.
UCLA	University of California Los Angeles.

UCLA GIRS	The B. John Garrick Institute for the Risk Sciences, UCLA.
US	United States.
VOT	Voter (K-of-N).
VRAM	Video/Graphics RAM.
WLAC	West Los Angeles College.
XAG	XOR-And-Inverter Graph.
XML	Extensible Markup Language.
XOR	Exclusive-Or.
ZDD	Zero-Suppressed Binary Decision Diagram.

Chapter 1

Introduction

1.1 Motivation

Probabilistic Risk Assessment (PRA) provides the quantitative substrate on which safety-critical decisions for nuclear installations are made. From the landmark WASH-1400 study through subsequent regulatory frameworks, its influence has grown in lock-step with the complexity of the engineered systems it seeks to evaluate [68]. Modern reactor models couple hundreds of event trees with thousands of fault trees, each containing many tens of thousands of basic events. The Boolean logic underpinning these structures is exacting—but it is also unforgiving: the number of minimal cut sets increases combinatorially with gate fan-in, and the inclusion–exclusion expansions required for exact probability calculations quickly exceed any realistic computational budget.

Historically, analysts have mitigated this hurdle through a hierarchy of approximations: truncating low-order terms, bounding probabilities with min-cut upper limits, or forcing the logic into syntactic fragments that are amenable to binary-decision diagrams. These techniques, implemented in widely deployed tools such as FTREX, SAPHIRE, SCRAM, XFTA, Phoenix Architect (formerly CAFTA), and RiskSpectrum, remain workhorses of industry practice. Yet each approximation trades rigor for tractability, often at the cost of conservatism or opacity – an increasingly unattractive compromise as regulatory emphasis shifts toward risk-informed, performance-based licensing.

A different path has opened in the last decade. The proliferation of many-core Graphics Processing Unit (GPU)s, coupled with mature programming models, enables billions of independent arithmetic operations per second on commodity hardware. Monte Carlo (MC) simulation is naturally aligned with such architectures: because individual samples are statistically independent, they map cleanly onto the embarrassingly parallel compute model of modern accelerators. Sampling the global state of a PRA model therefore offers a conceptually simple alternative to symbolic enumeration, provided that three longstanding obstacles can be overcome: (i) efficient generation of high-quality random events at scale, (ii) evaluation of large Boolean circuits at hardware line-rates, and (iii) principled convergence diagnostics that certify the statistical quality of the resulting estimates.

This dissertation tackles those obstacles head-on. It introduces a data-parallel Monte Carlo framework that compresses system states into machine-word bit-vectors, evaluates entire layers of the PRA logic in a single pass, and embeds rigorous stopping rules that blend frequentist and Bayesian error metrics. By revisiting PRA quantification from a hardware-conscious perspective we aim not merely to *approximate* classical methods but to redefine the achievable trade-space between fidelity and turnaround time. The developments that follow build the theoretical, algorithmic and empirical foundations for that goal.

1.2 Scope & Objectives

This dissertation seeks to rethink quantitative risk assessment from a hardware-conscious, data-parallel perspective. Its specific objectives are:

O1. **Methodological foundation.** Devise a Monte-Carlo workflow that evaluates complete PRA models—including all inter-linked event trees and fault trees—without minimal-cut enumeration or logic simplifications.

O2. **Algorithmic and data-structure design.** Create compilation, storage, and kernel-evaluation strategies that exploit many-core GPUs, multi-core CPUs, and reconfigurable hardware while remaining portable across architectures.

O3. **Robust statistical machinery.** Extend the baseline sampler to handle domain-specific challenges—rare events, common-cause failures, and importance derivatives—and embed composite convergence diagnostics with provable error guarantees.

O4. **Empirical validation.** Benchmark the resulting framework on public PRA datasets, reporting reproducible metrics for accuracy, runtime, and memory footprint, and compare its fidelity and performance to established exact and approximate tools.

Although the primary test bed is nuclear safety analysis, the techniques generalize to any industry that models complex Boolean risk scenarios (e.g. aerospace, chemical processing, autonomous vehicles). Dynamic simulations and strongly correlated uncertainty models lie outside the immediate scope; however, the architecture is deliberately extensible so that such capabilities can be integrated in future work.

1.3 Key Contributions

The principal technical contributions are:

C1. **Unified, data-parallel Monte-Carlo framework.** We design mcSCRAM, a hardware-accelerated sampler that evaluates the *entire* Probabilistic Directed Acyclic Graph (PDAG)—all coupled event trees and fault trees—in a single, layered pass without minimal-cut enumeration.

C2. **Hardware-native voting and cardinality gates.** We introduce bit-parallel algorithms for k-of-n, at-most, exact, and cardinality predicates, prove estimator equivalence to classical AND/OR expansions, and show exponential savings in graph size and kernel-launch overhead.

C3. **Knowledge compilation re-imagined for Monte Carlo queries.** We interrogate the design space of circuit transformations when the end-user query is high-throughput sampling rather than tractable logical inference. Dispensing with traditional normal-form constraints (e.g. decomposability and smoothness), we develop a pipeline that prioritizes arithmetic intensity, graph depth, and kernel locality; an indexed linear map and iterative gate normalization drive up faster compilation and a median 1.3× structural compression, illustrating the different optimality criteria that emerge once exact inference is no longer the goal.

C4. **Composite convergence diagnostics.** We formulate a stopping rule that fuses Wald, Bayesian, and information-theoretic criteria, providing rigorous error guarantees even in the rare-event regime and under external wall-time or iteration budgets.

C5. **Domain-specific extensions.** The framework incorporates: (i) common-cause failure groups through auxiliary/shadow nodes, (ii) first-order importance and sensitivity measures via in-tally covariance accumulation, and (iii) an importance-sampling module for ultra-rare events—all without modifying the core kernels.

C6. **Open-source implementation and benchmarks.** A SYCL-based reference implementation runs on GPUs, CPUs, and Field-Programmable Gate Array (FPGA)s; public benchmarks on the 43-model Aralia dataset demonstrate sub-percent relative error on graphs with a few thousand events in <5 s on entry-level GPUs, surpassing established PRA tools in throughput while matching their accuracy.

1.4 Outcomes & Related Documents

A core objective of this research program is open dissemination—code, data, and analysis tools are released under permissive licenses so that the wider PRA community can build upon them. Many of the outcomes listed below are the result of multi-institution collaborations and student projects; only a subset represent my direct, first-author contributions.

1.4.1 Software

The following repositories are maintained under my personal GitHub, and OpenPRA, open–source effort that now comprises over a dozen active contributors from academia and industry. Unless otherwise noted, my role has been to architect the core infrastructure and mentor fellow student developers; day-to-day development is a shared effort.

- **@a-earthperson/mcSCRAM**: A Data-Parallel Monte Carlo Probability Estimation Framework [30]. The central focus of this dissertation, implements the methods described in Part III and most of the refinements from Part IV. A fork of Olzhas Rakhimov's SCRAM [64].

- **@a-earthperson/inverse-canopy**: Tensorflow-based package for guessing conditional event probability distributions in an event tree to match target end-state frequencies. Implements the inverse problem described in Part V of this dissertation [29].

- **@a-earthperson/canopy-benchmarks**: Accuracy & performance benchmark results from MCSCRAM runs used in this dissertation [31].

- **@openpra-org/openpra-monorepo**: Mono repository for the OpenPRA web application [60]. Core code contributions for the distributed microservices supplementary to this dissertation are integrated here.

- **@openpra-org/model-benchmarks**: Benchmarking suite for SCRAM, XFTA, FTREX, SAPHSOLVE using benchexec in Docker [37]. Continuous Integration pipelines for automated runs & report generation are available.

- **@openpra-org/PRAcciolini**: Automates the conversion, validation, & translation of PRA models. Provides interfaces for describing grammars & translation rules between schema. [32].

- **@openpra-org/model-generator**: CLI utility for creating stochastically generated synthetic event trees & fault trees. Supports OpenPSA, SAPHSOLVE, FTREX and OpenFTA schema. [38].

- **@openpra-org/mef-schema**: Schema definitions for OpenPRA supported model exchange formats including FTREX FTP, SAPHSOLVE JSInp, JSCut, OpenPSA XMLs, & canopy flatbuffers [39].

- **@openpra-org/multi-hazard-model-generator**: Generates multi-hazard event trees & fault trees in MAR–D from internal events PRA models [19].

1.4.2 Datasets

All datasets are published on Zenodo with citable DOIs and were curated jointly with collaborators at North Carolina State University and Idaho National Laboratory.

- **Generic Pressurized Water Reactor (PWR) SAPHSOLVE Model**: Reference PWR model in SAPHSOLVE JSON (JSInp) format, supporting benchmarking & verification tasks [5].

- **Generic Pressurized Water Reactor (PWR) Open-PSA Model**: Equivalent PWR model in OpenPSA XML for cross-tool benchmark comparisons [4].

- **Generic Modular High Temperature Gas-cooled Reactor (MHTGR) Model**: Reference PRA model for a modular high temperature gas-cooled reactor, including event trees, fault trees, and basic event data in SAPHIRE and CAFTA formats [51].

- **Synthetic SAPHSOLVE Models**: Synthetically generated PRA models for benchmarking, quantification, & code verification [7].

- **Synthetic OpenPSA Models**: Synthetically generated OpenPSA XML format models for benchmark studies [6].

- **openpra-org/synthetic-models**: Centralized collection of stochastically generated PRA models in multiple supported formats, for validation & stress testing of quantification engines [12].

- **Benchmarking SAPHIRE, SCRAM & XFTA**: Dataset of synthetically generated fault trees with common cause failures, for head-to-head tool verification [34].

- **Generic OpenPSA Models: The Aralia Fault Tree Dataset**: Curated, large-scale synthetic PRA fault trees & event trees, compatible with the OpenPSA data model [35].

- **Post-processing Analysis & Supplementary Notes**: Excelsheets, plotting analysis, MATLAB scripts used for curating & analyzing the generated raw results. [33].

1.4.3 Journal Articles

- The Development, Implementation, and Application of a Probabilistic Risk Assessment Framework to Evaluate Supply Chain Shortages, Logistics, 2025 [63].

- A Systematic Diagnostics and Enhancement Framework for Advancing Probabilistic Risk Assessment Tools, Nuclear Technology, 2025 [15].

1.4.4 Conference Papers

2022

- Benchmark Study of XFTA and SCRAM Fault Tree Solvers Using Synthetically Generated Fault Trees Models, American Society of Mechanical Engineers (ASME) International Mechanical Engineering Congress and Exposition (IMECE) [16].

2023

- Introducing OpenPRA: A Web-Based Framework for Collaborative Probabilistic Risk Assessment, ASME IMECE [36].

- Model Exchange Methodology Between Probabilistic Risk Assessment Tools: SAPHIRE and CAFTA Case Study, American Nuclear Society (ANS) Probabilistic Safety Assessment (PSA) [50].

- Preliminary Benchmarking of SAPHSOLVE, XFTA, and SCRAM Using Synthetically Generated Fault Trees with Common Cause Failures, ANS PSA [43].

- Methodology and Demonstration for Performance Analysis of a Probabilistic Risk Assessment Quantification Engine: SCRAM, ANS PSA [11].

- Method of Developing a SCRAM Parallel Engine for Efficient Quantification of Probabilistic Risk Assessment Models, ANS PSA [10].

2024

- Advancing SAPHIRE: Transitioning from Legacy to State-of-Art Excellence, ANS Advanced Reactor Safety (ARS) [87].

- Evaluating PRA Tools for Accurate and Efficient Quantifications: A Follow-Up Benchmarking Study Including FTREX, ANS ARS [42].

- Towards a Deep-Learning based Heuristic for Optimal Variable Ordering in Binary Decision Diagrams to Support Fault Tree Analysis, ANS ARS [41].

- Enhancing the SAPHIRE Solve Engine: Initial Progress and Efforts, ANS ARS [8].

- Introducing OpenPRA's Quantification Engine: Exploring Capabilities, Recognizing Limitations, and Charting the Path to Enhancement, ANS ARS [9].

2025

- Automated OpenPSA Model Generation from Reliability Diagrams Using Agentic Retrieval Augmented Generation: A Case Study on MHTGR, ANS PSA [66].

- Design and Implementation of a Distributed Queueing System for OpenPRA, ANS PSA [67].

- Synthetical Model Generator for Probabilistic Risk Assessment Tools: Enhancing Testing, Verifying and Learning, ANS PSA [14].

- Facilitating PRA Model Accessibility: Model Converter Utility from SAPHIRE to Open-PSA, ANS PSA [13].

- Probability Estimation using Monte Carlo Simulation of Boolean Logic on Hardware-Accelerated Platforms, ANS PSA [40].

1.4.5 Theses & Dissertations

Graduate theses that have contributed to, or benefited from, the tooling or datasets developed during the conceptualization, design, or experimental stages of this dissertation.

- Asmaa Salem Amin Aly Farag, *Benchmarking Study of Probabilistic Risk Assessment Tools Using Synthetically Generated Fault Tree Models: SAPHSOLVE, XFTA, and SCRAM*, Master of Science, Department of Nuclear Engineering, NCSU, 2023 [45].

- Egemen Mutlu Aras, *Enhancement Methodology for Probabilistic Risk Assessment Tools through Diagnostics, Optimization, and Parallel Computing*, Doctor of Philosophy, Department of Nuclear Engineering, NCSU, 2024 [17].

- Hasibul Hossain Rasheeq, *[Working Title] Design and Implementation of a Distributed Queuing System for PRA Quantification*, Master of Science, Department of Nuclear Engineering, NCSU, Expected 2025 [65].

We now turn to the theoretical foundations underpinning the remainder of the work.

Part I

Foundations

Chapter 2

The Triplet Definition of Risk

A central goal of risk analysis in nuclear engineering is to enable sound decision-making under large uncertainties. To achieve this, risk must be defined in a way that is both rigorous and practically quantifiable. One widely accepted definition, tracing back to seminal work in [54, 48], frames risk as a *set of triplets*. Each triplet captures three essential dimensions:

1. *What can go wrong?*
2. *How likely is it to happen?*
3. *What are the consequences if it does happen?*

In more formal terms, let

$$R \;=\; \bigl\{\, \langle S_i,\, L_i,\, X_i \rangle \bigr\}_c \tag{2.1}$$

where R denotes the overall risk for a given system or activity, and the subscript c emphasizes *completeness*: ideally, all important scenarios must be included. In this notation:

- S_i specifies the ith **scenario**, describing something that can go wrong (e.g. an initiating event or equipment failure). Typically, $S_i \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible scenarios.

- L_i (sometimes denoted p_i or ν_i) is the **likelihood** (probability or frequency) associated with scenario S_i. In other words, $L_i \in [0, 1]$ if modeled as a probability, or $L_i \in [0, \infty)$ if modeled as a rate/frequency.

- X_i characterizes the **consequence**, i.e. the severity or nature of the outcome if the scenario occurs. Consequences can range from radiological releases and economic cost to broader societal impacts. In some analyses, X_i is a single-valued metric in $\mathcal{X}$; in others, it may be treated as a distribution over possible outcomes in $\mathcal{X}$.

The notation $\{\cdot\}_c$ in Eq. (2.1) stresses that *all* substantial risk scenarios must be included. Omitting a significant scenario might severely underestimate total risk. One might ask, "What are the uncertainties?" In this dissertation, uncertainties are embedded in each L_i (and sometimes X_i) via probability distributions.

2.1 A Scenario-Based Approach

A practical way to enumerate each triplet $\langle S_i, L_i, X_i \rangle$ is through logical decomposition of potential failures or disruptions, a process referred to as *scenario structuring.* Scenario structuring helps answer the question "*What can go wrong?*" in greater detail by dividing possible scenarios into commonly recognized classes. Each of these categories corresponds to a distinct family of *initiating events* (IE) that can trigger a chain of subsequent events or failures. At each node in the success scenario, we identify the IEs, which branch off from the initial success path S_0 into new pathways that may lead to undesirable states. Thus, each *scenario* S_i can be interpreted as a distinct departure from the baseline success path, triggered by some IE that occurs at node i. From that point onward, a sequence of *conditional events* or barriers may succeed or fail, culminating in an end-state ES_i.

2.2 Definition of an Event Tree

An Event Tree (ET) unravels how a single *initiating event* (I) can branch into multiple possible *end-states* (X) through a sequence of *functional* (or conditional) events. Each branch captures the success or failure of an important *functional event* (e.g. a safety barrier or operator intervention). By following all possible paths, one can systematically account for each final outcome X_j. Figure 2.1 provides a schematic view of this process for an initiating event I and two subsequent functional events, F_1 and F_2. Each terminal node (leaf) corresponds to a distinct end-state, denoted $X_1, X_2, \ldots, X_n$. Though this illustration is intentionally simple, more complex systems may include numerous functional events, each branching into further outcomes.

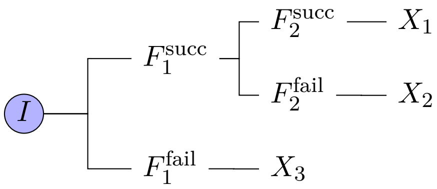


Figure 2.1: Illustrative event tree with an initiating event I, two functional events F_1 and F_2, and three end-states X_1, X_2, X_3.

At the highest conceptual level, an event tree is a collection of conditional outcomes. Let n be a positive integer, and let j range over some index set of end-states J. Then the scenario pathways can be defined as:

$$\Gamma = \Big\{ \langle I,\, F_1,\, F_2,\, \ldots,\, F_n,\, X_j \rangle : j \in J \Big\}, \tag{2.2}$$

where:

- I is the **initiating event**. In a nuclear system, this could be an abnormal occurrence such as a coolant pump trip or an unplanned reactivity insertion.

- F_k ($k = 1, \ldots, n$) denotes the kth **functional (conditional) event**, which may succeed (F_k^{succ}) or fail (F_k^{fail}). Typically, each F_k depends on the outcomes $F_1, \ldots, F_{k-1}$.
- X_j is an **end-state**, describing the final outcome along a particular branch. End-states might indicate safe shutdown, core damage, or a radiological release.

Each tuple $\langle I, F_1, \ldots, F_n, X_j \rangle$ in Γ encapsulates a distinct scenario pathway. In the broader context of the risk triplet, such a pathway corresponds to S_i, the possibility of something going wrong, while the associated probability and consequences map directly to L_i and X_i.

2.3 Definition of a Fault Tree

A Fault Tree (FT) is a top-down representation of how a specific high-level failure can arise from malfunctions in the components or subsystems of an engineered system. It is typically drawn as a tree or a Directed Acyclic Graph (DAG) whose unique root node is the top event and whose leaves/basic events capture individual component failures or other fundamental causes. This hierarchical decomposition proceeds until all relevant failure modes are captured in the leaves or else grouped as undeveloped events.

Formally, the nodes of a fault tree can be divided into two main categories:

- **Events**, which denote occurrences at different hierarchical levels.
  - A *Basic Event (BE)* represents the lowest-level failures, typically single-component malfunctions or individual human errors. They are often depicted as circles or diamonds in diagrams.
  - *Intermediate events* indicate the outcome of one or more lower-level events. Though intermediate events do not change the logical structure of the FT analysis, they can greatly enhance clarity by grouping sub-failures into a meaningful subsystem label (e.g., Cooling subsystem fails). They are typically drawn as rectangles.

  - *Top event* (TE) is a single node, unique in the tree, that represents the high-level failure of interest (e.g., System fails).

- **Gates**, which describe how events combine to produce a higher-level event. Each gate outputs a single event (often an intermediate or the top event), based on one or more input events.

Because a fault tree traces failures up toward the top event, the overall structure becomes a DAG. If a particular event (basic or intermediate) is relevant to multiple subsystems, it can be shared among the inputs of different gates. Consequently, while many small FTs have a pure tree shape, complex systems generally produce shared subtrees, yielding a more general DAG. If a system is large, detailed modeling of every component may not be warranted. In such cases, one may simplify certain subsystems by treating their failures as single *undeveloped events*. An undeveloped event is effectively a basic event for analysis purposes, even though it may internally comprise several components. This method conserves complexity where the subsystem is either of negligible importance or insufficiently characterized to break down further.

A convenient formalization, explained in detail in [72], treats an FT as a structure $F = \langle \mathcal{B}, \mathcal{G}, T, I \rangle$ where the unique top event t belongs to $\mathcal{G}$, and:

- $\mathcal{B}$ is the set of basic events.

- $\mathcal{G}$ is the set of gates or internal nodes.

- $T \rightarrow$ GateTypes: assigns a gate type (AND, OR, k/n, etc.) to each gate in $\mathcal{G}$.

- $I \rightarrow \mathcal{P}(\mathcal{B} \cup \mathcal{G})$: specifies the input set of each gate, i.e. which events (basic or intermediate) feed into that gate.

The graph is *acyclic* and has a unique root (the top event t) that is reachable from all other nodes. If an element is the input to multiple gates, it may be drawn once and connected multiple times or duplicated visually; either way, the logical semantics remain the same.

Interpreting a fault tree involves examining which higher-level events fail when a subset S of basic events has failed. Let $\pi_F(S, e)$ denote the failure state (0 or 1) of element e when the set $S \subseteq \mathcal{B}$ of basic events has failed. Then:

- For each basic event $b \in \mathcal{B}$:

$$\pi_F(S, b) \; = \; \begin{cases} 1, & b \in S, \\ 0, & b \notin S. \end{cases}$$

- For each gate $g \in \mathcal{G}$ with inputs $\{x_1, \ldots, x_k\} \subseteq \mathcal{B} \cup \mathcal{G}$:

$$\pi_F(S, g) \; = \; \begin{cases} \bigwedge_{i=1}^{k} \pi_F(S, x_i), & \text{if } T(g) = \text{AND}, \\ \bigvee_{i=1}^{k} \pi_F(S, x_i), & \text{if } T(g) = \text{OR}, \\ 1 \; - \; \pi_F(S, x_i), & \text{if } T(g) = \text{NOT (single input)}, \\ \sum_{i=1}^{k} \pi_F(S, x_i) \; \geq \; k, & \text{if } T(g) = \text{VOT}(k/n), \\ \Bigl(\sum_{i=1}^{k} \pi_F(S, x_i)\Bigr) \bmod 2, & \text{if } T(g) = \text{XOR}, \end{cases}$$

The top event t (i.e., TE) is a gate in $\mathcal{G}$; it is common to write simply $\pi_F(S)$ to denote whether the top event fails under the set S of failed BEs.

2.3.1 Definition of k-of-n Voting Logic

A *k-of-n* gate, denoted VOT(k/n), outputs 1 if and only if *at least* k of its n inputs equal 1. Such a gate, often referred to as a *threshold* or *majority voting* gate, conveniently models partial redundancy and majority-vote mechanisms. Concretely, let each of the n input events be represented by a binary variable:

$$X_1,\, X_2,\, \ldots,\, X_n \;\in\; \{0, 1\}.$$

The gate's output Y is then defined by

$$Y \;=\; \begin{cases} 1, & \text{if } \displaystyle\sum_{i=1}^{n} X_i \;\geq\; k, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

Equivalently, $Y = 1$ can be expressed as a disjunction of conjunctions:

$$Y \;=\; \Big[\sum_{i=1}^{n} X_i \;\geq\; k\Big] \;=\; \bigvee_{\substack{S \subseteq \{1,\ldots,n\} \\ |S|=k}} \Big(\bigwedge_{i \in S} X_i\Big), \tag{2.4}$$

meaning that at least one subset $S \subseteq \{1, \ldots, n\}$ of size k has all its corresponding X_i set to 1. Any larger subset $|S| > k$ naturally satisfies the same condition.

2.3.2 Common-Cause Failures

We consider a *Common Cause Component Groups (CCCG)* of size m composed of exchangeable components that share exposure, environment, and testing/maintenance policies. Two modeling regimes are distinguished and a consistent notation is adopted:

- **Demand-based models** (on-demand unavailability): probabilities are denoted $Q(\cdot)$.

- **Time-based models** (Poisson failure processes over a mission time): rates are denoted $\lambda(\cdot)$, with the small-time approximation $Q(\cdot) \approx \lambda(\cdot)\,\tau$ over mission time τ.

Within a CCCG we use tuple-specific common-cause parameters. For $k \in \{1, \ldots, m\}$, let

$$Q_k^{(m)} := \text{on-demand probability that a } \textit{specific } k\text{-subset fails due to a shared cause in a group of size } m,$$
$$Q_{G,k} := \binom{m}{k} Q_k^{(m)}, \qquad k = 1, \ldots, m,$$

where $Q_{G,k}$ is the on-demand probability that *some* (unspecified) k-subset fails (group-level multiplicity). The status of independent failures must be stated explicitly. Two conventions are common:

1. $Q_1^{(m)}$ includes single-component common-cause failures *and* independent failures; or
2. independent failures are modeled separately with a parameter Q_I (or λ_I in time-based models) and $Q_1^{(m)}$ represents only single-component CCFs.

We make this distinction clear below whenever it affects formulas.

2.3.2.1 Basic Parameter Model (BPM)

The Basic Parameter Model (BPM) assigns a parameter $Q_k^{(m)}$ to each multiplicity k in a group of size m. The per-component marginal on-demand failure probability Q_T (including whatever portion is placed in $Q_1^{(m)}$ under the chosen convention) satisfies

$$Q_T = \sum_{k=1}^{m} \binom{m-1}{k-1} Q_k^{(m)}. \tag{2.5}$$

This identity follows because a given component belongs to $\binom{m-1}{k-1}$ distinct k-subsets. The group-level multiplicity probabilities are

$$Q_{G,k} = \binom{m}{k} Q_k^{(m)}, \qquad k = 1, \ldots, m. \tag{2.6}$$

2.3.2.1.1 Remarks. BPM is saturated (one parameter per multiplicity) and can represent arbitrary exchangeable multiplicity structures subject to feasibility constraints (non-negativity and (2.5)). If independent failures are modeled separately by Q_I, then replace Q_T on the left of (2.5) by $Q_T^{(\mathrm{dep})} = Q_T - Q_I$ and set $Q_1^{(m)}$ to represent *dependent* single-component CCFs only.

2.3.2.2 Alpha Factor Model (AFM)

Under the standard (event-based) Alpha Factor Model (AFM), α_k denotes the fraction of common-cause *events* in the CCCG that involve exactly k components, so that

$$\alpha_k \geq 0, \qquad \sum_{k=1}^{m} \alpha_k = 1. \tag{2.7}$$

Let Q_T be the per-component on-demand failure probability (independent + dependent, per the convention in use). Under staggered testing, the canonical AFM→BPM mapping is

$$Q_k^{(m)} = \frac{\alpha_k}{\binom{m-1}{k-1}} Q_T, \qquad k = 1, \ldots, m. \tag{2.8}$$

Under non-staggered testing, the mapping includes a correction factor involving $\alpha_t := \sum_{i=1}^{m} i\, \alpha_i$ and becomes

$$Q_k^{(m)} = \frac{k\, \alpha_k}{\alpha_t \binom{m-1}{k-1}} Q_T, \qquad k = 1, \ldots, m. \tag{2.9}$$

A useful consequence of (2.8) is a "conservation by multiplicity" property: the per-component contribution of multiplicity k to Q_T equals $\alpha_k Q_T$, because $\binom{m-1}{k-1} Q_k^{(m)} = \alpha_k Q_T$. Thus the fraction of the per-component failure probability attributable to multiplicities ≥ 2 is $1 - \alpha_1$.

2.3.2.2.1 Estimation. With event counts n_k for multiplicity k, the empirical estimator is $\widehat{\alpha}_k = n_k / \sum_i n_i$. Bayesian estimation commonly adopts a Dirichlet prior $\alpha \sim \mathrm{Dir}(a_1, \ldots, a_m)$, yielding $\alpha \mid n \sim \mathrm{Dir}(a_1 + n_1, \ldots, a_m + n_m)$. When zeros occur at higher multiplicities, minimally informative or robust (imprecise) Dirichlet priors stabilize inference.

2.3.2.3 Multiple-Greek Letter (MGL) Model

To avoid symbol overload with the β-factor model, we parameterize the Multiple Greek Letter Model (MGL) cascade with $\rho_2, \rho_3, \ldots, \rho_m$, where

$$\rho_k = \Pr\big(\text{one additional component fails from the same cause} \mid \text{already } k{-}1 \text{ have failed}\big), \quad k = 2, \ldots, m.$$

This generative cascade implies an event multiplicity distribution $\pi = (\pi_1, \ldots, \pi_m)$:

$$\pi_1 = 1 - \rho_2, \qquad \pi_k = \Big(\prod_{j=2}^{k} \rho_j\Big)(1 - \rho_{k+1}) \ (2 \le k \le m-1), \qquad \pi_m = \prod_{j=2}^{m} \rho_j. \tag{2.10}$$

Under the event-based AFM convention we identify $\alpha_k = \pi_k$. Consequently, MGL→BPM follows from AFM→BPM via (2.8) (or (2.9) as applicable).

2.3.2.3.1 Inverse mapping (AFM→MGL).

Let $S_k := \sum_{j=1}^{k} \alpha_j$. From (2.10) one obtains

$$\rho_2 = 1 - \alpha_1, \qquad \rho_{k+1} = \frac{1 - S_k}{1 - S_{k-1}}, \quad 2 \le k \le m-1, \tag{2.11}$$

provided the denominators are non-zero (apply shrinkage when S_{k-1} is very close to 1).

2.3.2.3.2 Three-component example.

For $m = 3$,

$$\alpha_1 = 1 - \rho_2, \quad \alpha_2 = \rho_2(1 - \rho_3), \quad \alpha_3 = \rho_2\rho_3,$$

and, under staggered testing,

$$Q_1^{(3)} = \alpha_1 Q_T, \qquad Q_2^{(3)} = \frac{\alpha_2}{2} Q_T, \qquad Q_3^{(3)} = \frac{\alpha_3}{3} Q_T.$$

2.3.2.4 Binomial Failure Rate (BFR) Model

The Binomial Failure Rate Model (BFR) (Atwood) is time-based and posits two Poisson shock processes:

Lethal shocks at rate ω_L, which fail all m components when they occur.

Non-lethal shocks at rate ω_C. Conditional on such a shock, each component fails independently with probability $\rho \in [0, 1]$ (conditionally independent "hits").

Optionally include an independent per-component failure rate λ_I. Then:

$$\text{Group-level multiplicity event rates (non-lethal shocks):} \quad \lambda_{G,k}^{(\mathrm{nl})} = \omega_C \binom{m}{k} \rho^k (1-\rho)^{m-k}, \quad k = 0, 1, \ldots, m, \tag{2.12}$$

$$\text{Group-level lethal contribution (all-}m\text{-of-}m\text{):} \quad \lambda_{G,m}^{(\mathrm{lethal})} = \omega_L. \tag{2.13}$$

The per-component total rate is

$$\lambda_T = \lambda_I + \omega_L + \omega_C\, \rho. \tag{2.14}$$

2.3.2.4.1 Event-based α-factors under BFR. Consider only events that produce at least one failure. The total rate of such events is

$$\Lambda = \omega_L + \omega_C \big[1 - (1-\rho)^m\big].$$

Then the induced α-factors are

$$\alpha_k = \frac{\omega_C \binom{m}{k} \rho^k (1-\rho)^{m-k}}{\Lambda}, \quad 1 \le k \le m-1, \qquad \alpha_m = \frac{\omega_L + \omega_C \rho^m}{\Lambda}. \tag{2.15}$$

For demand-based analyses over a short mission τ, use $Q_T \approx \lambda_T\, \tau$ together with (2.15) and the AFM→BPM mapping.

2.3.2.5 Beta Factor Model

The β-factor Beta Factor Model (BFM) collapses all dependent mass into an m-of-m CCF. Let Q_T be the per-component on-demand failure probability. Then

$$\text{dependent all-}m\text{ failure:}\quad Q_m^{(m)} = \beta Q_T, \tag{2.16}$$

$$\text{independent remainder:}\quad (1-\beta)Q_T \text{ per component} \tag{2.17}$$

Accordingly, $\alpha_k = 0$ for $2 \le k \le m-1$, $\alpha_m = 1 - \alpha_1$, and the model is primarily useful for screening. It tends to be conservative for k-out-of-n structures with small k where mid-multiplicity failures dominate.

2.3.2.6 Mapping Between Models

Assume a fixed testing policy (staggered or non-staggered) and a declared convention for $Q_1^{(m)}$ vs. Q_I.

2.3.2.6.1 AFM → BPM.

$$\text{staggered:}\quad Q_k^{(m)} = \frac{\alpha_k}{\binom{m-1}{k-1}} Q_T, \tag{2.18}$$

$$\text{non-staggered:}\quad Q_k^{(m)} = \frac{k\,\alpha_k}{\alpha_t\binom{m-1}{k-1}} Q_T, \qquad \alpha_t = \sum_{i=1}^{m} i\,\alpha_i. \tag{2.19}$$

2.3.2.6.2 MGL ↔ AFM.

$$\text{MGL} \to \text{AFM:}\quad \alpha_k = \pi_k \text{ from (2.10)}, \tag{2.20}$$

$$\text{AFM} \to \text{MGL:}\quad \rho_2 = 1 - \alpha_1, \quad \rho_{k+1} = \frac{1 - \sum_{j=1}^{k} \alpha_j}{1 - \sum_{j=1}^{k-1} \alpha_j},\ 2 \le k \le m-1. \tag{2.21}$$

2.3.2.6.3 BFR → AFM. Compute α_k via (2.15). Then AFM→BPM by (2.18) or (2.19).

2.3.2.6.4 AFM → β. The fraction of the per-component probability due to dependent failures is

$$\beta \approx \sum_{k=2}^{m} \alpha_k = 1 - \alpha_1,$$

if one collapses all dependent mass into an m-of-m event as in (**??**).

2.3.2.6.5 Well-posedness checks. Any proposed parameter set should satisfy non-negativity, normalization (for α or the induced multiplicity distribution), and feasibility in (2.5). When converting between parameterizations, apply shrinkage for zero counts and verify that induced $Q_k^{(m)}$ are non-negative and that the implied Q_T (or λ_T) matches the target within numerical tolerance.

Industry databases such as the International Common Cause Failure Data Exchange (ICDE) [1] and the NRC's CCF database [58] provide valuable sources for parameter estimation.

2.4 Quantifying Event Sequences

Because risk analysis requires knowing how likely each branch in the tree is, event trees rely heavily on *conditional probabilities.* Let

$$p(I) \;\equiv\; \Pr(I)$$

be the probability (or frequency) of the initiating event. For each functional event F_k, define

$$p\big(F_k^{\text{succ}} \mid I,\, F_1,\, \ldots,\, F_{k-1}\big) \quad \text{and} \quad p\big(F_k^{\text{fail}} \mid I,\, F_1,\, \ldots,\, F_{k-1}\big),$$

which describe the likelihood of success or failure given all prior outcomes.

An *end-state* X_j arises from a particular chain of successes/failures:

$$\big(I,\, F_1^{\alpha_1},\, F_2^{\alpha_2},\, \ldots,\, F_n^{\alpha_n}\big) \;\longrightarrow\; X_j,$$

where each $\alpha_k \in \{\text{succ, fail}\}$. The probability of reaching X_j is the product of:

1. The initiating event probability $p(I)$.

2. The conditional probabilities of each functional event's success or failure.

Formally, if ω_j denotes the entire branch leading to end-state X_j, then

$$p(\omega_j) \;=\; p(I) \times \prod_{k=1}^{n} p\big(F_k^{\alpha_k} \mid I,\, F_1^{\alpha_1}, \ldots, F_{k-1}^{\alpha_{k-1}}\big). \tag{2.22}$$

Equation 2.22 shows that an event tree can be represented by a product (logical AND) of the relevant Boolean variables for the initiating event and each functional event's success/failure. The union of all the branches spans the full sample space of scenario outcomes generated by I and the subordinate functional events. Collecting all branches via logical OR in this way yields a disjunction of these products, precisely matching the standard structure of a Boolean expression in Disjunctive Normal Form (DNF). This idea is explored further in Section 4.2.3.

2.4.1 Exact Methods for Calculating Probabilities

Beyond describing which combinations of basic events can fail, most Fault Tree Analysis (FTA) requires a quantitative assessment of the *likelihood* that the top event ultimately occurs. This section details how probabilities are embedded within the fault tree structure, how to compute the top event's failure probability (or system *unreliability*), and some common methods for handling large or dependent fault trees.

2.4.1.1 Assigning Probabilities to Basic Events

Let $\mathcal{B} = \{b_1, \ldots, b_n\}$ be the set of basic events in the fault tree F. Each basic event b is associated with a *failure probability* $p(b) \in [0, 1]$. Interpreted as a Boolean random variable X_b, event b takes value 1 (failure) with probability $p(b)$ and value 0 (success) with probability $1 - p(b)$. Thus,

$$\Pr\bigl[X_b = 1\bigr] \;=\; p(b), \quad \Pr\bigl[X_b = 0\bigr] \;=\; 1 - p(b).$$

In the simplest *single-time* analysis, each basic event is either failed or functioning for the entire time horizon under study, and no component recovers once it has failed.

2.4.1.2 Event Probability Under Independence

If we assume that all basic events fail independently, then for any set $S \subseteq \mathcal{B}$ of failed basic events,

$$\Pr\bigl[\text{basic events in } S \text{ fail and others succeed}\bigr] \;=\; \prod_{b \in S} p(b) \;\times \prod_{b \notin S} [1 - p(b)].$$

Recall from Section 2.3 that the top event T (also called TE) fails given S precisely if $\pi_F(S, T) = 1$. Hence, the probability that the top event fails is the sum of these independent configurations S for which $\pi_F(S, T) = 1$:

$$\Pr\bigl[X_t = 1\bigr] \;=\; \sum_{S \subseteq \mathcal{B}} \Bigl[\pi_F(S, T) \prod_{b \in S} p(b) \prod_{b \notin S} \bigl[1 - p(b)\bigr]\Bigr]. \tag{2.23}$$

A direct computation of (2.23) often becomes unwieldy for large FTs because of the exponential number of subsets S. However, if the fault tree is *simple* (no shared subtrees) and each gate in $\mathcal{G}$ has independent inputs, one may propagate probabilities *bottom-up* through the DAG using basic probability rules:

$$\begin{aligned}
&\Pr[g=1] \;=\; \prod_{x\in I(g)} \Pr[x=1], && \text{if gate } g \text{ is AND,}\\
&\Pr[g=1] \;=1\;-\; \prod_{x\in I(g)} \bigl[1-\Pr[x=1]\bigr], && \text{if gate } g \text{ is OR,}\\
&\Pr[g=1] \;=1\;-\; \Pr[x=1], && \text{if gate } g \text{ is NOT (single input } x\text{),}\\
&\Pr[g=1] \;=\; \sum_{j=k}^{|I(g)|} \sum_{\substack{A\subseteq I(g)\\ |A|=j}} \prod_{x\in A}\Pr[x=1] \prod_{x\in I(g)\setminus A} \bigl[1-\Pr[x=1]\bigr], && \text{if gate } g \text{ is VOT}(k/n),\\
&\Pr[g=1] \;=\; \sum_{\substack{A\subseteq I(g)\\ |A| \text{ is odd}}} \prod_{x\in A}\Pr[x=1] \prod_{x\in I(g)\setminus A} \bigl[1-\Pr[x=1]\bigr], && \text{if gate } g \text{ is XOR.}
\end{aligned}$$

2.4.1.3 Inclusion-Exclusion: Probability Computation under Independence

When each X_i is modeled as a Bernoulli random variable taking the value 1 with probability

$$p_i \;=\; P(X_i=1),$$

assumed independent of the other inputs, one can write for each k-element subset $S \subseteq \{1,\ldots,n\}$:

$$A_S \;=\; \bigwedge_{i\in S}\{X_i=1\}, \quad P\bigl(A_S\bigr) \;=\; \prod_{i\in S} p_i.$$

The event $Y=1$ in Eq. (2.4) is the union of all such events A_S for $|S|=k$. Hence, in principle, to compute

$$P\Bigl(Y=1\Bigr) \;=\; P\Bigl(\bigvee_{|S|=k} A_S\Bigr),$$

one may apply the inclusion-exclusion principle. Specifically:

$$P\Bigl(\bigvee_{|S|=k} A_S\Bigr) \;=\; \sum_{|S|=k} P(A_S) \;-\; \sum_{1\le\alpha_1<\alpha_2\le M} P\Bigl(A_{S_{\alpha_1}} \cap A_{S_{\alpha_2}}\Bigr) \;+\; \ldots \tag{2.24}$$

where $M = \sum_{j=k}^{n} \binom{n}{j}$ is the total number of events A_S of interest. Unfortunately, direct enumeration of these intersections can become infeasible for large n.

2.4.1.3.1 Worst-Case Example using 3-of-5 Voting Logic In many PRA tools, the VOT(k/n) gate is provided as a built-in element. Internally, these tools may expand Eq. (2.4) into an OR-of-ANDs or maintain a more compact representation. Under independence assumptions, standard failure-probability calculations proceed by summing over combinations of basic-event failures. However, an explicit expansion incurs the combinatorial factor

$$\binom{n}{k} + \binom{n}{k+1} + \ldots + \binom{n}{n},$$

which can grow quickly as n increases. For example, consider the case where $k = 3, n = 5$. Using the notation from Section 2.3.1, the compressed form of the gate output is

$$Y = \text{VOT}(3/5)\big(X_1, X_2, X_3, X_4, X_5\big) = \big[\, X_1 + X_2 + X_3 + X_4 + X_5 \geq 3\big].$$

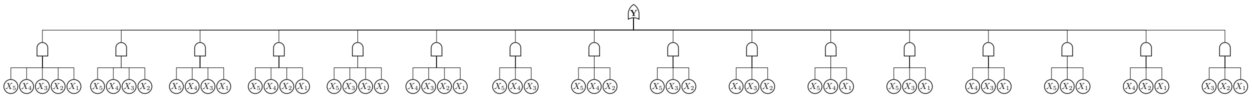


Figure 2.2: 3-of-5 voting logic, expanded as (AND-OR) Disjunctive Normal Form (DNF)

Since "at least 3 of 5" means any subset of size 3, 4, or 5 suffices, an explicit OR-of-ANDs expansion has the following terms:

$$\begin{aligned} Y \;=\; & \underbrace{X_1X_2X_3 \vee X_1X_2X_4 \vee X_1X_2X_5 \vee X_1X_3X_4 \vee X_1X_3X_5 \vee X_1X_4X_5 \vee X_2X_3X_4 \vee X_2X_3X_5 \vee X_2X_4X_5 \vee X_3X_4X_5}_{\text{subsets of size 3}} \\ & \vee\; \underbrace{X_1X_2X_3X_4 \vee X_1X_2X_3X_5 \vee X_1X_2X_4X_5 \vee X_1X_3X_4X_5 \vee X_2X_3X_4X_5}_{\text{subsets of size 4}} \\ & \vee\; \underbrace{X_1X_2X_3X_4X_5}_{\text{subset of size 5}}. \end{aligned} \tag{2.25}$$

Assume that each X_i is a Bernoulli random variable taking the value 1 with probability p_i, independently of the others. Then

$$p_i \;=\; P\big(X_i = 1\big), \quad i = 1, 2, 3, 4, 5.$$

We seek $P(Y = 1)$, i.e., the probability that at least three of the X_i are 1. A convenient way to write this is to sum, for $j = 3, 4, 5$, the probabilities that exactly j of the inputs are 1:

$$P\big(Y = 1\big) \;=\; \sum_{j=3}^{5} \sum_{\substack{S \subseteq \{1,2,3,4,5\} \\ |S|=j}} \prod_{i \in S} p_i \prod_{i \notin S} (1 - p_i).$$

Exactly three of the five inputs are 1 (there are $\binom{5}{3} = 10$ such terms):

$$\begin{aligned} & p_1p_2p_3\,(1-p_4)\,(1-p_5) \;+\; p_1p_2p_4\,(1-p_3)\,(1-p_5) \;+\; p_1p_2p_5\,(1-p_3)\,(1-p_4) \;+ \\ & p_1p_3p_4\,(1-p_2)\,(1-p_5) \;+\; p_1p_3p_5\,(1-p_2)\,(1-p_4) \;+\; p_1p_4p_5\,(1-p_2)\,(1-p_3) \;+ \\ & p_2p_3p_4\,(1-p_1)\,(1-p_5) \;+\; p_2p_3p_5\,(1-p_1)\,(1-p_4) \;+\; p_2p_4p_5\,(1-p_1)\,(1-p_3) \;+ \\ & p_3p_4p_5\,(1-p_1)\,(1-p_2). \end{aligned}$$

Exactly four of the five inputs are 1 (there are $\binom{5}{4} = 5$ such terms):

$$p_1p_2p_3p_4\,(1-p_5) \;+\; p_1p_2p_3p_5\,(1-p_4) \;+\; p_1p_2p_4p_5\,(1-p_3)$$
$$+\; p_1p_3p_4p_5\,(1-p_2) \;+\; p_2p_3p_4p_5\,(1-p_1).$$

All five inputs are 1 (there is $\binom{5}{5} = 1$ such term):

$$p_1p_2p_3p_4p_5.$$

Summing all of the above terms yields $P(Y = 1)$, the fully expanded probability of a 3-of-5 vote under independence.

2.4.1.3.2 Complexity of the Inclusion-Exclusion Approach Observe that the union in Eq. (2.24) comprises M events, where

$$M \;=\; \sum_{j=k}^{n} \binom{n}{j}.$$

The inclusion-exclusion principle for a union of M events involves sums of intersections of size r, for $r = 1, \ldots, M$. The total number of terms is

$$\sum_{r=1}^{M} \binom{M}{r} \;=\; 2^M - 1,$$

which implies a worst-case computational complexity on the order of

$$\mathcal{O}\big(2^M\big).$$

Moreover, for $k \approx n/2$, the binomial coefficients $\binom{n}{k}$ become largest, so M can itself grow exponentially in n. Consequently, the inclusion-exclusion expansion may require up to $\mathcal{O}\big(2^{2^n}\big)$ operations for moderately large n.

2.4.2 Methods for Approximating Probabilities

2.4.2.1 The Rare-Event Approximation

When each basic event (b) has a sufficiently small failure probability (p(b)), the likelihood of multiple minimal cut sets failing simultaneously is often negligible. Under these conditions, one may approximate the failure probability of the top event by treating each Minimal Cut Sets as though it fails independently. Specifically, let $\mathrm{MCS}(T)$ denote the set of all minimal cut sets that either have cardinality up to some *truncation value* (T) or exceed a chosen probability threshold. Then one replaces the exact summation over all MCS by summing only over $\mathrm{MCS}(T)$, yielding

$$\Pr\big[X_t = 1\big] \approx \sum_{C \in \mathrm{MCS}(T)} \prod_{b \in C} p(b). \tag{2.26}$$

This approach is justified in highly reliable systems where multi-cut-set failures have low probability. Moreover, it reduces the computational burden by screening out higher-order or low-probability minimal cut sets. The choice of (T) typically adheres to engineering practice: for example, in a system designed to tolerate any single failure (often referred to as ($N-1$) redundancy), all minimal cut sets of size up to ($N-1$) might be considered while larger cut sets, deemed improbable, are excluded.

2.4.2.1.1 Error Bound for Truncated Approximations A natural way to measure the quality of a truncated approximation is by comparing it to the exact probability of top-event failure. Denote the exact probability by

$$\Pr\big[X_t = 1\big],$$

and the truncated approximation, which sums only over the minimal cut sets in MCS(T), by

$$\Pr_{T}\big[X_t = 1\big] = \sum_{C \in \mathrm{MCS}(T)} \prod_{b \in C} p(b).$$

Then the *error* associated with truncation up to (T) is

$$\Delta(T) = \Pr\big[X_t = 1\big] - \Pr_{T}\big[X_t = 1\big]. \tag{2.27}$$

Under the assumption that failures are sufficiently rare and interactions among higher-order minimal cut sets are negligible, an *upper bound* for this error may be obtained by summing the omitted terms:

$$\big|\Delta(T)\big| \leq \sum_{C \in \mathrm{MCS} \setminus \mathrm{MCS}(T)} \prod_{b \in C} p(b). \tag{2.28}$$

In practice, computing $\Delta(T)$ or its bound usually requires identifying and evaluating all minimal cut sets outside MCS(T), which may still be tractable if the omitted sets are large, unlikely, or both. Consequently, choosing a suitable truncation parameter, T (by size or probability threshold), ensures that the unaccounted failure modes contribute negligibly to the overall system unreliability.

2.4.2.2 The Min-Cut Upper Bound

Another method for bounding the probability of a top event interprets system failure as the union of all minimal cut set (MCS) failures. The most direct upper bound is obtained by applying the union bound (Boole's inequality), which states that the probability of the union of events is no greater than the sum of their individual probabilities. For a set of minimal cut sets $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, where each C_i is a set of basic events, the union bound yields:

$$\Pr\left[X_t = 1\right] \leq \sum_{C \in \mathrm{MCS}} \prod_{b \in C} p(b), \tag{2.29}$$

where $p(b)$ is the probability of basic event b.

However, most modern PRA tools, including SAPHIRE [79], implement a slightly tighter upper bound, known as the *minimal cut set upper bound*, which is given by:

$$\Pr\left[X_t = 1\right] \leq 1 - \prod_{C \in \mathrm{MCS}} \left[1 - \prod_{b \in C} p(b)\right]. \tag{2.30}$$

This expression accounts for the fact that the system fails if *any* cut set fails, and thus computes the probability that at least one cut set occurs, assuming independence between cut sets. It is always at least as large as the true probability, and is generally less conservative than the simple union bound, especially when cut sets overlap.

The min-cut upper bound is exact if and only if all minimal cut sets are mutually disjoint (i.e., share no basic events). In practice, cut sets often overlap, leading to double counting of shared failure modes and thus a conservative overestimate. This bound remains valid regardless of the magnitude of the basic event probabilities, unlike the rare-event approximation, which assumes that cut set failures are nearly disjoint and probabilities are small.

For illustration, consider a fault tree with three minimal cut sets [77]: $C_1 = \{\mathrm{A}, \mathrm{B}\}$, $C_2 = \{\mathrm{B}, \mathrm{C}\}$, and $C_3 = \{\mathrm{D}\}$, with $p(\mathrm{A}) = p(\mathrm{B}) = p(\mathrm{C}) = 0.7$ and $p(\mathrm{D}) = 0.5$. The min-cut upper bound is:

$$\begin{aligned}
\Pr(X) &\leq 1 - \left[1 - p(\mathrm{A})p(\mathrm{B})\right]\left[1 - p(\mathrm{B})p(\mathrm{C})\right]\left[1 - p(\mathrm{D})\right] && (2.31)\\
&= 1 - (1 - 0.49)(1 - 0.49)(1 - 0.5) && (2.32)\\
&= 1 - (0.51)(0.51)(0.5) && (2.33)\\
&= 1 - 0.13005 = 0.86995. && (2.34)
\end{aligned}$$

This value is a conservative estimate of the true top event probability.

2.4.3 Probability Estimation using Monte Carlo Sampling

Monte Carlo methods provide a versatile framework for approximating expectations, probabilities, and other quantities of interest by simulating random observations from an underlying distribution. At its core, a Monte Carlo estimator uses repeated random draws to approximate quantities such as

$$\mathbb{E}[f(X)] \;=\; \int f(x)\, p(x)\, \mathrm{d}x \;\;\approx\;\; \frac{1}{N}\sum_{i=1}^{N} f\big(x^{(i)}\big), \tag{2.35}$$

where $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$ are independent and identically distributed (i.i.d.) samples drawn from p. The function f is a measurable function of the random variable X. In reliability and PRA contexts, f might be an indicator of a particular event (e.g., a system failure), in which case $\mathbb{E}[f(X)]$ becomes the probability of that event.

2.4.3.1 Convergence and the Law of Large Numbers

A central theoretical result underpinning Monte Carlo sampling is the Law of Large Numbers (LLN). In one of its classical forms, the Strong LLN states:

Theorem 1 (Strong Law of Large Numbers)**.** *Let $X_1, X_2, \ldots$ be a sequence of i.i.d. random variables with finite expectation $\mathbb{E}[X_1]$. Then, with probability 1,*

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} X_i \;=\; \mathbb{E}[X_1].$$

Applied to the sample estimator in Eq. (2.35), the LLN implies that as the number of samples N grows large, the average of the function values $f\big(x^{(i)}\big)$ converges to $\mathbb{E}[f(X)]$. Thus, by simply drawing enough samples, one can approximate probabilities or expectations arbitrarily well (with probability 1).

2.4.3.2 Central Limit Theorem and Error Analysis

Another classical result is the *Central Limit Theorem (CLT)*, which indicates that the Monte Carlo estimator's distribution (around its true mean) approaches a normal distribution for large N. Specifically,

Theorem 2 (Central Limit Theorem)**.** *Suppose $X_1, X_2, \ldots$ are i.i.d. random variables with mean $\mu = \mathbb{E}[X_1]$ and variance $\sigma^2 = \mathbb{V}[X_1] < \infty$. Then the sample mean satisfies*

$$\sqrt{N}\left(\frac{1}{N}\sum_{i=1}^{N} X_i - \mu\right) \ \overset{\mathrm{d}}{\to}\ \mathcal{N}(0, \sigma^2),$$

where $\overset{\mathrm{d}}{\to}$ denotes convergence in distribution.

In practical terms, the Central Limit Theorem (CLT) implies that for sufficiently large N, the sampling fluctuations of the Monte Carlo estimator around the true mean are approximately normal. The variance of this normal distribution decreases with $1/N$. Therefore, one can estimate confidence intervals, standard errors, and convergence rates by tracking empirical variance across the sample.

The above principles remain valid even when f is an indicator of a Boolean event or a composite system failure embedded in an event/fault tree. One need only be able to draw samples $\big(x^{(i)}\big)$ from the system's joint distribution over basic events (or from any suitable representation of the PRA model) and then evaluate the function f to determine system success/failure for each sample. Subsequent chapters will expand on how these samples can be generated for event trees, fault trees, or more complex DAG-based representations.

2.4.3.3 Generating Random Numbers

Monte Carlo estimators rely on the ability to generate random realizations from a given distribution. Computers, however, do not typically provide true randomness; instead, they use Pseudo Random Number Generator (PRNG)s to produce sequences of numbers that mimic realizations from a uniform distribution on $[0, 1]$. From these *uniform* samples, one can then derive samples from more general distributions using various transformations (e.g., the *inverse transform* method, acceptance-rejection, composition methods, or specialized sampling algorithms).

A PRNG is formally a deterministic function that, given an initial *seed*, generates a long sequence of values in $(0, 1)$. Popular choices include:

- *Linear Congruential Generator (LCG)s*, which use a recurrence of the form

  $$X_{n+1} \; = \; (a\, X_n + c) \mod m,$$

  then normalize $\frac{X_{n+1}}{m}$ to produce a pseudo-random variate in $(0, 1)$.

- *Mersenne Twister*, which generates high-quality pseudo-random numbers with a very long period (e.g., $2^{19937} - 1$).

- *Philox* or other counter-based methods that deliver high performance and reproducible streams across parallel computations.

While these methods provide deterministic sequences, strong design ensures that the resulting outputs pass numerous statistical tests for randomness. If the seed is chosen randomly (or from a secure source), these methods can approximate uniformity closely enough for most Monte Carlo studies.

2.4.3.3.1 Random Variates via Transformations Given access to uniform samples $U \sim \text{Unif}(0, 1)$, one can construct samples from many other distributions. Two widely used techniques are:

1. **Inverse Transform Sampling:** Suppose a continuous variable X has Cumulative Distribution Function (CDF) $F_X(x)$. If $U \sim \text{Unif}(0, 1)$, then $X = F_X^{-1}(U)$ follows the

same distribution as X. More precisely,

$$P\big[X \le x\big] \;=\; P\big[F_X^{-1}(U) \le x\big] \;=\; P\big[U \le F_X(x)\big] \;=\; F_X(x),$$

provided F_X is continuous and strictly increasing.

2. **Acceptance-Rejection:** For certain distributions where the inverse CDF is not straightforward, one can sample from an easier *proposal distribution* $q(x)$ that bounds the targeted density $p(x)$. Specifically, if $p(x) \le M\, q(x)$ for all x, then:

   (a) Draw $Y \sim q(\cdot)$ and $Z \sim \mathrm{Unif}(0,1)$.

   (b) Accept Y if $Z \le \frac{p(Y)}{M\, q(Y)}$. Otherwise, reject and repeat.

   The accepted sample Y follows distribution $p(x)$.

2.4.3.3.2 Boolean Events as Discrete Random Variables Many variables are *discrete*, often Bernoulli (success/failure) or categorical (e.g. multiple failure modes). Generating $\{0,1\}$-valued samples is then straightforward, since for each basic event b,

$$\Pr[b=1] \;=\; p(b), \quad \Pr[b=0] \;=\; 1-p(b).$$

Given a uniform variate U, one sets

$$b \;=\; \begin{cases} 1, & U \le p(b), \\ 0, & \text{otherwise.} \end{cases}$$

This approach naturally extends to multi-categorical events. More complex dependencies among events can also be captured by specifying appropriate conditional distributions.

2.4.3.3.3 Extending Boolean Events to Continuous Random Variables A *continuous* random variable Y has a Probability Density Function (PDF) $f_Y(y)$ on a continuous domain $\mathcal{Y} \subseteq \mathbb{R}$. Common examples in reliability include:

- **Exponential Distribution**, often used to model times to failure under a constant hazard rate λ. Its PDF is

$$f_Y(y) \;=\; \lambda\, e^{-\lambda y}, \quad y \geq 0.$$

- **Weibull Distribution**, with flexible shape parameter $\beta > 0$ and scale parameter $\alpha > 0$. Its PDF is

$$f_Y(y) \;=\; \frac{\beta}{\alpha}\Bigl(\frac{y}{\alpha}\Bigr)^{\beta-1} \exp\Bigl[-\bigl(y/\alpha\bigr)^{\beta}\Bigr], \quad y \geq 0.$$

- **Lognormal Distribution**, where $\log(Y)$ follows a normal distribution. This is sometimes employed for components whose lifetimes span multiple orders of magnitude.

Continuous random variables typically arise when modeling the *time dimension*: for instance, the time until a valve sticks closed, or the moment when a pipe experiences a critical crack. One can then generate a Bernoulli indicator for whether the failure has occurred by time t using

$$\Pr[Y \leq t] \;=\; \int_0^t f_Y(y)\,\mathrm{d}y \;=\; F_Y(t),$$

where F_Y is the Cumulative Distribution Function (CDF) of Y. Evaluating this probability at each Monte Carlo trial and comparing against a uniform random variate yields a discrete failure indicator. Hence, continuous distributions can be mapped to discrete states at any chosen time horizon.

2.5 Generating Cut Sets and Implicants

Once the fault tree or event tree model has been constructed, the *qualitative* phase of risk characterization begins. The objective is to identify unique combinations of basic events that are of interest because they either *cause* the top event (system failure) or *guarantee* its prevention (system success). These combinations are formalized by the logic concepts defined below.

Implicants are general combinations of events (either failure or success) that satisfy the top event. *Cut sets* are specialized implicants containing only failure events that cause system failure. *Path sets* are specialized implicants of the complement function ($\overline{T}$) containing only success events that ensure system operation. *Minimal cut sets* are prime implicants containing only failure events, representing the minimal ways the system can fail. *Maximal path sets* are prime implicants of $\overline{T}$ containing only success events, representing the maximal ways the system can operate successfully.

2.5.1 Implicants

Let $T(\mathbf{x})$ be the Boolean top event function of the model and $I \subseteq \{x_1, \ldots, x_n\}$ be a set of basic events. A set I is an *implicant* of T if

$$T(\mathbf{1}_I) = 1$$

where $\mathbf{1}_I$ is the truth vector that sets the basic events in I to TRUE. Implicants can contain both failure and success events depending on the analysis context.

- *Cut set*: An implicant containing only failure events.
- *Path set*: The complement of an implicant containing only success events.

2.5.1.1 Prime Implicants

A *prime* implicant P is an implicant that is not a subset of any other implicant. Formally, P is a prime implicant if:

1. $T(\mathbf{1}_P) = 1$

2. $\forall P' \subset P, T(\mathbf{1}_{P'}) = 0$

2.5.1.2 Essential Prime Implicants

An *essential* prime implicant π is a prime implicant that contains at least one minterm that is not covered by any other prime implicant, i.e., there exists at least one basic event $\mathbf{a}$ such that $T(\mathbf{a}) = 1$ and $\mathbf{a}$ satisfies π but does not satisfy any other prime implicant of T. This property guarantees that π must appear in every irredundant disjunctive normal form representation of T.

2.5.2 Path Sets

A *path set* P is a set of success events such that when all events in P occur, the system operates successfully. If $\overline{T}$ represents the complement of the top event function (system success), then:

$$\overline{T}(\mathbf{1}_P) = 1 \text{ where } P \text{ contains only success events}$$

2.5.2.1 Maximal Path Sets

A *maximal path set* is a path set to which no basic event can be added while still ensuring system success. A path set X is maximal if:

1. $\overline{T}(\mathbf{1}_X) = 1$

2. $\forall x \notin X, \overline{T}(\mathbf{1}_{X \cup \{x\}}) = 0$

Maximal path sets represent the largest combinations of component successes that guarantee system operation.

2.5.3 Cut Sets

A *cut set* C is an implicant containing only failure events such that when all events in C occur, the system fails. Formally, for a Boolean function T representing the top event:

$$T(\mathbf{1}_C) = 1 \text{ where } C \text{ contains only failure events}$$

2.5.3.1 Minimal Cut Sets

A *minimal cut set* is a cut set which causes the failure of the system, but when a basic event is removed from the cut set, it does not cause system failure anymore [3]. A cut set M is minimal if:

1. $T(\mathbf{1}_M) = 1$
2. $\forall M' \subset M, T(\mathbf{1}_{M'}) = 0$

Minimal cut sets represent the smallest combinations of component failures that can cause system failure.

2.5.3.2 Method for Obtaining Minimal Cut Sets (MOCUS)

Originally proposed by Fussell and Vesely in 1974 [47], the MOCUS algorithm remains one of the most widely deployed techniques for top-down generation of minimal cut sets in PRA tools. The procedure starts from the TOP event of a fault tree and repeatedly expands each logic gate until only basic events remain.

Let the following symbols be defined:

- w identifier of a logic gate
- φ identifier of a basic event
- $\rho_{w,i}$ i-th input to gate w

- λ_w fan-in (number of inputs) of gate w
- $\Delta_{x,y}$ entry located at row x, column y of the Boolean Indicated Cut Sets (BICS) matrix
- $x_{\max}$ ($y_{\max}$) current maximum row (column) index in Δ

2.5.3.2.1 Recursive expansion Starting with $\Delta_{1,1} = w_{\text{TOP}}$, every occurrence of a gate identifier is eliminated via the following transformation rules:

$$\Delta_{x,y} = \rho_{w,1}, \tag{2.36}$$

$$\Delta_{x,y_{\max}+1} = \begin{cases} \rho_{w,\pi}, & \text{if } w \text{ is an AND gate,} \\ \begin{cases} \Delta_{x,n}, \quad n = 1, \dots, y_{\max}, \; n \neq y, \\ \rho_{w,\pi}, \quad n = y, \end{cases} & \text{if } w \text{ is an OR gate.} \end{cases} \tag{2.37}$$

with $\pi = 2, 3, \dots, \lambda_w$. Equation (2.36) seeds the matrix, while Equation (2.37) generates new columns (AND) or rows (OR) until all w symbols have been replaced by φ symbols, thereby producing the complete set of BICS.

Two refinement steps convert BICS to MCS:

1. Remove duplicate basic events within each row.
2. Discard any BICS that is a superset of another BICS.

Table 2.1 summarizes the sample fault tree by matching the formulas provided for the algorithm.

Figure 2.4 schematically depicts the evolution of the $\Delta_{x,y}$ matrix for the fault tree in Figure 2.3.

After gate expansion, elimination of duplicates (e.g. event X_2) and removal of supersets (e.g. $\{X_2, X_4, X_3\}$) yield the MCS family:

$$\{\, \{X_1, X_2\},\ \{X_2, X_3\},\ \{X_1, X_4\} \,\}.$$

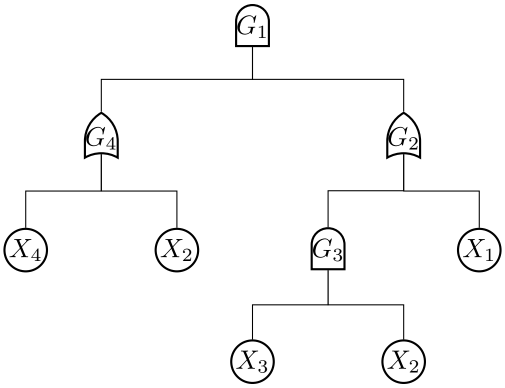


Figure 2.3: Sample fault tree for MOCUS demonstration [17].

Table 2.1: Structural data for the sample fault tree in Figure 2.3.

w	Gate type	λ_w	$\rho_{w,i}$
G_1	AND	2	G_2 G_4
G_2	OR	2	X_1 G_3
G_4	OR	2	X_2 X_4
G_3	AND	2	X_2 X_3

Although algorithmically elegant, the recursive nature of MOCUS can incur considerable computational overhead for large, deeply nested fault trees. Nonetheless, virtually every PRA tool integrates some variant of MOCUS for minimal cut-set calculation.

2.6 Importance Measures for Basic Events

Reliability and PRA studies rely on *importance measures* to quantify how individual basic events contribute to an overall risk metric R (such as a top-event probability in a mission time window or a frequency measure like CDF or Large Early Release Frequency (LERF)). These measures support ranking, screening, design improvement, and configuration-risk management. This section formalizes the measures most widely used in PRA/QRA, adopting notation and sign conventions consistent with the NASA PRA Practitioners' Guide [78] and US NRC risk-informed guidance.

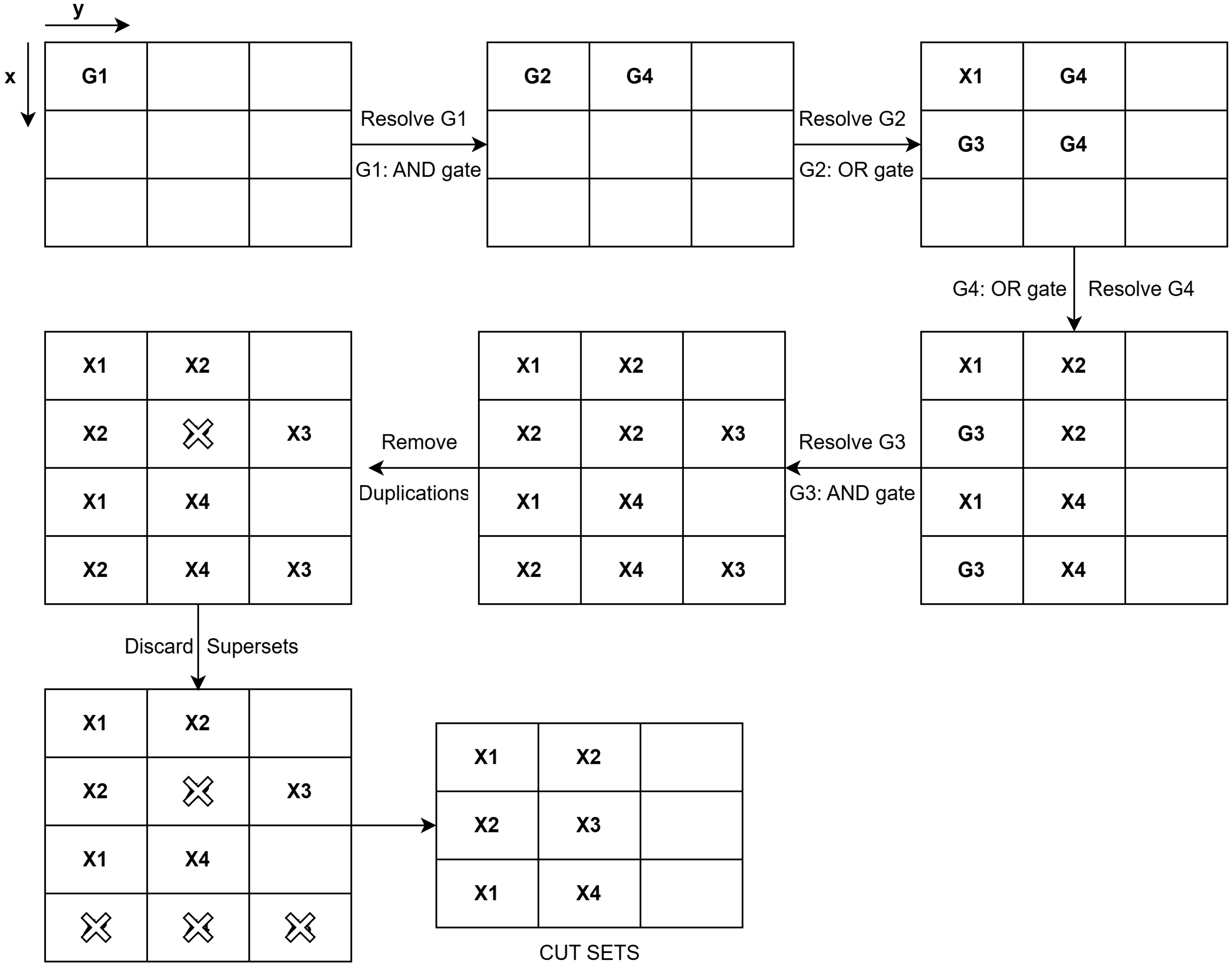


Figure 2.4: Schematic representation of MOCUS algorithm application to the sample fault tree [17].

Notation

- q_i – failure/unavailability probability of basic event i.
- x_i – binary state indicator of event i ($x_i = 1$: failed, $x_i = 0$: succeeded).
- R – baseline risk metric (probability or frequency).
- $R \mid x_i = 1$ – risk when event i is forced failed.
- $R \mid x_i = 0$ – risk when event i is forced successful.
- $\mathcal{MCS}$ – union of all minimal cut sets (failure space).
- $\mathcal{MCS}_i$ – union of minimal cut sets that contain event i.

Throughout, forcing a basic event applies the Boolean absorption $x_i = 1 \rightarrow$ true or $x_i = 0 \rightarrow$ false in the fault-tree logic before re-quantification.

2.6.1 Fussell–Vesely Importance (FV)

$$\mathrm{FV}_i = \frac{\Pr(\cup\,\mathcal{MCS}_i)}{\Pr(\cup\,\mathcal{MCS})} = \frac{R - R \mid x_i = 0}{R}. \tag{2.38}$$

Equation (2.38) expresses the fractional contribution of basic event i to the current risk. High FV values indicate that a large proportion of current risk flows through cut sets containing i; it is therefore the workhorse metric for contributor ranking and screening.

2.6.2 Risk Reduction Worth (RRW)

$$\mathrm{RRW}_i = \frac{R}{R \mid x_i = 0}, \qquad \Delta R_i^- = R - R \mid x_i = 0. \tag{2.39}$$

A large RRW signals substantial potential benefit from reliability or availability upgrades of component i. The interval form ΔR_i^- is useful when an absolute (rather than relative) risk decrement is required.

2.6.3 Risk Achievement Worth (RAW)

$$\mathrm{RAW}_i = \frac{R \mid x_i = 1}{R}, \qquad \Delta R_i^{+} = R \mid x_i = 1 - R. \tag{2.40}$$

RAW asks how much risk would be *achieved* (increased) if component i were failed. It is central to configuration-risk management; industry practice often screens for $\mathrm{RAW}_i > 2$ when identifying safety-significant items.

2.6.4 Birnbaum (Marginal) Importance (MIM)

$$I_{\mathrm{B}}(i) = \frac{\partial R}{\partial q_i} \approx R \mid x_i = 1 - R \mid x_i = 0, \quad \text{(linear-risk models)} \tag{2.41}$$

Birnbaum importance is a local sensitivity: it measures the structural leverage of i on R. Because it is not scaled by q_i, a very reliable yet pivotal component can have a high I_{B}.

2.6.5 Differential Importance Measure (DIM)

For a contemplated small change vector $\{\mathrm{d}q_j\}$,

$$\mathrm{DIM}_i = \frac{\dfrac{\partial R}{\partial q_i}\,\mathrm{d}q_i}{\displaystyle\sum_j \frac{\partial R}{\partial q_j}\,\mathrm{d}q_j}. \tag{2.42}$$

The DIMs sum to 1 and allocate the total risk change across parameters, which can be useful when many small improvements are pursued simultaneously.

2.6.6 Criticality Importance (CRI)

$$\mathrm{CRI}_i = I_{\mathrm{B}}(i)\,\frac{q_i}{R} = \frac{q_i\,[R \mid x_i = 1 - R \mid x_i = 0]}{R}. \tag{2.43}$$

CRI gives the posterior probability that event i has both failed and is *critical* when the top event occurs, supporting post-event diagnostics and maintenance prioritization.

2.6.7 Diagnostic Importance Factor (DIF)

$$\mathrm{DI}_i = \Pr(x_i = 1 \mid \mathrm{Top}) = \frac{q_i\, R \mid x_i = 1}{R}. \tag{2.44}$$

Unlike CRI, DIF does not require that i be pivotal, only that it has failed. It thus quantifies the probability that i is the culprit in need of inspection once the system has failed.

2.6.8 Improvement Potential (IP)

$$\mathrm{IP}_i = R - R \mid x_i = 0 = R\,\mathrm{FV}_i. \tag{2.45}$$

IP expresses the absolute risk points removable by perfecting component i. When paired with a cost metric, it guides benefit-per-dollar prioritization in maintenance optimization.

2.6.9 Prevention Worth (PW)

In the success domain (path-set analysis), PW formalizes how much the overall *mission success* probability improves when basic event i is guaranteed to succeed. Let

$$S \;=\; 1 - R \;=\; \Pr\bigl(\cap\,\mathcal{MPS}\bigr), \tag{2.46}$$

where S is the baseline probability of reaching the top success state and $\mathcal{MPS}$ denotes the union of all minimal path sets.

Forcing $x_i = 0$ in the logic (i.e., assuring component i succeeds) yields an updated success probability $S \mid x_i = 0 = 1 - R \mid x_i = 0$. The *Prevention Worth* of i is then

$$\mathrm{PW}_i \;=\; \frac{S \mid x_i = 0}{S} \;=\; \frac{1 - R \mid x_i = 0}{1 - R}, \qquad \Delta S_i^{+} = S \mid x_i = 0 - S. \tag{2.47}$$

A value $\mathrm{PW}_i > 1$ signals that assuring the success of component i produces a proportional uplift in overall mission success, while the interval form ΔS_i^{+} gives the absolute success points gained. These metrics support availability optimization and design-for-success studies in contexts where objectives are framed in terms of maximizing S rather than minimizing R.

Chapter 3

Probabilistic Circuits

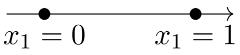
3.1 Boolean Functions

Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be a vector of n Boolean variables, where each $x_i \in \{0, 1\}$. A *Boolean function* is a map

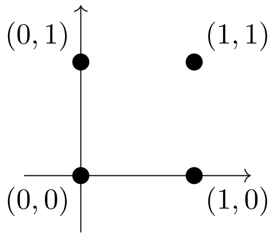
$$F(\mathbf{x}) \;=\; F(x_1, x_2, \dots, x_n) \;\in\; \{0, 1\}. \tag{3.1}$$

This function takes each possible configuration of $\mathbf{x}$ (i.e., each element of $\{0, 1\}^n$) to a single binary output in $\{0, 1\}$. Boolean functions appear throughout digital logic, circuit design, and a wide range of computational applications.

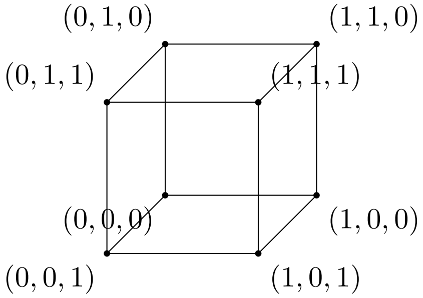
For illustration, we can visualize small Boolean functions based on the size of $\mathbf{x}$. For $n = 1$, there are two possible input states:



For $n = 2$, the four possible states can be positioned on a 2D lattice:



In three dimensions ($n = 3$), the eight possible states correspond to the vertices of a cube:



Boolean operators such as AND ($\wedge$), OR ($\vee$), and NOT ($\neg$) allow constructing a wide variety of logical relationships. For example, in a setting where a system fails if any one of two components fails, the Boolean function can be written as

$$F(x_A, x_B) \;=\; x_A \;\vee\; x_B.$$

Here, $F = 1$ precisely when $x_A = 1$ or $x_B = 1$, encompassing a failure event if either component A or B is in state 1. More complex systems with many interdependent components may require Boolean functions with numerous variables and deeply nested operators.

Although Boolean functions are crucial for representing logical configurations, they operate purely in a binary framework and do not directly encode probability distributions. To include probabilistic behavior, we can move to a more expressive framework called *probabilistic circuits*, which describe the distribution of variables in a directed acyclic graph (DAG). Such representations can capture both the combinatorial structure of system states and the uncertainty or likelihood associated with these states.

3.2 Definition and Structure

Consider a set of random variables $\mathbf{X} = (X_1, X_2, \ldots, X_n)$. A *probabilistic circuit* $\mathcal{C}$ is a DAG whose nodes consist of:

- **Input nodes (leaves):** Each leaf encodes a base distribution over some subset of $\mathbf{X}$. Often, these leaves correspond to univariate distributions $p(X_i)$ or constant/indicator functions.

- **Internal nodes (gates):** Each gate combines incoming distributions from its children using either:

  - **Sum-gates (mixture gates):** Weighted sums of child distributions, with non-negative weights summing to 1.
  - **Product-gates:** Factorized products of child distributions, each child covering disjoint subsets of $\mathbf{X}$.

The acyclic nature of the graph ensures that information flows consistently from the leaves toward a designated *root* node.

3.2.1 Sum-Gates and Product-Gates

Let v be an internal node in $\mathcal{C}$. Denote the children of v by $\mathrm{ch}(v)$. Then:

- **Sum-gate:** Suppose v has children $u_1, \ldots, u_k$ with mixture weights $\{\theta_{v,u_i}\}_{i=1}^{k}$ satisfying $\sum_{i=1}^{k} \theta_{v,u_i} = 1$ and $\theta_{v,u_i} \geq 0$. The distribution encoded at v is

$$p_v(\mathbf{x}) \; = \; \sum_{i=1}^{k} \theta_{v,u_i} \, p_{u_i}(\mathbf{x}), \tag{3.2}$$

  where $p_{u_i}(\mathbf{x})$ is the distribution encoded by child node u_i.

- **Product-gate:** Suppose v has children $u_1, \ldots, u_k$, each covering disjoint subsets of $\mathbf{X}$. Let $\mathbf{X} = \bigcup_{i=1}^{k} \mathbf{X}_{u_i}$ and $\mathbf{X}_{u_i} \cap \mathbf{X}_{u_j} = \varnothing$ for $i \neq j$. Then the distribution at v is

$$p_v(\mathbf{x}) \; = \; \prod_{i=1}^{k} p_{u_i}\big(\mathbf{x}_{u_i}\big), \tag{3.3}$$

  where $\mathbf{x}_{u_i}$ is the restriction of $\mathbf{x}$ to the variables in $\mathbf{X}_{u_i}$.

3.2.2 Leaf Nodes and the Circuit Distribution

Each leaf node encodes a base distribution over its subset of variables (or a constant/indicator). Let v be a leaf node associated with $p_v(\mathbf{X}_v)$. When the circuit is evaluated, each leaf contributes its assigned distribution or constant term. By recursively composing sum-gates and product-gates, every node v in the circuit defines a distribution $p_v(\mathbf{x})$. The distribution of the entire circuit is given by evaluating its *root* node r:

$$p_r(\mathbf{x}) \;=\; (r \text{ evaluated from the leaves up}).$$

Probabilistic circuits unify structural and probabilistic modeling in a single formalism. They are widely used in fields such as artificial intelligence, machine learning, and automated reasoning, offering a tractable way to represent complex, high-dimensional probability distributions while preserving interpretable, compositional structure.

3.3 Connection to Probabilistic Risk Assessment

Chapters 2.2–4.1 introduce event–tree and fault–tree logic and show how their inter–connections form a unified probabilistic directed acyclic graph (PDAG). That PDAG is, in fact, a *specialized probabilistic circuit*:

- The leaf distributions are the Bernoulli failure variables of basic events (and, later, common–cause variables).

- Internal OR / AND / voting gates correspond to deterministic product– or sum–gates whose mixture weights are restricted to $\{0, 1\}$.

- Event–tree branching is expressed through sum–gates with non-trivial weights $\theta_{u\to v}$ that encode the conditional probabilities of functional events.

Framing PRA models as probabilistic circuits brings three advantages that motivate the developments in subsequent chapters:

(a) **Unified semantics.** Both logical determinism and stochastic branching coexist in one mathematical object, enabling end–to–end inference without shuttling between separate “logic” and “probability” representations.

(b) **Compiler compatibility.** Decades of work on knowledge compilation for circuits (§18.2) becomes directly applicable, informing the eight-level transformation pipeline adopted in Chapter 18.

(c) **Sampling tractability.** The layered Monte-Carlo kernels of Part III exploit the decomposability of product–gates to evaluate entire batches in parallel, while sum–gates inject controlled stochasticity that preserves unbiasedness (Chapter 14).

With this perspective, the remainder of Part I can be read as instantiating general circuit concepts with the domain-specific structure of nuclear PRA models. Chapter 4 will leverage that instantiation to formalize quantitative risk assessment as a problem of *knowledge compilation* over probabilistic circuits, setting the stage for the data-parallel algorithms analyzed in Parts III and IV.

Chapter 4

Quantitative Risk Assessment as Knowledge Compilation

The traditional PRA modeling language—with its separate constructs for Initiating Events, Event Trees, and Fault Trees—was devised for human readability and incremental model construction. Those distinctions help engineers reason about failure propagation and refine a model's granularity, yet they also fragment information across heterogeneous data structures. For small systems this heterogeneity is innocuous; for today's models containing hundreds of event trees and thousands of fault trees it becomes a barrier to scalable analysis, impeding both exact probability calculations and qualitative queries such as minimal-cut-set enumeration.

To overcome these limitations we advocate translating the entire model into a single probabilistic directed acyclic graph (PDAG). The transformation is loss-free, deterministic, and yields a representation that is simultaneously human-verifiable and machine-amenable. Once in PDAG form the model enjoys three key benefits: (i) a uniform graph data structure that supports high-performance algorithms for probability estimation and structural queries, (ii) compatibility with knowledge-compilation techniques that enable selective normal-form transformations, and (iii) the ability to accommodate incremental design changes through lightweight graph updates.

The remainder of this chapter explains the PDAG construction, illustrates the mapping on the running example of Fig. 4.1, and shows how classical PRA operations can be re-expressed as graph traversals or as queries over a propositional knowledge base. This perspective not only clarifies the formal semantics of PRA models but also lays the groundwork for the data-parallel Monte-Carlo methods developed in later chapters.

In this section, we will show how these requirements can be achieved by viewing risk models as Probabilistic Directed Acyclic Graph (PDAG). We show how PDAG model maps on PDAG and how standard PRA methods can be viewed as operations on graphs. Furthermore, we show how PDAGs can be viewed as collections of propositional logic statements — knowledge bases. This view provides fundamental mathematical rigor to PRA formalism, by casting PRA methods as queries over and transformation of aforementioned knowledge bases. Not only this formalism provides strong computational bounds and guarantees for the algorithms of interest, but it also explicitly separates the preparation and analysis steps of PRA model, allowing for efficient separation and querying.

4.1 Risk Models as Probabilistic Directed Acyclic Graphs

Up to this point, we have introduced ETs to capture the forward evolution of scenarios and FTs to capture the top-down decomposition of system failures. In a full-scale PRA, many ETs and FTs are linked to form a single overarching model. The goal is to represent:

1. The branching structure of multiple event trees, which may feed into one another (ET→ET),

2. Multiple fault trees that themselves can reference or be referenced by other FTs (FT→FT),

3. Event trees that invoke fault trees to quantify key failure probabilities (ET→FT), and similarly fault trees whose outcome may direct the next branch or state in an event tree.

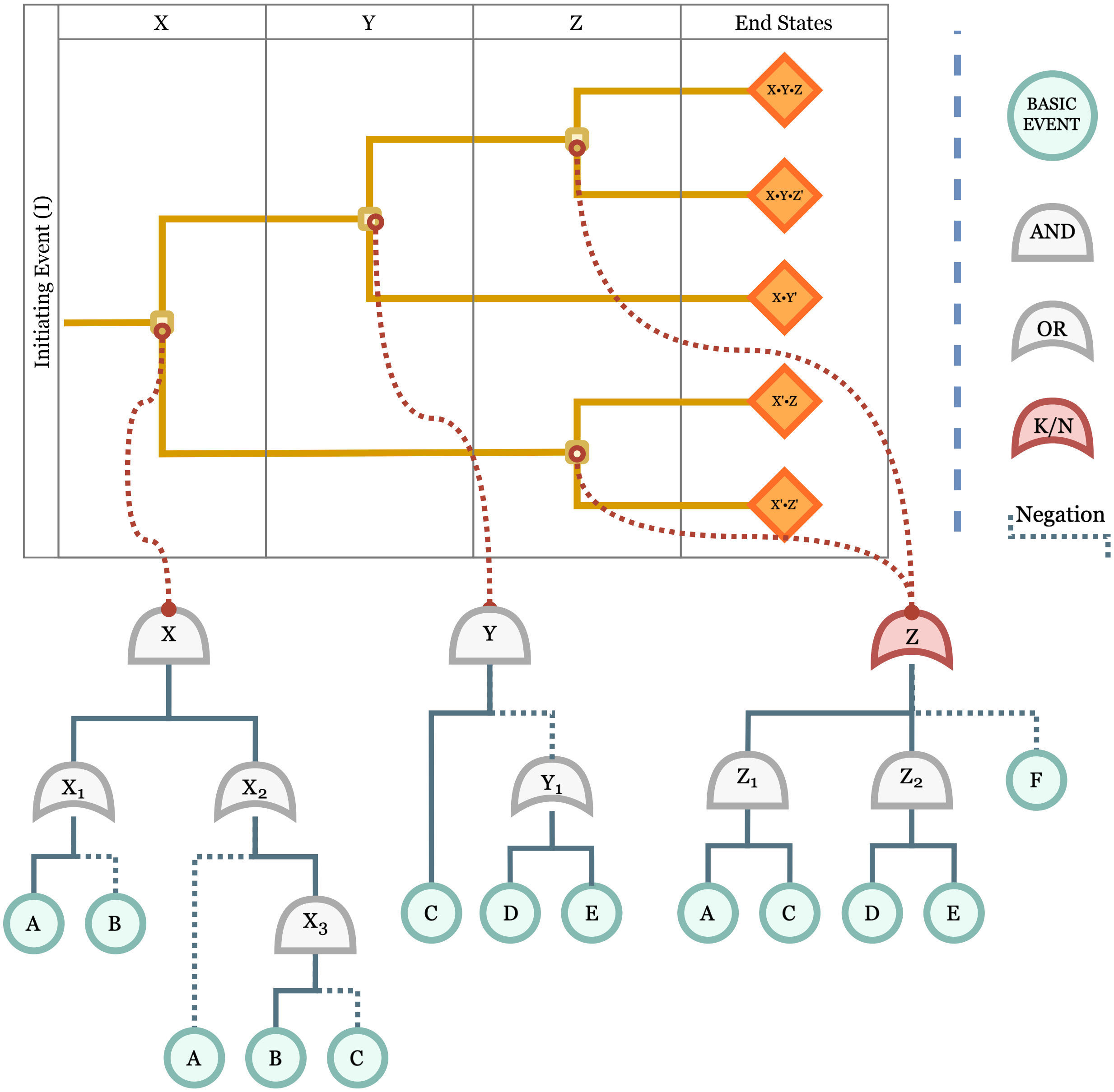


Figure 4.1: A working example: Starting with an initiating event (I), an event tree (ET) with three linked fault trees (FT), and shared basic events (FT), and five end states (ES).

All of these interconnections can be consolidated into a single PDAG. In broad terms, its nodes stand for either (i) ET nodes (initiating or functional events), (ii) FT gates or intermediate events, or (iii) basic events.

4.1.1 Basic Structure and Notation

Let us denote:

- $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_M\}$ as the *collection of event trees*, where each Γ_i may represent a different initiating event or system phase. Every Γ_i is itself structured as in Section 2.2, with a set of node events and directed edges (success/failure branches).
- $\{\Phi_1, \Phi_2, \ldots, \Phi_N\}$ as the *collection of fault trees*, each built according to Section 2.3. Every Φ_j has a unique top event, an acyclic arrangement of gates (AND, OR, voting, etc.), and a set of basic events.

In a large PRA, any ET Γ_i may:

- Lead to another ET Γ_k under certain branch outcomes (ET→ET).
- Include a branch that requires computing "System X fails" via a fault tree Φ_j (ET→FT).

Similarly, a fault tree Φ_j may:

- Contain the top event of another fault tree Φ_k as one of its inputs (FT→FT).
- Generate an outcome (e.g. subsystem fails) that triggers a branch in some event tree Γ_i.

These inter-dependencies can be organized into a single PDAG, denoted

$$\mathcal{M} \;=\; (\mathcal{V},\, \mathcal{A}),$$

where:

- $\mathcal{V}$ is the set of *all* nodes in the unified model. Each node $v \in \mathcal{V}$ has a type indicating whether it belongs to an event tree (ET-node), a fault tree (FT-gate), or is a Basic Event (BE).

- $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of directed edges. Each $(u, v) \in \mathcal{A}$ signifies a logical or probabilistic dependency from node u to node v.

By design, $\mathcal{M}$ is acyclic: no path loops back through the same node. This condition prevents paradoxical definitions of probabilities or statements (e.g., an event that depends on itself).

4.1.2 Nodes and Their Inputs

We partition $\mathcal{V}$ into three principal categories:

1. **Basic Events (BEs).** Let $\mathcal{B} \subset \mathcal{V}$ be the set of all basic events across all FTs. Each $b \in \mathcal{B}$ is associated with a failure probability $p(b) \in [0, 1]$. A node b in the PDAG has no incoming edges (i.e., it is a leaf source for the rest of the logic).

2. **Fault Tree (FT) Nodes.** Let $\mathcal{G} \subset \mathcal{V}$ be the set of *internal FT-gates or intermediate FT-events*, unified across $\{\Phi_1, \ldots, \Phi_N\}$.

    - Each FT node $g \in \mathcal{G}$ has one output event g (the node itself in the PDAG).

    - The set of inputs to g may include basic events $\mathcal{B}$ (e.g., component failures) and/or other FT nodes $\mathcal{G}$ (subsystem-level events). If g is the top event of Φ_j, then it may appear as an input into a *different* FT's gate or an ET node.

    - As with standard FT logic, each gate has a type in {AND, OR, VOT(k/n), …}. Its output (failure state) is a Boolean function of its inputs' failure states.

3. **Event Tree (ET) Nodes.** Let $\mathcal{E} \subset \mathcal{V}$ be the set of *ET-nodes*, each referring either to an initiating event (IE) or to a functional event within some event tree Γ_i.

- An ET node e in Γ_i can have multiple outgoing edges, each labeled by a particular outcome (e.g., success/failure). These edges may lead to another ET node (continuing the same event tree), to the root/top event of a fault tree (to evaluate subsystem reliability), or even to the root of a *different* event tree (ET→ET).
- As in Section 2.2, each outgoing branch from e has an associated conditional probability, conditioned on the event e itself having occurred.

Hence,

$$\mathcal{V} \;=\; \underbrace{\mathcal{B}}_{\text{basic events}} \;\cup\; \underbrace{\mathcal{G}}_{\text{FT nodes}} \;\cup\; \underbrace{\mathcal{E}}_{\text{ET nodes}},$$

and every node v in $\mathcal{V}$ has an *input set*

$$I(v) \;\subseteq\; \mathcal{B} \cup \mathcal{G} \cup \mathcal{E} \;=\; \mathcal{V} \setminus \{v\},$$

indicating which nodes feed into v. By the PDAG property, v cannot be an ancestor of itself.

4.1.3 Edge Types and Probability Assignments

Each directed edge $(u, v) \in \mathcal{A}$ belongs to one of several categories:

- **ET → ET edges.** [**Transfers**] These edges connect an ET node $u \in \mathcal{E}$ to another ET node $v \in \mathcal{E}$ within the same tree Γ_i or leading to a subsequent tree Γ_k. In typical diagrams, these edges denote if u occurs, then with probability $\theta_{u \to v}$ we transition to v. Probabilities on all child edges of u sum to 1, reflecting the partition of possible outcomes.
- **ET → FT edges.** [**Functional Events**] These edges represent the case where an ET node $u \in \mathcal{E}$ designates Check if subsystem Φ_j has failed. Formally, the next node $v \in \mathcal{G}$ is the top event (or relevant subsystem event) in fault tree Φ_j. The probability of v failing is not assigned directly on the edge but is computed via the logical structure of Φ_j.

- **FT → FT edges. [Transfer Gates]** Such edges arise when the top event (or an intermediate gate) $u \in \mathcal{G}$ of one fault tree is input to a gate $v \in \mathcal{G}$ in another fault tree. For instance, if Φ_1 captures the failure mode of a pump and Φ_2 captures the failure of a coolant subsystem that includes that same pump's top event.

- **FT → ET edges. [Initiating Events]** Less common but still possible are edges that carry an outcome of a fault tree node $u \in \mathcal{G}$ to an ET node $v \in \mathcal{E}$. For instance, an initiating event might depend on whether a certain subsystem fails, as computed by a separate FT Φ_j. Support system FTs are one such example.

- **BEs as sources (no incoming edges).** Each basic event $b \in \mathcal{B}$ has probability $p(b)$ of failing, so $\Pr[X_b = 1] = p(b)$. These do not have incoming edges because they represent fundamental failure modes, not dependent on other events within the model.

Denote by $\theta_{u \to v}$ the *conditional probability* weighting an ET-type edge $(u \to v)$. If node u splits into children $v_1, \ldots, v_k$, then

$$\sum_{i=1}^{k} \theta_{u \to v_i} \; = \; 1, \quad \theta_{u \to v_i} \; \geq \; 0.$$

By contrast, FT-type edges do not carry numerical probabilities directly. Instead, a gate node $v \in \mathcal{G}$ aggregates its inputs' *fail/success states* via Boolean logic (AND, OR, VOT(k/n), etc.) to yield $\pi_{\mathcal{M}}(S, v) \in \{0, 1\}$, the node v's failure state under a set S of basic-event failures.

4.1.4 Semantics of the Unified Model

A full *scenario* in $\mathcal{M}$ extends from a designated *initial node* (often an initiating event $I \in \mathcal{E}$) forward through whichever ET or FT edges are triggered. Because no cycles exist, every path eventually terminates in either (a) an ET leaf (end-state), (b) a top event that is not expanded further, or (c) a final subsystem outcome deemed not to propagate further risk.

4.1.4.1 Failure States in the Fault Trees.

For any subset $S \subseteq \mathcal{B}$ of basic events that fail:

1. Each $b \in \mathcal{B}$ fails iff $b \in S$.
2. Each FT node $g \in \mathcal{G}$ has failure state $\pi_F(S, g)$ determined by the usual fault tree semantics (Section 2.3).

That is, a node g in a fault tree Φ_j is in failure mode when its logical gate type indicates failure is activated by the failures of its inputs (which might be other gates or basic events).

4.1.4.2 Branching in the Event Trees.

Whenever an ET node $e \in \mathcal{E}$ is reached, outgoing edges $\{(e \to e_1), (e \to e_2), \dots\}$ partially partition the scenario space. The choice of which child e_i is realized is probabilistic, with probabilities $\theta_{e \to e_i}$.

4.1.4.3 Event-Tree to Fault-Tree Links.

If an edge $(e \to g)$ connects an ET node e to a *fault tree top event* $g \in \mathcal{G}$, the scenario path triggers the question Does g fail? The probability that g is in failure, conditional on having arrived at node e, is determined by the set $S \subseteq \mathcal{B}$ of basic events that happen to fail in that scenario plus the gate logic of Φ_j.

Altogether, scenario outcomes in $\mathcal{M}$ thus combine:

- $\mathbf{X}_{\mathcal{B}} = \{\, X_b : b \in \mathcal{B}\}$, where $X_b \in \{0, 1\}$ indicates whether basic event b fails or not, and
- A chain of ET decisions or fault-tree outcomes, traveling through the PDAG until reaching a terminal node.

If all basic events are assumed independent with probabilities $\{p(b)\}$, then the *global* likelihood of a specific path ω from an initiating event I to a final outcome (and with a particular pattern of success/failure across $\mathcal{B}$) factors into products of:

1. The product of $\prod_{b \in S} p(b) \times \prod_{b \notin S} [\, 1 - p(b) \,]$ for the relevant $S \subseteq \mathcal{B}$.

2. The product of all ET-edge probabilities $\theta_{u \to v}$ encountered along the path ($u \in \mathcal{E}$).

3. The logical constraints from each visited fault tree node $g \in \mathcal{G}$, which impose $\pi_F(S, g) \in \{0, 1\}$ in or out of failure.

Summing over all valid paths (or equivalently over all subsets $S \subseteq \mathcal{B}$ and the result of each ET/FT branching) yields the total system risk measure, such as the probability of a severe radiological release, or the probability that a certain undesired top event emerges.

4.1.5 Formal Definition of the Unified Model

Bringing these elements together, we propose the following definition:

Definition 1 (Unified PRA Model)**.** *A* unified PRA model *is a Probabilistic Directed Acyclic Graph (PDAG)*[1]

$$\mathcal{M} \;=\; \Big\langle \mathcal{V} \;=\; \mathcal{B} \cup \mathcal{G} \cup \mathcal{E}, \;\; \mathcal{A}, \;\; p(\cdot), \;\; \pi_F \Big\rangle$$

with the following properties:

1. *$\mathcal{B}$ is the set of **basic events**, each $b \in \mathcal{B}$ failing with probability $p(b)$. These nodes have no incoming edges in $\mathcal{M}$.*

2. *$\mathcal{G}$ is the set of **Fault Tree nodes**, each representing a gate or intermediate event in a fault tree. For $g \in \mathcal{G}$, the function*

$$\pi_F(S, g) \;=\; \begin{cases} 1, & \textit{if fault-tree logic declares } g \textit{ fails under } S \subseteq \mathcal{B}, \\ 0, & \textit{otherwise.} \end{cases}$$

[1] Whether the "P" in PDAG is "probabilistic" or "propositional" is, appropriately, still a probabilistic proposition, so we'll invoke whichever flavor suits the moment.

*3. $\mathcal{E}$ is the set of **Event Tree nodes**, each having zero or more outgoing edges. If node e has children $\{v_1, \ldots, v_k\} \subset \mathcal{V}$, then the edges $\{(e \to v_i)\}_{i=1}^{k}$ carry probabilities $\theta_{e \to v_i} \geq 0$ summing to 1.*

4. $\mathcal{A} \subseteq (\mathcal{V} \times \mathcal{V})$ is the set of directed edges. An edge $(u \to v)$ can be:

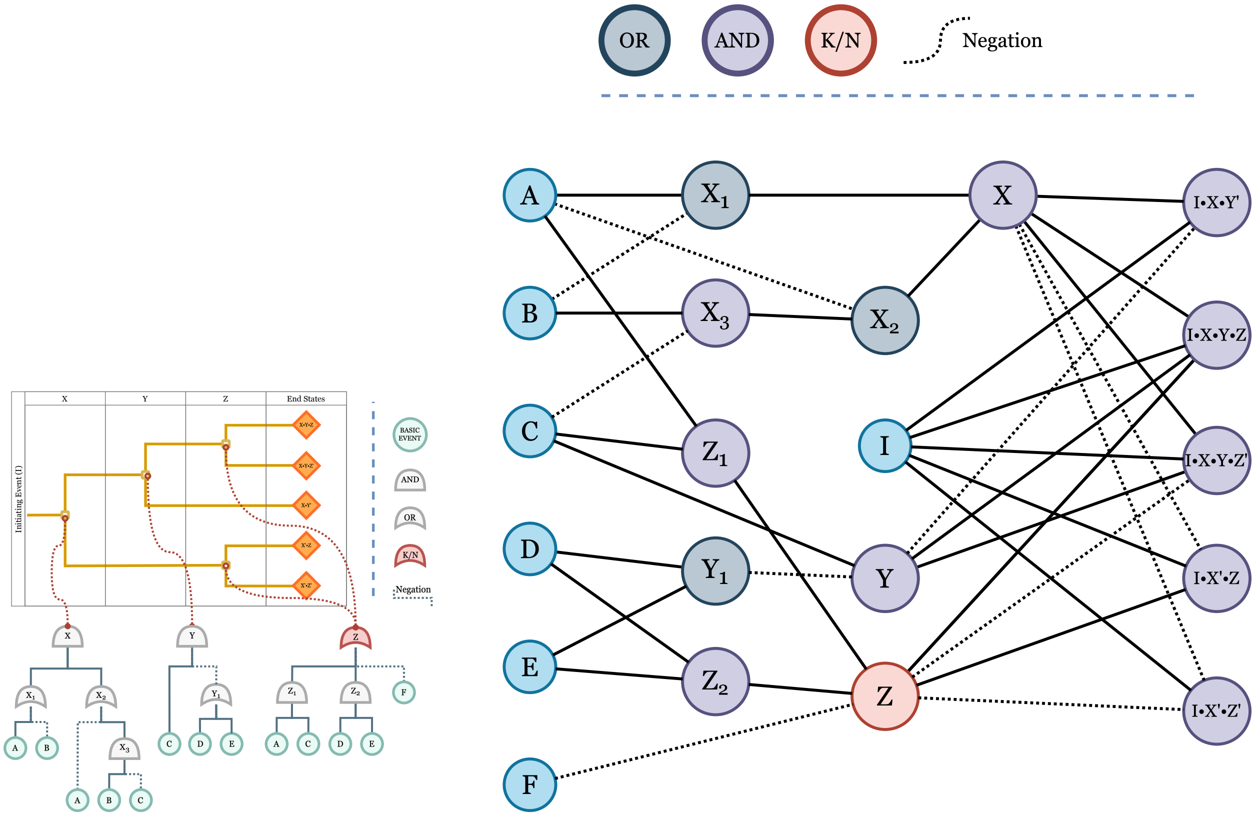
- ET → ET*: event-tree branching,*
- ET → FT*: an event tree referencing a fault tree's top event,*
- FT → FT*: linking one fault tree node to another's input,*
- FT → ET*: an FT outcome passed back to an event tree node,*
- BE → ∅*: no edges emanate from a basic event node.*

5. The graph $\mathcal{M}$ is acyclic: no path in $\mathcal{A}$ returns to a previously visited node.

4.1.6 Operations on Probabilistic Directed Acyclic Graphs

In practice, we anticipate large nuclear PRAs to contain hundreds of event trees and thousands of fault trees, sharing many basic events or subsystem-level fault trees. By embedding them in $\mathcal{M}$, one can systematically compute probabilities for any high-level risk measure (e.g., core damage, large release) by enumerating scenario paths or using specialized algorithms (e.g. binary decision diagrams, minimal cut set expansions, or simulation-based techniques). The unified PDAG structure codifies both the forward scenario expansions (ET logic) and the top-down sub-component dependencies (FT logic) without creating contradictions or cycles. Definition 1 makes manifest *which* event tree references *which* fault tree, how fault trees are chained together, and how basic events ultimately feed every higher-level node. Once built, one may:

- **Traverse** each path from an initiating event to a final end-state, propelling forward along the ET edges and evaluating any FT nodes via π_F.



(a) Working example from Fig. 4.1.

(b) PDAG constructed from Fig.4.2a

Figure 4.2: A unifying representation of a Probabilistic Risk Assessment (PRA) model as a Propositional Directed Acyclic Graph (DAG).

- **Find minimal cut sets and path sets** by traversing the PDAG and making cuts along the way.

- **Sum** (or bound, or approximate) scenario probabilities to quantify overall risk.

- **Perform sensitivity analyses** by biasing subsets of basic-event probabilities $p(b)$ or gate dependencies.

All standard PRA methods (minimal cut sets, Monte Carlo simulation, bounding formulas, etc.) remain applicable, but now from within a single unified PDAG representation.

4.2 Transformations

4.2.1 Knowledge Compilation

Casting PRA model as PDAG allows to view the model a set of boolean propositional statements. Thus, any question that can be asked about the system, can be seen as a general reasoning (queries or transformations) over the knowledge base, defined by the propositional statements. A common approach to dealing with such problems is knowledge compilation.

Knowledge Compilation has emerged as a significant direction of research for addressing the computational intractability inherent in general propositional reasoning tasks. This approach, which has a long tradition in reasoning Artificial Intelligence, was notably structured and analyzed in the work of Darwiche and Marquis. KC fundamentally involves splitting the reasoning process into two distinct phases:

1. An **off-line compilation phase**: In this initial phase, a knowledge base (represented, for instance, as a propositional theory or formula) is transformed or ”compiled” into a different representation, referred to as a tractable form, or target language. The target language is specifically chosen because it supports certain desirable properties, such as tractability for specific queries (questions) or allowing polynomial time evaluation.

2. An **on-line query-answering phase**: Once compiled, the resulting target representation is utilized to answer queries efficiently. The goal is for these query answering procedures to be polynomial time with respect to the size of the compiled representation.

The primary rationalization behind this two-phase approach is to **shift as much of the computational overhead as possible into the off-line compilation phase**. While the compilation step itself can be computationally hard, this initial cost is amortized over the potentially large number of subsequent on-line queries. This amortized cost makes the overall reasoning process more efficient when multiple queries are anticipated on the same knowledge base.

4.2.2 Negation Normal Form (NNF)

The field of knowledge compilation operates primarily on a subset of boolean expression that conform to a set of properties. In general, imposition of more properties results increased tractability of the queries, i.e. answering questions becomes easier, however, at the cost of the size of boolean expression. The primary "workhorse" of Knowledge compilation is Negation Normal Form (NNF).

Boolean Negation Normal Form (NNF) is a syntactic restriction for Boolean formulas such that negations (NOT operators) are applied only directly to variables and not to compound subformulas. Formally, a Boolean formula is in NNF if it is built from variables, their negations, conjunctions (AND), and disjunctions (OR), where NOT appears only as part of literals.

A boolean expression in NNF can be represented as a rooted directed acyclic graph, DAG. The leaf of the graph correspond to constants (0, 1) or literals (a, $\neg b$), presented in the expression. The internal nodes of the DAG correspond to AND ($\wedge$) and OR ($\vee$) gates. Internal gates cannot be associated with NOT gates.

In the context of knowledge compilation, NNF serves as a foundational target language. Knowledge bases compiled into NNF allow efficient model checking and form the basis for further restricted normal forms like Conjunctive Normal Form (CNF), Disjunctive Normal Form (DNF), Decomposable Negation Normal Form (DNNF), or Deterministic Decomposable Negation Normal Form (d-DNNF). NNF enables subsequent transformations and reasoning tasks to be performed with well-bounded complexity, as it ensures logical negation is "pushed down" to the leaves of the formula's parse tree, simplifying subsequent manipulations.

4.2.2.1 Properties of NNF

NNFs can be classified based on adherence to particular properties/restrictions. The set of NNFs that adhere to a set of properties defines a representational language, L in Table 4.2.

4.2.2.1.1 Decomposability An NNF is decomposable if, at every conjunction ($\wedge$) gate, the sets of variables feeding into the subformulas of its children are pairwise disjoint. That is, for any $\wedge$-node with children, the sets of variables involved in the subformulas rooted at those children share no variables. Formally, for any $\wedge$-node with children $(\alpha_1, \ldots, \alpha_n)$,

$$\mathrm{Vars}(\alpha_i) \cap \mathrm{Vars}(\alpha_j) = \emptyset \quad \forall i \neq j$$

This property ensures tractable consistency checking and supports efficient computation on the representation. The language of DNNF comprises those NNFs that are decomposable.

4.2.2.1.2 Determinism An NNF is deterministic if, at every disjunction ($\vee$) gate, the sets of models (i.e., assignments that make the subformulas true) computed by its children are mutually exclusive–no single assignment satisfies more than one child. Zero-overlap of assignments between children guarantees tractable model counting. Determinism together with decomposability yields Deterministic Decomposable Negation Normal Form (d-DNNF), which enables polytime validity, implicant, and model counting queries. Sentential Decision Diagram (SDD)s are a strict subset of d-DNNF with additional structure.

4.2.2.1.3 Smoothness An NNF is smooth if, at every disjunction (∨) gate, all children depend on the same set of variables (atoms). That is, for every OR-node, the set of variables in each child's subformula is identical. Smoothness simplifies certain operations in knowledge compilation and ensures that the models of disjuncts are over a fixed set of variables. Smoothness can always be enforced on any DNNF in polynomial time without affecting succinctness. Smooth/Structured Deterministic Decomposable Negation Normal Form (sd-DNNF) enforces decomposability, determinism, and smoothness.

4.2.2.1.4 Flatness An NNF is flat if the circuit/tree has height at most two: the root is an AND or OR, and the children are literals or simple conjunction/disjunctions of literals. Both CNF and DNF are examples of Flat Negation Normal Form (f-NNF): In CNF, the root is AND, its children are ORs of literals (clauses); in DNF, the root is OR, its children are ANDs of literals (terms). Flatness restricts structural complexity and further subclasses are defined by additional properties: for instance, CNF requires each clause to have no repeated variables, and DNF requires each term to have unique variables.

4.2.2.2 Summary

The following sections explore common target languages that find uses in Knowledge Compilation and can be used in PRA. For each selected language we note it's main properties, construction algorithms, and tractable queries available for this languages. We focus on the following:

1. CNF — the most well-studied form, ubiquitous in solving SAT problem.

2. DNF — a form that naturally renders itself for Event Tree analysis and acts as a precursor for more sophisticated essential prime implicant search.

3. BDD, ZDD, SDD — most tractable target languages, featured in the majority of practical applications.

This is not an exhaustive list. It's intention is to provide a summary of rigorous formalism that can find uses in Probabilistic Risk Assessment (PRA).

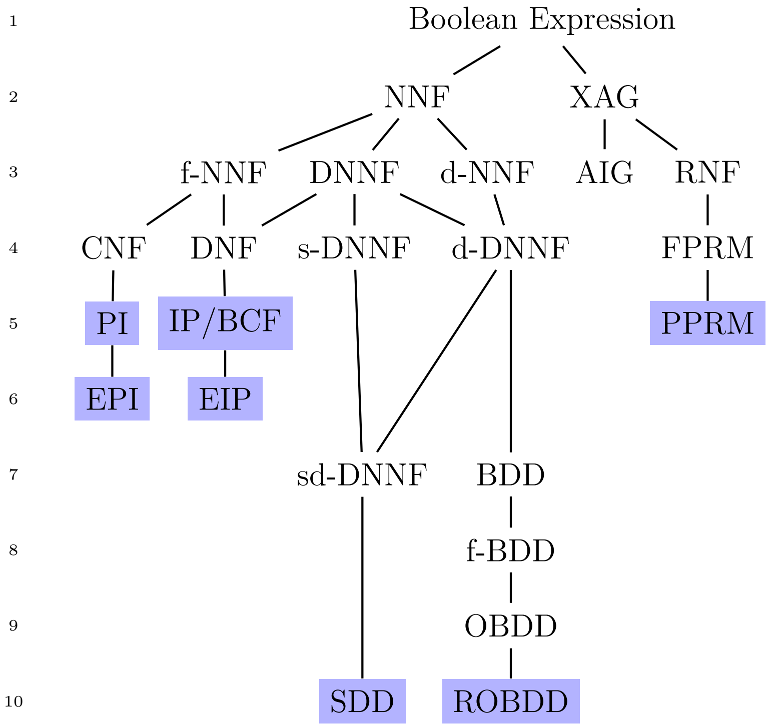


Figure 4.3: Hierarchy of compiled target languages. Blue nodes represent canonical forms.

Table 4.1: Compiled target languages, acronyms defined.

Acronym	**Full form**
NNF	Negation Normal Form
XAG	XOR-And-Inverter Graph
AIG	And-Inverter Graph
ANF/RNF	Algebraic/Ring Normal Form
f-NNF	Flat Negation Normal Form
DNNF	Decomposable Negation Normal Form
d-NNF	Deterministic Negation Normal Form

Table 4.1: Compiled target languages, acronyms defined.

Acronym	**Full form**
FPRM	Fixed Polarity Reed-Muller
CNF	Conjunctive Normal Form
DNF	Disjunctive Normal Form
s-DNNF	Smooth/Structured Decomposable Negation Normal Form
d-DNNF	Deterministic Decomposable Negation Normal Form
sd-DNNF	Smooth/Structured Deterministic Decomposable Negation Normal Form
PPRM	Positive Polarity Reed-Muller
PI	Prime Implicate
IP	Prime Implicant
BCF	Blake Canonical Form
EPI	Essential Prime Implicate
EIP	Essential Prime Implicant
BDD	Binary Decision Diagram
f-BDD	Free/Read-Once Binary Decision Diagram
OBDD	Ordered Binary Decision Diagram
SDD	Sentential Decision Diagram
RoBDD	Reduced Ordered Binary Decision Diagram

Table 4.2: Properties of selected target languages.

Property	**NNF**	**CNF**	**DNF**	**DNNF**	**d-DNNF**	**SDD**	**ROBDD**
Decomposable			$\top$	$\top$	$\top$	$\top$	$\top$
Structured Decomposable						$\top$	$\top$
Deterministic			$\top$		$\top$	$\top$	$\top$
Strong Determinism						$\top$	$\top$
Smooth						$\top$	$\top$
Flat		$\top$	$\top$				
Query Type							
Consistency (CO)	NP-c	coNP-c	polytime	polytime	polytime	polytime	polytime
Model Enumeration (ME)	NP-c	poly-delay	poly-delay	poly-delay	poly-delay	poly-delay	poly-delay
Model Counting (CT)	#P-hard	#P-hard	#P-hard	#P-hard	polytime	polytime	polytime
Equivalence (EQ)	coNP-c	coNP-c	coNP-c	coNP-c	coNP-c	polytime*	polytime†
Conditioning	polytime	polytime	polytime	polytime	polytime	polytime	polytime
Forgetting	coNP-c.	coNP-c.	coNP-c.	coNP-c.	coNP-c.	polytime	NP-hard
Boolean Combination	polytime	polytime	polytime	polytime	polytime	polytime	polytime

$*$: SDD equivalence: polytime when compressed with a fixed vtree.

†: OBDD equivalence: polytime with fixed variable order.

4.2.3 Disjunctive Normal Form (DNF)

4.2.3.1 Definition and Properties

Disjunctive Normal Form (DNF) is a classical representation language for propositional theories. A DNF formula is a disjunction of terms, with each term being a conjunction of literals (variables or their negations). Formally, a DNF formula has the form ($T_1 \vee T_2 \vee \cdots \vee T_m$), where each ($T_i = l_{i1} \wedge l_{i2} \wedge \cdots \wedge l_{ik}$), with l_{ij} — a literal. DNF is a subset of NNF and more specifically, a Flat Negation Normal Form (f-NNF). It is universal: any propositional theory has a DNF representation. DNF is not canonical; equivalent functions may have different DNF syntactic forms. Every DNF formula is also a DNNF but is not in general deterministic (and thus not always a d-DNNF).

4.2.3.2 Construction

To construct a DNF, rewrite the propositional theory as a disjunction of conjunctions of literals, with each term representing a possible partial assignment that satisfies the theory. Mechanical conversion from other forms (such as CNF) to DNF can require exponential space in the worst case. Applications DNF appears in:

1. Model-based diagnosis, where explicit representation of models is useful for enumerative reasoning.

2. Knowledge compilation pipelines as a source or target language and as an intermediary form in bottom-up compilation to tractable representations.

3. Problems requiring efficient model enumeration, such as explanation generation or solution space exploration.

4.2.3.3 Compilers and Implementations

4.2.3.3.1 Bottom-Up Compilation from DNF to OBDD/ SDD Each DNF term (conjunction of literals) is individually compiled into the target structure (OBDD or SDD). Terms are then combined using the “Apply” (disjunction) function, which is polynomial-time in the size of the operands.

1. OBDD Implementations: The CUDD package is a widely used library supporting OBDD construction and manipulation, including efficient Apply operations.

2. SDD Implementations: The SDD package, by the authors of SDD, is the primary implementation for Sentential Decision Diagrams, fully supporting bottom-up compilation and ”Apply”.

4.2.3.3.2 Top-Down Compilation Compilation algorithms initially designed for CNF, such as Decision- SDD compilers, can also be used directly on DNF input. These compilers operate recursively using principles from SAT solving, caching, and structural decomposition.

1. Actual Implementations: The SDD package and other SAT-based compilers (e.g., c2d for DNNF compilation) can process DNF input, although their performance is generally tuned for CNF.

No sources identify compilers specific to DNF-to- CNF conversion or tailored DNF-to-DNF simplification outside of general boolean function minimization algorithms (such as Quine-McCluskey or Espresso), which are not the focus of knowledge compilation pipelines.

4.2.3.4 Polynomial-Time Queries and Complexities

4.2.3.4.1 Consistency (CO) ($O(|\Delta|)$). Satisfiability is determined by checking that at least one term contains no complementary literals.

4.2.3.4.2 Model Enumeration (ME) Polynomial delay per model ($\mathcal{O}(|T_i|)$) per model for term (T_i). All models can be enumerated efficiently by expanding the terms.

4.2.3.4.3 Others All other standard queries (e.g., Validity [VA], Clausal Entailment [CE], Model Counting [CT], Equivalence [EQ], etc.): Intractable unless ($P = NP$) or $\#P = FP$ (e.g., Model Counting is $\#P$-hard).

1. Validity (VA): co-NP-complete;

2. Clausal Entailment (CE): co-NP-complete;

3. Model Counting (CT): #P-hard.

DNF is a flat, non-canonical, universal language supporting tractable consistency checking and model enumeration. It is commonly used as a source or intermediate representation in compilation pipelines targeting OBDD, SDD, or related languages, with mainstream libraries like CUDD and the SDD package implementing practical bottom-up compilation from DNF. All other semantic queries, including entailment and model counting, are computationally intractable.

Despite the aforementioned intractability, however, DNF find a unique place in PRA analysis. A pure Even-Tree Diagrams can be represented as sum-or-products boolean formulas.

4.2.3.5 Event Tree Structures as Sum-Product Networks

Consider a specific branch ω_j leading to the end-state X_j. By definition, ω_j occurs if and only if:

1. The initiating event I happens: $i = 1$.

2. For each functional event F_k, the branch specifies a particular outcome (success or failure). Suppose ω_j includes successes for some subset of indices $\alpha \subseteq \{1, \ldots, n\}$ and

failures for the complementary indices. We can write this as:

$$\bigwedge_{k\in\alpha}\big(f_k^{\text{succ}}=1\big) \quad\wedge\quad \bigwedge_{k\notin\alpha}\big(f_k^{\text{fail}}=1\big).$$

Hence, the branch event ω_j is logically equivalent to a single *product term*:

$$\omega_j \;\equiv\; \big(i=1\big) \;\wedge\; \bigwedge_{k\in\alpha}\big(f_k^{\text{succ}}=1\big) \;\wedge\; \bigwedge_{k\notin\alpha}\big(f_k^{\text{fail}}=1\big). \tag{4.1}$$

In standard Boolean notation, each literal (e.g., f_k^{succ}) is a variable that can be 0 or 1, and the branch is the $\wedge$ (AND) of those variables. An event tree describing all possible outcomes from I and the subsequent functional events can be viewed as the union (logical OR) of its disjoint branches:

$$\Omega \;=\; \omega_1 \;\cup\; \omega_2 \;\cup\; \cdots \;\cup\; \omega_m.$$

In Boolean terms, this is the $\vee$ (OR) of the product terms corresponding to each branch:

$$\Omega \;\equiv\; \omega_1 \;\vee\; \omega_2 \;\vee\; \cdots \vee\; \omega_m. \tag{4.2}$$

Substituting each branch's conjunction form (as in Eq. (4.1)) into Eq. (4.2) yields:

$$\Omega \;=\; \Big[i \;\wedge\; \prod_{k\in\alpha_1} f_k^{\text{succ}} \;\wedge\; \prod_{k\notin\alpha_1} f_k^{\text{fail}}\Big] \;\vee\; \Big[i \;\wedge\; \prod_{k\in\alpha_2} f_k^{\text{succ}} \;\wedge\; \prod_{k\notin\alpha_2} f_k^{\text{fail}}\Big] \;\vee\; \cdots$$

where each α_r is the set of functional events that succeed along branch r.

A standard DNF (SoP) expression in Boolean algebra is

$$\big(\text{literal}_1 \;\wedge\; \text{literal}_2 \;\wedge\; \cdots\big) \;\vee\; \big(\text{literal}'_1 \;\wedge\; \text{literal}'_2 \;\wedge\; \cdots\big) \;\vee\; \cdots$$

Each term in the sum (OR) is a logical AND of literals (variables or their negations). Comparing with Eq. (4.2), we see that an event tree is exactly a disjunction of terms, each term being a conjunction of the initiating event i (set to 1) and the success/failure indicators for each F_k. Since any negation can be encoded by stating whether F_k is succ ($f_k^{\text{succ}} = 1$) or fail ($f_k^{\text{fail}} = 1$), the entire event tree Ω is in DNF:

$$\Omega \;=\; \bigvee_{j=1}^{m} \Big[\bigwedge_{\ell \in \Lambda_j} (\text{appropriate literal}) \Big].$$

4.2.3.6 Tractability of Event Trees

Within SP−networks (sum-product networks), a sum gate provides a weighted sum of child distributions, whereas a product gate factorizes them. An event tree can be cast as an SP−network by feeding each branch's literal probabilities into product gates (one per branch), then summing over all branches with a sum gate. Once constructed, evaluating the resulting SP−network at a specific configuration $\mathbf{x}$ or marginalizing out some of the variables is linear in the size of that network. Nevertheless, the tractability of event trees (and their circuit representations) heavily depends on their size and structure. We summarize several key considerations below:

1. *DNF size grows exponentially.*

   Suppose an event tree includes n functional events, each of which can succeed or fail. In the worst case, enumerating *all* possible outcome branches (i.e. each success/failure pattern) yields up to 2^n conjunction terms. Hence, the disjunctive normal form (DNF) representation can become exponentially large. Computing or marginalizing probabilities over such a large DNF may become prohibitively expensive if n is large enough.

2. *Evaluation linear in network size.*

   Even though the DNF itself may blow up exponentially, once the event tree is translated into an SP−network, key inference tasks (such as evaluating it at a configuration or marginalizing over certain variables) proceed in time linear in the *compiled network* *size.* That said, if the underlying network has already reached exponential size in the number of events, the linear-time evaluation does not necessarily improve the overall worst-case complexity.

3. *Approximations abound.*

   In practice, analysts often employ approximations to keep event trees tractable. One possibility is *decomposability*, a core principle behind tractable probabilistic circuits whereby each product gate operates on disjoint sets of variables. If the system decomposes (e.g. different safety barriers protect disjoint sets of equipment), one can evaluate probabilities without enumerating all branches. Another common approximation is to prune paths with very low probabilities or ignore paths that only negligibly contribute to the overall risk.

Fully enumerated event trees, regardless of being interpretable as DNF/SP networks, trade tractability for expressivity. The intuitive branching structure and conditional probability assignments make event trees easy to interpret. PRA analysts can read off and reason about the high-level scenario decomposition, incorporate domain knowledge, and analyze each branch explicitly. If the number of critical functional events is moderate, enumerating all branches remains tractable. As the depth and breadth of the tree grow, any brute-force probability computation over such a large DNF/SoP circuit is equally exponential in the worst case. Even though SP−networks offer efficient linear-time evaluation with respect to the circuit size, the underlying circuit itself may have size exponential in n.

4.2.4 Conjunctive Normal Form (CNF)

Conjunctive Normal Form (CNF) is a specific subset of f-NNF. CNF further restricts this structure: a CNF formula is a conjunction of clauses, where each clause is a disjunction of literals. Thus, every CNF is an NNF where the formula has depth at most two: a top-level conjunction whose direct children are disjunctions of literals. Negations never apply to anything except individual variables.

Despite computational intractability of most queries, CNF is the dominant input language for SAT solvers, which employ highly optimized heuristics far surpassing brute-force complexity in practice. Many real-world verification, synthesis, and combinatorial search problems are encoded as CNF for this reason.

4.2.4.1 Key Properties

4.2.4.1.1 Flatness As a formula or DAG, a CNF has depth at most two. The root is a conjunction $\wedge$ whose direct children are disjunctions $\vee$ of literals (leaf nodes).

4.2.4.1.2 Simple-disjunction All disjunctions operate directly on literals–there are no nested or compound subformulas inside disjunctions.

4.2.4.1.3 Closure under conjunction The conjunction of two CNF formulas yields another CNF formula.

4.2.4.1.4 Non-uniqueness CNF is not canonical. Logically equivalent formulas can have very different CNF representations due to clause or literal redundancy, reordering, or other syntactic differences.

4.2.4.2 Construction and Compilation

Constructing CNF from arbitrary Boolean formulas using Tseitin encoding or distributive laws takes ($\mathcal{O}(|\varphi|)$) time and space, potentially with introduction of auxiliary variables for compactness. Any propositional formula can be converted to an equisatisfiable CNF using methods such as Tseitin encoding, which can be performed in ($\mathcal{O}(|\varphi|)$) time and size for a formula (φ). This transformation ensures that the original and resulting CNF formulas have the same satisfiability, though logical equivalence is not guaranteed.

4.2.4.3 Query Complexity

4.2.4.3.1 Implicant Query (IM) Checking whether a term implies a CNF formula can be done in $(\mathcal{O}(|\text{term}| + |\text{CNF}|))$ time.

4.2.4.3.2 Satisfiability (Consistency, CO) Determining satisfiability of a general CNF formula is NP-complete; the best algorithms run in time $(\mathcal{O}(2^n))$ in the worst case, where (n) is the number of variables, though modern SAT solvers perform much better in practice.

4.2.4.3.3 Validity (VA) Checking whether a CNF is a tautology is co-NP-complete. No polynomial-time algorithm is known unless $P = NP$.

4.2.4.3.4 Prime Implicant Generation Extracting a single prime implicant from a known model (an assignment satisfying the CNF) can be performed in polynomial time. This process involves iteratively attempting to remove each literal from the model and checking if the CNF remains satisfied. Each removal can be checked in time linear in the size of the CNF, and all removals together give a total complexity of $(\mathcal{O}(|\text{CNF}| \cdot |M|))$, where M is the model. With efficient algorithms, this task can be done in linear time.

1. Finding any model or implicant: Equivalent to SAT, and thus NP-complete.

2. Enumerating all prime implicants: Exponential time in the worst case; the number of prime implicants can be exponential in formula size.

3. Recognizing essential prime implicants: NP-complete.

4.2.4.3.5 Model Counting (CT) Counting the number of satisfying assignments is $\#P$-complete. For a CNF with primal treewidth (w) and (n) clauses, model counting via dynamic programming on a tree decomposition can be done in $(\mathcal{O}(n2^w))$ time when the decomposition is given. For a CNF with incidence treewidth (w) and (N) tree decomposition nodes, counting can be done in $(\mathcal{O}(2^w(kd+\delta)N))$, where (d) is maximum variable degree, (δ) is multiplication time.

4.2.4.3.6 Model Enumeration (ME) Enumerating all models is not feasible in polynomial time in general, due to potentially exponential output size.

4.2.4.3.7 Clausal Entailment (CE), Equivalence (EQ), Sentential Entailment (SE) All are co-NP-complete (or worse) in general for CNF.

4.2.4.4 CNF as a Source for Knowledge Compilation

CNF serves as the standard input for compilation into tractable reasoning languages:

4.2.4.4.1 CNF to DNNF With given decomposition tree of width (w) and (n) clauses, can be compiled in $(\mathcal{O}(nw2^w))$ time and space. Complexity is singly exponential in treewidth and linear in formula size when width is bounded.

4.2.4.4.2 CNF to d-DNNF Using tools like c2d and DSHARP, for CNF with incidence treewidth (k) and size (n), can be compiled into DNNF of size $(\mathcal{O}(2^k n))$.

4.2.4.4.3 CNF to SDD For a CNF with (n) variables and vtree of width (w), bottom-up compilation yields SDD of size $(\mathcal{O}(n2^w))$, and can be performed in $(\mathcal{O}(nw))$ time if the vtree is fixed.

4.2.4.4.4 CNF to OBDD Top-down approaches with caching result in size and time $(\mathcal{O}(2^{\text{pathwidth}}))$ for OBDD, where pathwidth is a width measure related to treewidth.

4.2.4.5 Compilers

1. DNNF/d-DNNF: c2d, dsharp (output size $(\mathcal{O}(2^{\text{width}}n))$)

2. RoBDD: CuDD, BuDDy (output size $(\mathcal{O}(2^{\text{pathwidth}}))$)

3. SDD: SDD package (output size ($\mathcal{O}(n2^w)$))

4.2.5 Decomposable Negation Normal Form (DNNF)

Decomposable Negation Normal Form (DNNF) is a central target language in knowledge compilation, representing a significant refinement of Negation Normal Form (NNF). In NNF, every propositional sentence is represented as a directed acyclic graph (DAG): leaves are labeled by literals (variables or their negations) or by the constants `true`/`false`, while internal nodes are labeled by conjunction ($\wedge$) or disjunction ($\vee$) operations. NNF allows unrestricted subformula structure as long as negations only appear at the literal level, but on its own does not provide tractability for key reasoning tasks unless P = NP.

4.2.5.1 Definition of DNNF and Related Forms

A formula is in **DNNF** if it is in NNF and satisfies the *decomposability* property: at every conjunction ($\wedge$-node), the conjuncts mention disjoint sets of variables. Formally, if a conjunction node has children $\varphi_1, \ldots, \varphi_m$, then $\mathrm{Var}(\varphi_i) \cap \mathrm{Var}(\varphi_j) = \emptyset$ for all $i \neq j$. This property enables efficient independent evaluation of circuit branches.

- **Deterministic DNNF (d-DNNF):** Adds the *determinism* property. At every disjunction ($\vee$-node), disjuncts are mutually contradictory (i.e., the conjunction of any two child subcircuits is inconsistent). Determinism enables additional tractable queries, such as model counting.

- **Structured DNNF / Structured d-DNNF:** These subclasses further restrict DNNF/d-DNNF by requiring that decompositions reflect a hierarchical structure (typically enforced by a vtree specifying the variable partitioning at each conjunction or disjunction). This *structured decomposability* allows additional tractability (notably, certain transformations), and forms the basis for languages like Sentential Decision Diagrams (SDDs), which are strict subsets of structured d-DNNF.

4.2.5.2 Construction and Compilation

Compiling an arbitrary propositional formula, often given in CNF or DNF, into DNNF or d-DNNF is a central algorithmic task. For CNF inputs, one algorithm uses a decomposition tree (related to variable treewidth). Given n clauses and treewidth w, d-DNNF can be compiled in time and space $O(nw2^w)$; thus, for bounded treewidth, linear-sized d-DNNF representations can be obtained in linear time. Practical compilers include `C2D`, `DSHARP`, and `d4`. `DSHARP`, leveraging #SAT technology, frequently exceeds the speed of `C2D` while producing similarly sized d-DNNF. OBDD representations for propositional theories can be converted into equivalent DNNF in linear time relative to OBDD size.

4.2.5.3 Succinctness

Succinctness compares the minimum size of representations of the same function across languages. The succinctness ordering (strict, unless the polynomial hierarchy collapses) is:

$$\text{DNNF} < \text{d-DNNF} < \text{FBDD} < \text{OBDD} < \text{CNF/DNF}$$

That is:

- DNNF (and its subclasses) are strictly more succinct than OBDDs.
- SDDs sit as a strict subset of structured d-DNNF.
- DNNF is generally more succinct than FBDDs, which are more succinct than OBDDs.
- CNF and DNF generally remain exponentially sized compared to DNNF for many functions.

Smooth deterministic DNNF (sd-DNNF) is as succinct as d-DNNF.

4.2.5.4 Supported Queries and Complexities

A principal utility of DNNF is to enable several queries in time polynomial to circuit size. The main queries and their tractability status for DNNF and derivatives:

- **DNNF:**
  - **Consistency (CO):** Polynomial time
  - **Clausal Entailment (CE):** Polynomial time
  - **Model Enumeration (ME):** Polynomial time; for sd-DNNF, output-linear time
- **d-DNNF/structured d-DNNF:**
  - All above, plus:
  - **Validity (VA):** Polynomial time
  - **Model Counting (CT):** Polynomial time (linear for sd-DNNF)
  - **Model-based Diagnosis:** Minimum-cardinality diagnosis, etc., in polynomial time
  - **Implicant Checking (IM)**, **Model Minimization:** Polynomial time
  - **Equivalence Testing (EQ):** Not known to be polynomial time for d-DNNF, unlike OBDD

4.2.5.5 Empirical Benchmarks

Empirical results show a clear advantage for DNNF and d-DNNF in both size and compilation efficiency versus OBDD on relevant AI tasks:

- Diagnoses compiled into DNNF are often orders of magnitude smaller than those into OBDD.

- Compilation time is generally faster with DNNF compilers (`DSHARP` is up to 27 times faster on average than `C2D`, and both outperform OBDD compilation for relevant model-based diagnosis inputs).

- Diagnostic and enumeration queries are more efficient on DNNF than OBDD, due to reduced circuit size and higher tractability.

DNNF and its subclasses, notably d-DNNF and sd-DNNF, provide a foundational set of tractable languages in knowledge compilation, balancing high succinctness with strong support for key inference queries. Their compilation is practical for many instances, and empirical results show clear advantages over OBDDs. SDDs and structured d-DNNFs offer further tractable transformations at some cost in succinctness.

4.2.6 Decision Diagrams

Decision diagrams provide a powerful, directed-graph-based representation of logical or Boolean functions. Their roots can be traced back to branching program ideas explored by Lee and Akers in the 1950s–1970s, but major refinement and widespread adoption occurred after Bryant's seminal work on *Ordered Binary Decision Diagrams (OBDDs)* in 1986. In reliability analysis, formal verification, and combinational circuit design, decision diagrams frequently offer more computationally tractable methods than naïve enumeration of all input patterns.

This section introduces the basic concepts of *Binary Decision Diagrams (BDDs)* and *Zero-Suppressed Decision Diagrams (ZDDs)*, along with the special class of *Ordered* and *Reduced* BDDs that guarantee a canonical (unique) form under fixed conditions. We emphasize:

- The structure and interpretation of BDDs as directed acyclic graphs (DAGs).

- The notion of an *ordered* BDD, imposing a strict arrangement on variable testing.

- Techniques for *reducing* BDDs into smaller yet equivalent graphs by merging or removing redundant parts.

- The main principles of Zero-Suppressed Decision Diagrams, designed for efficiently encoding sparse sets or combinatorial families.

Earlier, we saw that *event trees* and *fault trees* can be merged into a single directed acyclic graph to represent complex system dependencies. BDDs and ZDDs, by contrast, focus more narrowly on Boolean functions, providing specialized node-splitting and merging operations to systematically capture logical behavior. Despite differing motivations, both families of DAG-based representations benefit from the avoidance of cycles and the ability to encode large models in a structured form.

4.2.6.1 Binary Decision Diagrams (BDD)

Let $f : \{0,1\}^n \to \{0,1\}$ be an n-variable Boolean function. A Binary Decision Diagram (BDD) is a directed acyclic graph whose internal nodes represent decisions on a single Boolean variable, and whose terminal (sink) nodes represent constant outputs (0 or 1).

Definition 2 (Binary Decision Diagram)**.** *A Binary Decision Diagram for f is a tuple $B = \big(N, n_0, V, E, T\big)$ with the following components:*

1. *N is a finite set of nodes, partitioned into* internal *nodes and* terminal *nodes.*

2. *$n_0 \in N$ is the* root node*, where evaluations begin.*

3. *$V = x_1, x_2, \ldots, x_n$ is the set of Boolean variables associated with the internal nodes.*

4. *$E \subseteq N \times 0, 1 \times N$ is the edge set. Each internal node u has two labeled edges, $(u, 0, v_0)$ and $(u, 1, v_1)$, indicating the next node in the diagram if $x_i = 0$ or $x_i = 1$ at node u.*

5. *T is a mapping that assigns the value* 0 *or* 1 *to each terminal node of B.*

For any input $a = (a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ one identifies a unique path from n_0 to a terminal node by at each internal node following the edge labeled by the tested variable's value in a. The value of $f(a)$ is given by the terminal node reached, as encoded by T.

Interpretation. Each internal node corresponds to a variable test: if the variable is 0 (i.e. false), follow the *0-edge*, and if it is 1 (true), follow the *1-edge*. Eventually, one reaches a sink node labeled false = 0 or true = 1.

4.2.6.1.1 Example. For the three-variable function $f(a, b, c) = a \wedge \big(b \vee c\big)$, Figure 4.4 shows a small BDD. Each circular node tests one variable a, b, or c; the dashed and solid edges denote the 0- and 1-branches, respectively. Terminal nodes (squares) contain a 0 or 1 label.

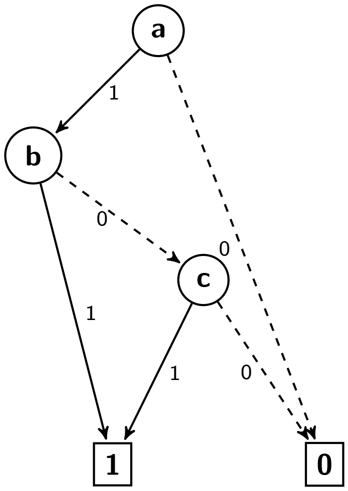


Figure 4.4: A Binary Decision Diagram (BDD) for $f(a, b, c) = a \wedge \big(b \vee c\big)$.

In practice, BDDs may experience large variations in size depending on how variables are tested as one traverses the graph. The *ordered* and *reduced* variants of BDDs are especially important, as they yield canonical forms for fixed variable orderings.

Definition 3 (Ordered Binary Decision Diagram (OBDD))**.** *An Ordered Binary Decision Diagram (OBDD) imposes a strict ordering π on the variables $x_1, \ldots, x_n$. For every path from the root to a terminal node, if the path encounters variables x_i and x_j, then x_i is tested before x_j whenever $i < j$ with respect to π. Equivalently, no path may test a higher-indexed variable and later test a lower-indexed one.*

OBDDs are also known as *read-once branching programs with an ordering restriction.* Bryant showed that under a particular variable order, the representation is often more compact than arbitrary BDDs and that many operations (e.g., equivalence checking, conjunction, disjunction) can be carried out efficiently.

Definition 4 ([Reduced Ordered Binary Decision Diagram (RoBDD))**.** *An OBDD is said to be* reduced *if it contains no isomorphic subgraphs and no node whose 0- and 1-branches lead to the exact same child. Equivalently, one applies two reduction rules:*

1. ***Elimination Rule:*** *If, for a given node v, the 0-edge and 1-edge both point to the same successor, remove v and connect its incoming edges directly to that successor.*

2. ***Merging Rule:*** *If two distinct nodes u and v test the same variable and have identical 0- and 1-successors, merge them into a single node.*

A Reduced Ordered Binary Decision Diagram (RoBDD) *respects a global variable order π and has been minimized via these rules.*

Canonical Representation. One of the principal advantages of ROBDDs is that, for a fixed variable ordering, every Boolean function has a unique representation. Consequently, checking whether two functions are identical reduces to testing whether their ROBDDs coincide as node- and edge-labeled graphs.

Theorem 3 (Canonical Form of RoBDDs)**.** *Let π be a fixed ordering on the variables $x_1, \ldots, x_n$. Then for any Boolean function f of n variables, its reduced OBDD with respect to π is unique.*

A direct consequence is that striving for reduced ordered forms both shrinks redundant structure and supports robust equivalence checks.

4.2.6.2 Zero-Suppressed Decision Diagrams (ZDD)

For certain applications, notably combinatorial itemset enumeration and other *sparse* set representations, Zero-Suppressed Binary Decision Diagram (ZDD) can be more compact than standard BDDs. Although ZDDs adhere to similar principles of node-based variable testing, they selectively *omit* many zero-branches that do not yield new information.

4.2.6.2.1 Key Distinctions. While ZDDs also enforce an ordering and can be reduced via isomorphism checks, the core difference lies in the zero-suppression mechanism:

- If following a 0-edge provides no meaningful distinction in the final outcome, that 0-edge and its corresponding node are pruned.
- The 1-branches are retained but merged where possible, much as in RoBDDs.

By removing portions of the diagram where ”nothing interesting” (i.e. no new sets or subsets) occurs, the diagram remains compact.

4.2.6.2.2 Illustrative Example Revisiting $f(a, b, c) = a \wedge \left(b \vee c\right)$ from above, Figure 4.5 sketches a plausible ZDD. Note here:

- Node (a) splits into a 0-edge that immediately goes to a node (or directly to a `0`-terminal) that is pruned if it carries no unique set representation.
- The 1-edge leads to further variable tests (b or c), but many 0-branches are again suppressed if they do not alter the final outcome distinct from an already-represented path.

In general, ZDDs apply much the same merging rules as RoBDDs and can yield similarly unique structures for a given variable order. They tend to excel in representing large but sparsely populated families of subsets (e.g., all minimal cut sets in a reliability system) because superfluous 0-edges are systematically suppressed.

4.2.6.3 Probabilistic Sentential Decision Diagrams (PSDD)

A Probabilistic Sentential Decision Diagram (PSDD) is a tractable representation for a probability distribution over a set of propositional variables subject to logical constraints. In

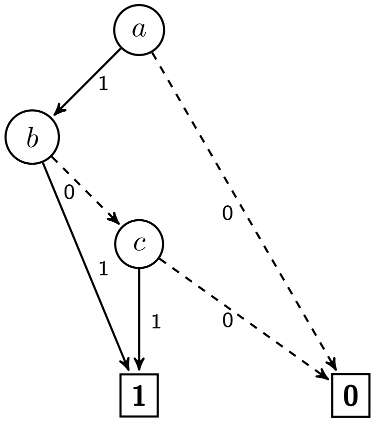


Figure 4.5: A Zero-Suppressed Binary Decision Diagram (ZDD) for $f(a,b,c) = a \wedge \big(b \vee c\big)$. Many zero-branches are pruned.

essence, a PSDD is a parameterized Sentential Decision Diagram (SDD) in which each node is assigned a well-defined local distribution. By construction, the PSDD's global distribution respects a given *base theory* (i.e., a propositional formula representing constraints), assigns zero probability to every assignment violating that theory, and factors the remaining assignments according to a hierarchical decomposition.

Definition 5 (Normalized SDD)**.** *Given a* vtree*, v, over variables $X_1, ..., X_n$, an normalized SDD over T is defined recursively as follows:*

1. *A* terminal *node is a literal X_i or $\neg X_i$*

2. *At an* internal vtree node *with left subtree V_l and right subtree V_r, a normalized SDD is a finite disjunction:*

$$\bigvee_{i=1}^{k} (P_i \wedge S_i)$$

where:

   (a) *For every i, P_i is a normalized SDD over the variables in V_l, and S_i is a normalized SDD over the variables in V_l.*

   (b) *The set $\{P_1, \ldots, P_k\}$ forms a partition of the space over V_l; that is, $\forall i \neq j$: $P_i \wedge P_j = \bot$ (mutually exclusive) and $\bigvee_{i=1}^{k} P_i = \top$ (exhaustive).*

(c) Each S_i is distinct (no two are equivalent).

(d) Each prime-sub pair (P_i, S_i) is compressed: distinct primes never associate to equivalent subs and vice versa (no redundancy).

Once the normalized SDD is fixed, one may introduce continuous parameters to obtain a PSDD. These parameters, in effect, turn each decision node into a *local mixture* of its prime components, while terminal nodes over variables become Bernoulli distributions.

Definition 6 (PSDD Syntax)**.** *Let n be an SDD node normalized for a vtree node v.*

- *If n is a terminal node:*

  1. *If it encodes a literal (e.g. X, $\neg X$) or the constant $\bot$, then its probability is fixed implicitly (e.g. $\neg X$ yields $\Pr(\neg X) = 1$, $\Pr(X) = 0$ for that node alone).*

  2. *If it is the constant $\top$ and v corresponds to some variable X, then we assign a parameter $\theta \in (0, 1)$ indicating $\Pr(X)$ at this node.*

- *If n is a decision node with k elements $[(p_1, s_1), \ldots, (p_k, s_k)]$, we assign nonnegative parameters $\theta_1, \ldots, \theta_k$ such that $\sum_{i=1}^{k} \theta_i = 1$. Furthermore, if $s_i = \bot$, then θ_i must be zero (no probability is allotted to a sub whose base is unsatisfiable).*

The resulting parameterized structure is called a PSDD.

Each node n in a PSDD induces a local distribution $\Pr_n(\cdot)$ on the variables of the vtree node n is normalized for. At a decision node, the probability of a complete assignment is given by multiplying:

1. The probability that we "choose" a particular prime p_i, labeled by θ_i.

2. The probability contributed by prime node p_i on its variables.

3. The probability contributed by sub node s_i on its (disjoint) variables.

Summing across all primes yields $\Pr_n$ at that node. By construction, the *root* node's distribution $\Pr_r$ then covers all variables and zeroes out any assignments that do not satisfy the base theory.

Theorem 4 (Base Property [26])**.** *If a PSDD node n is normalized for vtree node v, then* $\Pr_n(x) = 0$ *whenever x does not satisfy the SDD sub-formula* $[n]$*. At the root node r,* $\Pr_r(x) > 0$ *only if x satisfies the entire theory.*

4.2.6.3.1 Parameter Semantics. A key property of PSDDs is that each parameter θ_i can be interpreted *locally* as a conditional probability given the *context* of the decision node. Formally, if node n has context γ_n (i.e., the partial assignment implied by traversing the SDD from the root to n), then:

$$\theta_i = \Pr\big([p_i] \bigm| [\gamma_n]\big)$$

where $[, p_i]$ is the logical content of prime p_i. This ensures that local parameters align with global $\Pr(\cdot)$ in a transparent, compositional way.

4.2.6.3.2 Context-Specific Independence. Due to the vtree-based factorization, PSDDs capture rich *context-specific independences* [21]. At high level, once we know the node's context (which is a partial assignment or formula), certain subsets of variables become conditionally independent of the rest. These independence statements can be read directly from the PSDD structure, generalizing common conditional-independence ideas in Bayesian or Markov networks.

4.2.6.3.3 Inference and Tractability. A central advantage of PSDDs is that computing $(\Pr_r(e)$ for any evidence e can be done in time linear in the size of the PSDD. This incremental algorithm proceeds bottom-up through each node, locally aggregating evidence contributions and summing accordingly. Moreover, once node-level evidence statistics are available, one can also efficiently compute single-variable or pairwise marginals using a second top-down pass.

4.2.6.3.4 Parameter Learning under Complete Data. Another appealing property is that *maximum-likelihood* parameters can be determined in closed form when every training example is a complete assignment of all variables. Specifically, if a PSDD node n with context γ_n has elements (p_i, s_i), one sets

$$\theta_i = \frac{\text{number of data points satisfying both } \gamma_n \text{ and } p_i}{\text{number of data points satisfying } \gamma_n}.$$

Parallel rules apply for terminal nodes representing $\top$. Because sub-contexts $\gamma_n \wedge p_i$ are pairwise disjoint, it suffices to tabulate data counts for each feasible sub-context. The outcome is a simple frequency-based update analogous to parameter estimation in Bayesian networks, yet here it respects the underlying SDD constraints exactly.

For any propositional distribution (and chosen vtree), there exists a corresponding PSDD whose root distribution matches it exactly. Furthermore, if the PSDD is kept *compressed* (meaning no redundant substructures), this representation is *unique* up to isomorphic details [26]. Thus, PSDDs can serve as canonical forms for distributions under logical constraints.

In contrast to classical graphical models, PSDDs operate at the confluence of tractable *Boolean* structure (via SDDs) and probability theory. They explicitly encode zero-probability assignments (via the SDD base) while ensuring all positive assignments factor through the decision nodes. Their parameter semantics aligns each local weight with a well-defined global conditional probability. In addition, closed-form parameter learning is possible in the complete-data setting. Hence, PSDDs provide a principled, canonical choice for modeling distributions when the domain is governed by complex logical constraints.

4.3 Queries

The primary choice that dictates the selection of target languages in PRA is availability of tractable/executable queries for a given language. The two most important query types that are of interest to PRA are Model Counting (CT) and Model Enumeration (ME).

4.3.1 Model Counting

Model Counting, a.k.a #SAT-problem, is a classic problem in Computability Theory and VLSI (Very Large Scale Integration) analysis. It strive to answer the following question about a given boolean expression (theory): "How many valid true/false variable assignments (models) satisfy the boolean expression?" This problem is central to PRA, as it the compuation algorithm behind Weighted Model Counting approach used for computing probabilities of top events in Fault Trees and End States in Even Sequence Diagrams.

Importance of this query has largely determined the relative scarcity of underlying implementations of PRA software. Tractability of this query in OBDD family of target languages is primarily reason for popularity of this data structure among PRA software vendors. However, despite the its perceived NP-complexity, CNF-based #SAT-solvers can also be used, due to their efficient implementations and long-running research. Finally, with new research showing still flowing, SDDs, identify themselves as a promising alternative due to their improved succinctness.

4.3.2 Model Enumeration

Model enumeration is arguably even more important query for PRA, as it responsible for implementations exhaustive Minimal Cutset search. This is one of the most intractable algorithms allowing for only polynomial-time delay between successive solution, as the primary determining complexity factor is exponential "blow-up" of the number of cutsets in the first place. Few implementations, offer truly efficient and tractable solutions, with CNFs and ROBDDs yet again taking the spotlight.

Part II

Identifying Gaps

Chapter 5

Current State of PRA Software

Probabilistic Risk Assessment (PRA) has developed steadily since the 1970s, in parallel with increasing model complexity and advances in computing. Early efforts, including those supporting the Reactor Safety Study (WASH-1400), made clear how hardware constraints dictated the scope and fidelity of probabilistic analyses. Despite steady gains in compute capability, from mainframes to Personal Computer (PC)s to modern clusters and cloud services, PRA software has largely adopted incremental improvements, leaving persistent gaps in scalability, usability, and methodological transparency.

Several codes emerged in rapid succession in the field's early years. PREP and KITT automated fault-tree evaluation via Monte Carlo and deterministic methods; Method for Obtaining Minimal Cut Sets (MOCUS) and SIGPI advanced Minimal Cut Sets (MCS) generation and system probability calculations. MODULE, RISKMAN, and Integrated Reliability and Risk Analysis System (IRRAS) added capabilities for importance measures, uncertainty handling, and more extensive PC-based analysis. By the late 1980s, Computer Aided Fault Tree Analysis System (CAFTA), Systems Analysis Programs for Hands-On Integrated Reliability Evaluations (SAPHIRE), and RiskSpectrum consolidated many of these functions and became widely adopted.

Contemporary models, covering fire PRA, seismic events, multi-unit sites, and other

Table 5.1: An incomplete list of current and legacy Probabilistic Risk Assessment tools.

Tool	Description	Maintainer	License
PREP	MCS via Monte Carlo/deterministic methods [82].	Legacy	-
KITT	Calculates probabilities [81].	Legacy	-
MOCUS	Method for Obtaining Minimal Cut Sets [47].	Legacy	-
MICSUP	Minimal Cut Sets Upward [25].	Legacy	-
MODULE	Importance measures and uncertainty analysis [53].	Legacy	-
RISKMAN	First full suite of PRA modeling tools [84, 83].	ABS Consulting	-
IRRAS	Precursor to SAPHIRE [73, 74].	Legacy	-
SIGPI	Probabilistic performance of complex systems [56].	Legacy	-
Phoenix Architect	Windows-based PRA modeling tools, i.e. CAFTA [55, 23].	EPRI	Commercial
SAPHIRE	US NRC supported tool [79, 76, 77].	INL	Free to use
KIRAP	KAERI Integrated Reliability Analysis Code Package [52].	KAERI	-
RiskSpectrum	Windows-based PSA modeling tools [59].	RiskSpectrum	Commercial
@RISK	Formerly known as Palisade [62, 61].	Lumivero	Commercial
RiskA	Windows-based, limited availability [88].	FDS	-
XFTA	Crossplatform PSA quantification tools [69].	AltaRica	Free to use
SCRAM	Command-line risk analysis multi-tool [64].	OpenPSA	GPL 3.0
DeRisk	Uses Dynamic Uncertain Causality Graphs [90].	-	-
FTREX	Advanced Fault Tree quantification [46].	KAERI	Commercial
QRAS	Precursor to Trilith [49].	Legacy	Commercial
Trilith	Implements Hybrid Causal Logic (HCL) [85].	Prediction Tech	Commercial
HCLA	Web-based, implements HCL [70, 71].	UCLA GIRS	Commercial
OpenPRA	Web-based [36]	OpenPRA	MIT

external hazards, regularly exceed the capacity of legacy single-threaded workflows. Although High Performance Computing (HPC) techniques, distributed computing, and open-source ecosystems offer paths to greater scalability and interoperability, adoption is uneven. Barriers include commercial and institutional lock-in, limited automation for large-scale model manipulation, and continued difficulty conveying risk insights to non-specialists.

5.1 An Evolving Computing Landscape

In the 1970s and 1980s, PRA analyses typically ran as batch jobs on mainframes or minicomputers, with fault-tree quantification consuming hours or days. Early codes (e.g., PREP, KITT, and MOCUS) established viable proof-of-concept approaches—Monte Carlo methods, in particular—despite severe memory and processing limits. As desktop computing matured, PRA tools shifted toward locally installed PC software, reducing dependence on mainframes but inheriting new constraints such as storage limits, slower clock speeds, and constrained user interfaces.

By the end of the 1980s, PRA had expanded to include more intricate regulatory and external-event analyses. Integrated environments began to combine event-tree construction, fault-tree quantification, and uncertainty analysis within a single workflow. However, the prevailing software architectures remained oriented toward single-core execution and local memory.

Today, PRA models increasingly encode multi-hazard interactions (e.g., seismic with fire), site-wide resource couplings across multiple units, and broader parametric and epistemic uncertainties. Without systematic concurrency and distributed execution, quantification becomes prohibitively slow. The near-term requirement is clear: PRA software must fully leverage multicore and distributed platforms.

5.2 Persistent Limitations

5.2.1 Scalability

Although most PRA codes now run on modern operating systems, few scale cleanly to clusters or cloud environments. Even well-established tools (e.g., Phoenix Architect, SAPHIRE, XFTA) largely employ serial quantification with limited concurrency, though this is rapidly changing. Fire or seismic PRA models can include tens of thousands of basic events; tasks such as cut set generation and manipulation push memory demands into the gigabyte range and extend runtimes substantially. In the absence of robust parallelization and memory-aware data structures, analyses can stall or fail during quantification.

5.2.2 Model Development

Building and maintaining PRA models remains labor intensive. Flag-file edits, manual event-tree expansions, and ad hoc data updates can introduce inconsistent states, an acute concern for multi-hazard and dependency analyses. Extending coverage to additional external hazards (e.g., tornado missiles, external flooding, wildfire) often entails substantial re-modeling and is error-prone. While most tools provide GUIs for diagrammatic and tabular editing, the community continues to seek higher levels of automation that support version control, auditing, and systematic scenario extensions.

5.2.3 Dependency Analysis

Capturing dependencies at scale, particularly those involving human actions, remains challenging. Human Reliability Analysis (HRA) dependencies arise from operator interactions with evolving scenarios and are difficult to encode within classical fault trees [28, 27]. Few tools offer built-in methods for advanced HRA models or cross-system dependencies beyond standard common-cause failures. As a result, risk insights involving operator actions under multi-hazard conditions may rely on oversimplified assumptions.

5.2.4 Multi-Hazard, Multi-Unit Modeling

Modern applications increasingly require multi-unit site analyses. Standard PRA assumptions of unit independence often do not hold; resources such as power, control staff, and emergency cooling may be shared. Incorporating these couplings, along with combined external hazards, can yield combinatorial growth in scenario space. Naïve enumeration of unified hazard representations (e.g., site-level seismic and flooding profiles) can exhaust memory and degrade performance.

5.2.5 Communication of Risk Insights

As models and results grow more sophisticated, communicating those results and their uncertainties to non-PRA stakeholders remains difficult. User-facing interfaces have improved, but effective decision support, particularly for time-sensitive contexts requires intuitive dashboards, real-time updates, and responsive visualization. Closed-source suites often restrict extensibility, complicating the development of tailored analytics.

5.2.6 Transparency, Licensing and Community Support

The most widely used platforms (e.g., SAPHIRE, XFTA, CAFTA, RiskSpectrum) are closed-source or only partially accessible. This limits collaborative development and impedes optimization for multicore, cluster, or cloud settings, where concurrency-friendly data structures, partitioned workflows, and specialized libraries may require deep source-level changes. It also slows the adoption of modern practices (containerization, workflow automation, and HPC libraries), forcing reliance on special arrangements or workarounds.

Open-source efforts such as SCRAM [64] and OpenFTA [18] enable community-driven improvement. SCRAM, in particular, provides a flexible toolkit for fault-tree quantification, cut set manipulation, and simplified event-tree analysis.

5.3 Looking Forward

PRA practice is transitioning from single-threaded desktops to scalable, automated, and transparent infrastructures. In the near term, progress will rely on HPC-oriented frameworks that support parallel quantification, distributed uncertainty propagation, and memory-efficient representations of large correlated event spaces. Over longer horizons, emerging paradigms such as quantum computing and Fully Homomorphic Encryption (FHE) may enable secure and larger-scale analyses, though at present they are primarily research topics. Sustained advances will depend on integrating parallel computing, reducing manual model-development burden, representing multi-hazard and multi-unit interactions consistently, and delivering risk information that is interpretable and actionable for non-specialists.

Chapter 6

Developing Benchmark Models

6.1 Model Translation

Model representation in the PRA domain is encumbered by format fragmentation and proprietary development practices. A small group of stakeholders, each using specialized file structures, has little incentive to converge on a single standard. Proposing a new ”universal” format can inadvertently add yet another layer of fragmentation if adoption proves limited. Conversely, attempting a fully connected translation map among all existing formats is a significant undertaking, requiring both ongoing maintenance and broad community participation that may not materialize.

6.1.1 PRAcciolini: Translation without a Single Intermediate Representation

In view of these constraints, the short-term strategy is to create a system of lightweight interfaces under an open-source tool, referred to here as *PRAcciolini*, that interlinks existing translation tools without defaulting to a single pivot format. Rather than striving for every possible conversion path, the goal is to form what might be called a ”translation spanning tree”: a subset of connections that covers most practical needs while avoiding excessive overhead. This design allows each domain-specific library to retain its native parsing and exporting routines, with conversions performed only when required.

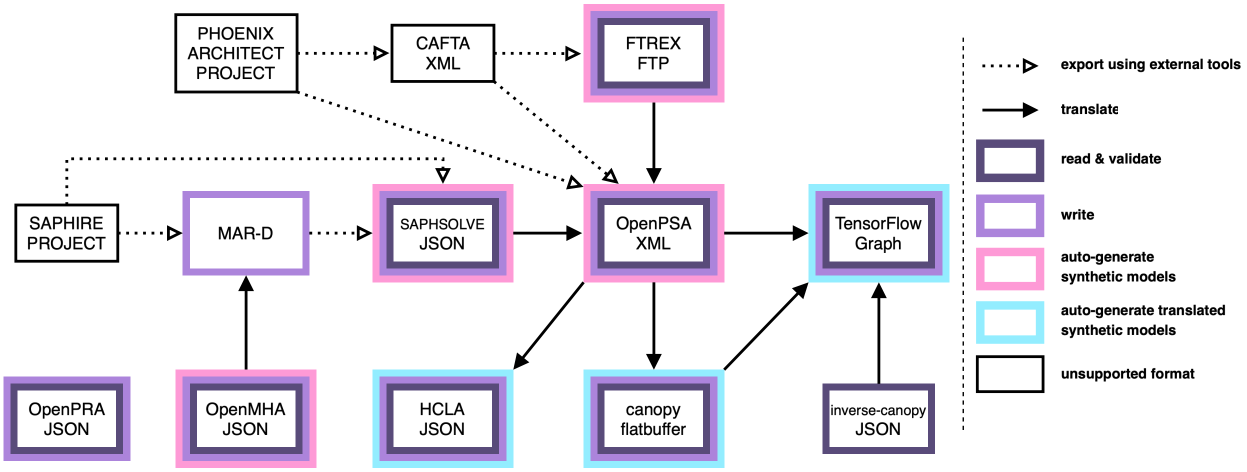


Figure 6.1: Example of supported model translations in PRAcciolini.

As shown in Figure 6.1, a CAFTA XML file can be transformed into FTREX FTP or SAPHSOLVE JSON, while MAR-D data can be republished as OpenPSA XML and potentially mapped to a TensorFlow compatible format. Round-trip testing (translating from one format to another and back again) verifies fidelity, ensuring that essential model information remains intact. By removing the need for a monolithic ”canonical” structure, this partial translation network can remain both flexible and modular, minimizing risky lock-step updates whenever new releases or extensions emerge.

6.1.2 Long-Term Goal: OpenPRA JSON Schema

The long-term initiative extends beyond partial connectivity to propose an openly licensed JSON schema under the OpenPRA umbrella. This plan intends to balance retained compatibility with existing standards against the specialized requirements of nuclear licensing. Two major factors underlie this choice:

- **Extensibility and Clarity:** JSON natively supports hierarchical key-value pairs that facilitate both human readability and efficient parsing. PRA models tend to store domain-specific knowledge, not merely numeric parameters, making readability a high priority. Unlike FlatBuffers or similar purely binary formats, JSON offers a middle ground by allowing large, descriptive models without sacrificing parser performance.

- **Nuclear-Specific Needs:** Although OpenPSA XML has served as a standard in probabilistic safety, it is no longer actively maintained. The OpenPRA JSON schema seeks to preserve core OpenPSA semantics while adding a namespace dedicated to nuclear regulation. This extension accommodates plant-specific elements and licensing-related fields that exceed the scope of general probabilistic models.

Thus, the near-term effort builds a manageable translation spanning tree through PRAcciolini, minimizing integration burdens across multiple closed-source tools. Simultaneously, OpenPRA JSON emerges as a robust, future-facing schema for those organizations requiring more comprehensive and domain-tailored data structures. This dual approach, maintaining workable interoperability while evolving a modern open standard, addresses both the immediate priorities of the PRA community and the long-term need for a sustainable, extensible foundation.

6.2 Generic Models

6.2.1 The Aralia Fault Tree Data Set

The Aralia dataset is a collection of 43 fault trees designed to exercise a wide array of logical features, problem sizes, and failure probability scales [35]. In Table 6.1, each entry denotes a distinct fault tree with a different configuration of basic events, gates, and minimal cut sets, ultimately yielding broad diversity in both computational complexity and system reliability.

A prominent feature of the Aralia dataset is its wide range of fault-tree sizes and structures. On the smaller side, some references list only a few dozen basic events (e.g., `chinese` with

25 BEs or `isp9605` with 32 BEs). Others, such as `nus9601`, contain over 1,500 basic events, reflecting the scale of complex engineered systems or composite subsystems. Moreover, each fault tree employs a varied mix of `AND`, `OR`, `K/N` (voting), and occasionally `XOR` or `NOT` gates. This diversity of logical constructs makes Aralia an effective testbed for evaluating algorithms that parse and solve fault trees beyond the typical AND/OR structure.

Another key aspect of the Aralia models is the large spread in minimal cut set counts and top-event probabilities. Some trees report only a few hundred or a few thousand minimal cut sets, while others claim tens of millions or more (reaching up to 8×10^{10} in the most expansive configurations). The top-event probabilities vary from very rare failures on the order of 10^{-13} to moderately likely failure rates above 0.7. This variance is crucial when assessing numerical stability and runtime performance: methods employing rare-event approximations can dramatically underestimate probabilities for the more frequent events, while computational overhead grows rapidly for cut set expansions in highly interconnected models. All of the Aralia fault trees are provided in OpenPSA Extensible Markup Language (XML) format.

Table 6.1: Summary statistics for the Aralia fault tree dataset [35].

#	**Fault**	**Basic**	**Logic Gates**					**Minimal**	**Top Event**
	Tree	**Events**	**Total**	AND	VOT	XOR	NOT	**Cut Sets**	**Probability**
1	baobab1	61	84	16	9	-	-	46,188	1.01708E-04
2	baobab2	32	40	5	6	-	-	4,805	7.13018E-04
3	baobab3	80	107	46	-	-	-	24,386	2.24117E-03
4	cea9601	186	201	69	8	-	30	130,281,976	1.48409E-03
5	chinese	25	36	13	-	-	-	392	1.17058E-03
6	das9201	122	82	19	-	-	-	14,217	1.34237E-02
7	das9202	49	36	10	-	-	-	27,778	1.01154E-02
8	das9203	51	30	1	-	-	-	16,200	1.34880E-03

Table 6.1: *Summary statistics for the Aralia fault tree dataset.*

#	**Fault**	**Basic**	**Logic Gates**					**Minimal**	**Top Event**
	Tree	**Events**	**Total**	AND	VOT	XOR	NOT	**Cut Sets**	**Probability**
9	das9204	53	30	12	-	-	-	16,704	6.07651E-08
10	das9205	51	20	2	-	-	-	17,280	1.38408E-08
11	das9206	121	112	21	-	-	-	19,518	2.29687E-01
12	das9207	276	324	59	-	-	-	25,988	3.46696E-01
13	das9208	103	145	33	-	-	-	8,060	1.30179E-02
14	das9209	109	73	18	-	-	-	8.20E+10	1.05800E-13
15	das9601	122	288	60	36	12	14	4,259	4.23440E-03
16	das9701	267	2,226	1,739	-	-	992	26,299,506	7.44694E-02
17	edf9201	183	132	12	-	-	-	579,720	3.24591E-01
18	edf9202	458	435	45	-	-	-	130,112	7.81302E-01
19	edf9203	362	475	117	-	-	-	20,807,446	5.99589E-01
20	edf9204	323	375	106	-	-	-	32,580,630	5.25374E-01
21	edf9205	165	142	30	-	-	-	21,308	2.09351E-01
22	edf9206	240	362	126	-	-	-	385,825,320	8.61500E-12
23	edfpa14b	311	290	70	-	-	-	105,955,422	2.95620E-01
24	edfpa14o	311	173	42	-	-	-	105,927,244	2.97057E-01
25	edfpa14p	124	101	42	-	-	-	415,500	8.07059E-02
26	edfpa14q	311	194	55	-	-	-	105,950,670	2.95905E-01
27	edfpa14r	106	132	55	-	-	-	380,412	2.09977E-02
28	edfpa15b	283	249	61	-	-	-	2,910,473	3.62737E-01
29	edfpa15o	283	138	33	-	-	-	2,906,753	3.62956E-01
30	edfpa15p	276	324	33	-	-	-	27,870	7.36302E-02
31	edfpa15q	283	158	45	-	-	-	2,910,473	3.62737E-01

Table 6.1: Summary statistics for the Aralia fault tree dataset.

#	**Fault**	**Basic**	**Logic Gates**					**Minimal**	**Top Event**
	Tree	**Events**	**Total**	AND	VOT	XOR	NOT	**Cut Sets**	**Probability**
32	edfpa15r	88	110	45	-	-	-	26,549	1.89750E-02
33	elf9601	145	242	97	-	-	-	151,348	9.66291E-02
34	ftr10	175	94	26	-	-	-	305	4.48677E-01
35	isp9601	143	104	25	1	-	-	276,785	5.71245E-02
36	isp9602	116	122	26	-	-	-	5,197,647	1.72447E-02
37	isp9603	91	95	37	-	-	-	3,434	3.23326E-03
38	isp9604	215	132	38	-	-	-	746,574	1.42751E-01
39	isp9605	32	40	8	6	-	-	5,630	1.37171E-05
40	isp9606	89	41	14	-	-	-	1,776	5.43174E-02
41	isp9607	74	65	23	-	-	-	150,436	9.49510E-07
42	jbd9601	533	315	71	-	-	-	150,436	7.55091E-01
43	nus9601	1,567	1,622	392	47	-	-	unknown	unknown

6.2.2 The Generic Pressurized Water Reactor Model

Since 1995, INL has developed Standardized Plant Analysis Risk (SPAR) models for the US NRC to provide critical risk-informed input to the regulatory process. The generic PWR model is available as a SAPHIRE model, accompanied by a report detailing the foundations of the modeling phenomena and referencing failure data. The current version of the SAPHIRE model is v1.2; however, the documented model information corresponds to v1.0 [5]. This model is continuously evolving, with researchers and PRA practitioners actively working to improve it.

At present, the model considers various initiating events, including seismic activity, internal flooding, internal fires, hurricanes, high winds, large break Loss of Cooling Accident

(LOCA), ISLOCA (Interfacing Systems LOCA), upset conditions leading to transients, loss of Component Cooling Water (CCW), loss of DC bus, loss of feedwater, loss of offsite power, large steam line breaks, medium break LOCA, steam generator tube ruptures, small LOCA, tornadoes, and excessive LOCA. Version 1.0 of the model includes 56 event trees linked with 140 fault trees. Summary statistics and SAPHIRE quantification results for the Generic PWR v1.0 model are provided in Table 6.2.

Table 6.2: Summary statistics for the Generic Pressurized Water Reactor model, quantified in SAPHIRE [5].

#	Input Model	Description	# MCS	Probability	Seq.
1	EQK-BIN1	Seismic event in Bin 1 (0.1–0.3g) occurs Bin PGA 0.17)	610,016	2.085×10^{-9}	85
2	EQK-BIN2	Seismic event in Bin 2 (0.3–0.5g) occurs (Bin PGA 0.39)	811,575	2.430×10^{-7}	85
3	EQK-BIN3	Seismic event in Bin 3 (0.5–0.75g) occurs (Bin PGA 0.61)	802,392	2.336×10^{-6}	85
4	EQK-BIN4	Seismic event in Bin 4 (0.75–1.0g) occurs (Bin PGA 0.87)	598,505	2.857×10^{-6}	85
5	EQK-BIN5	Seismic event in Bin 5 (1.0–1.5g) occurs (Bin PGA 1.22)	440,041	2.348×10^{-6}	85
6	EQK-BIN6	Seismic event in Bin 6 (1.5–3.0g) occurs (Bin PGA 2.12)	177,330	6.006×10^{-7}	85
7	EQK-BIN7	Seismic event Bin 7 (> 3.0g) occurs	2	9.623×10^{-9}	1
8	FLI-4160VACA	IF – 4160V AC Room A	363,408	2.551×10^{-9}	29
9	FLI-4160VACB	IF – 4160V AC room B	364,430	2.563×10^{-9}	29
10	FLI-AFW-ROOM	IF – AFW pump rooms	126,120	2.109×10^{-8}	29
11	FLI-CCW-ROOM	IF – CCW pump rooms	283,639	2.941×10^{-8}	29
12	FLI-CCW-ROOMA	IF – CCW pump room A	631,309	1.065×10^{-9}	29
13	FLI-CCW-ROOMB	IF – CCW pump room B	636,421	1.066×10^{-9}	29
14	FLI-CVC-ROOM	IF – CVC pump room	277,526	3.958×10^{-10}	29
15	FLI-RHR-ROOM	IF – RHR pump room	36,008	1.071×10^{-10}	29
16	FLI-SWS-ROOM	IF – SWS pump rooms	190,025	2.997×10^{-7}	29

Table 6.2: Summary statistics for the Generic Pressurized Water Reactor model, quantified in SAPHIRE.

#	Input Model	Description	# MCS	Probability	Seq.
17	FLI-SWS-ROOMA	IF – SWS pump room A	182,124	$1.402{\times}10^{-8}$	29
18	FLI-SWS-ROOMB	IF – SWS pump room B	188,786	$1.409{\times}10^{-8}$	29
19	FRI-AB-AFWAB	Fire in auxiliary building causing failure AFW MDP A&B	420,246	$1.243{\times}10^{-8}$	29
20	FRI-AB-CCWBC	Fire in auxiliary building causing failure CCW trains A&C	528,190	$7.673{\times}10^{-10}$	29
21	FRI-AB-LOOP	Fire in aux building causes LOOP	29,283	$1.098{\times}10^{-8}$	33
22	FRI-AB-LOOP-DIVA	Fire in aux building causes LOOP and loss of div A AC	45,325	$3.138{\times}10^{-6}$	33
23	FRI-AB-LOOP-DIVB	Fire in aux building causes LOOP and loss of div B AC	46,090	$3.159{\times}10^{-6}$	33
24	FRI-AB-RHRA	Fire in auxiliary building causing failure RHR train A	647,063	$2.114{\times}10^{-9}$	29
25	FRI-AB-SIS	Fire in auxiliary building causing failure SIS trains	659,428	$5.416{\times}10^{-8}$	29
26	FRI-AB-SLOCA	Fire in auxiliary building causes spurious PORV opening	14,257	$5.499{\times}10^{-7}$	8
27	FRI-MCR	Fire in main control room causes an evacuation	3,464	$9.081{\times}10^{-6}$	6
28	FRI-SWS-BLD	Fire in service water building failing SWS	71,923	$9.331{\times}10^{-8}$	29
29	HCN-BIN1	Hurricane wind event Bin 1 (111 mph–135 mph)	1,703,431	$4.868{\times}10^{-9}$	63
30	HCN-BIN2	Hurricane wind event Bin 2 (136 mph–165 mph)	1,064,448	$2.181{\times}10^{-9}$	63
31	HCN-BIN3	Hurricane wind event Bin 3 (166 mph–200 mph)	475,235	$9.346{\times}10^{-10}$	63
32	HCN-BIN4	Hurricane wind event Bin 4 (> 200 mph)	230,212	$4.737{\times}10^{-10}$	63
33	HWD-96MPH	High wind (<110 mph) event occurs	2,558,791	$2.264{\times}10^{-8}$	63

Table 6.2: Summary statistics for the Generic Pressurized Water Reactor model, quantified in SAPHIRE.

#	Input Model	Description	# MCS	Probability	Seq.
34	ISL-RHR-CL	ISLOCA RHR cold leg	217	3.063×10^{-11}	2
35	ISL-RHR-HL	ISLOCA RHR hot leg	3	2.063×10^{-8}	2
36	L4160ACA	Loss of 4160 VAC Bus A	160,390	2.207×10^{-7}	29
37	L4160ACB	Loss of 4160 VAC Bus B	162,204	2.220×10^{-7}	29
38	LLOCA	Large break LOCA	20,119	3.617×10^{-7}	3
39	LOCCW	Total loss of CCW	778,360	8.245×10^{-8}	29
40	LODCA	Loss of 125 VDC Bus A	156,714	4.249×10^{-8}	29
41	LODCB	Loss of 125 VDC Bus B	160,549	4.271×10^{-8}	29
42	LOMFW	Loss of main feedwater	1,550,153	7.502×10^{-8}	18
43	LOOPGR	LOOP (grid-related)	1,111,725	7.913×10^{-7}	33
44	LOOPPC	LOOP (plant-centered)	851,575	7.619×10^{-8}	33
45	LOOPSC	LOOP (switchyard-centered)	1,253,736	7.286×10^{-7}	33
46	LOOPWR	LOOP (weather-related)	1,014,915	5.660×10^{-7}	33
47	LSSB	Large steam line break (unisolable inside containment)	216,762	1.872×10^{-7}	16
48	MLOCA	Medium break LOCA	67,620	9.332×10^{-6}	5
49	SGTR	Steam generator tube rupture	1,039,329	2.668×10^{-6}	12
50	SLOCA	Small break LOCA	309,917	2.453×10^{-5}	8

Table 6.2: Summary statistics for the Generic Pressurized Water Reactor model, quantified in SAPHIRE.

#	**Input Model**	**Description**	**# MCS**	**Probability**	**Seq.**
51	TOR-BIN1	Tornado event Bin-1 (136–165 mph)	128,067	$5.355{\times}10^{-12}$	63
52	TOR-BIN2	Tornado event Bin-2 (166–200 mph)	126,084	$2.614{\times}10^{-11}$	63
53	TOR-BIN3	Tornado event Bin-3 (> 200 mph)	95,064	$1.296{\times}10^{-10}$	63
54	TRANS	Transient	4,178,502	$1.525{\times}10^{-7}$	29
55	XLOCA	Excessive LOCA	1	$1.000{\times}10^{-7}$	1
	TOTAL		**28,559,059**		**1965**

6.3 Synthetic Models

6.3.1 Automated Synthetic Model Generation

Synthetic PRA models were generated using an automated model generation utility that supports both fault tree (FT) and event tree (ET) structures [38]. The generation process is configurable, with users specifying probabilistic parameters either through a command-line interface (CLI) for fault trees or via a comma-separated values (CSV) configuration file for event trees. The generator outputs models in multiple formats, including OpenPSA XML and SAPHSOLVE JSInp and XFTA.

For fault tree generation, the CLI interface accepts arguments that define the number of basic events, the types and weights of logic gates, the probabilities assigned to basic events, and the structure of the tree, including the presence of common basic events and gates as well as parent-child relationships. The event tree generator, on the other hand, is configured through a CSV file that specifies the event tree structure and the associated fault trees for each functional event. This file includes arguments for naming conventions, the number of functional events, and the logic and data for each linked fault tree. For both FTs and ETs, the generator allows for the specification of random seeds to ensure reproducibility.

Model validation is performed for OpenPSA models using a RELAX NG schema, which ensures that the generated XML files conform to the required standard. Validation results, including any errors, are logged for further review and correction.

Table 6.3 summarizes the key input parameters required for both FT and ET generation [44].

Table 6.3: Configuration options for automatically generating synthetic event & fault trees.

Argument	**Option**	**Type**	**Default**	**Applied Option**
Name for event tree	–et-name	RegEx/string	autogen	n_min_max
Number of Functional Event (FE)s	–num-func	range, int	1	1:10:100
Name for fault tree	–ft-name	RegEx/string	autogen	n_min_max
Name for the top gate	–root	string	root	
Seed for PRNG	–seed	int	123	372
Number of BEs	–num-basic	range, int	100	100:50:5000
Avg. number of gate arguments	–num-args	+ve float	3.0	
Weights [AND, OR, VOT, NOT, XOR]	–weights-g	+ve float[]	[1,1,1,1,1]	[1,1,1,0,0]
Avg. % of common BEs per gate	–common-b	+ve float	0.3	
Avg. % of common gates per gate	–common-g	+ve float	0.1	
Avg. number parents for common BEs	–parents-b	+ve float	2	
Avg. number of parents for common gates	–parents-g	+ve float	2	
Number of gates (<= 0 means auto)	–num-gate	int	0	
Maximum probability of BEs	–max-prob	float [0,1]	0.5	0.05
Minimum probability of BEs	–min-prob	float [0,1]	0.5	0.01
Number of Flag/House Event (HE)s	–num-house	int	0	
Number of CCF groups	–num-ccf	int	0	
Output format [XML, JSON, JSInp]	–out	{xml, jsinp, json}	xml	xml, jsinp

6.3.2 Summary of Generated Synthetic Datasets

A total of thirteen distinct synthetic model datasets [12] were generated using the methodology described above. Each dataset is characterized by its structural configuration, parameter ranges, and output formats. The datasets are designed to cover a broad spectrum of model sizes and complexities. All models are deep fault trees, with variations in the number of gates, basic events, and gate type distributions.

The `100_to_100k_voter` dataset consists of models with 100, 1,000, 10,000, and 100,000 basic events, with gate weights configured either for all gate types or for K/N gates only, and a fixed percentage of common basic events. The `1_to_4k` dataset contains models with the number of gates ranging from 1 to 4,000, alternating between OR and AND gates. The `1_to_50k` and `1_to_5k` datasets similarly vary the number of gates from 1 up to 50,000 and 5,000, respectively, with alternating gate types and are provided in multiple formats. The `2_to_100k` dataset includes models with basic events ranging from 2 to 100,000, with AND and OR gate types and varying percentages of common basic events. The `500_to_750` dataset features models with 500, 550, 600, and 750 basic events, with specific gate weights, common event percentages, and, in some cases, common-cause failure groups.

The remaining datasets, labeled `c1-P_0.01-0.05` through `c7-P_0.35-0.9`, are parameter sweeps where the number of basic events ranges from 100 to 5,000, and the maximum and minimum probabilities for basic events are systematically varied. These datasets are provided in OpenPSA and, where applicable, SAPHSOLVE JSInp formats.

Table 6.4 provides an overview of the generated datasets, including their naming conventions, key parameter ranges, and supported formats. All datasets are publicly available and are structured to facilitate reproducibility and direct comparison across PRA quantification engines.

Table 6.4: Summary of generated synthetic model datasets and their parameterizations [12].

Dataset Name	**Parameter**	**Value / Range**	**Formats**
100_to_100k_voter	Number of basic events	100, 1,000, 10,000, 100,000	OpenPSA
	Gate weights	all gates; K/N only	
	Common basic events (%)	0.01	

Table 6.4: Summary of generated synthetic model datasets and their parameterizations.

Dataset Name	**Parameter**	**Value / Range**	**Formats**
1_to_4k	Number of gates	1–4,000	JSInp
	Gate type sequence	Alternating OR/AND	
1_to_50k	Number of gates	1–50,000	OpenPSA,
	Gate type sequence	Alternating OR/AND	JSInp
1_to_5k	Number of gates	1–5,000	OpenPSA,
	Gate type sequence	Alternating OR/AND	JSInp
2_to_100k	Number of basic events	2, 10, 100, 1,000, 10,000, 100,000	OpenPSA
	Gate types	AND, OR	
	Common basic events (%)	0.01–0.09	
500_to_750	Number of basic events	500, 550, 600, 750	OpenPSA
	Gate weights	[AND, OR, K/N]	
	Common basic events (%)	0.5–0.7	
	Common gates (%)	0.3–0.5	
	CCF groups	0 or 2	
c1-P_0.01-0.05	Number of basic events	100:50:5000	OpenPSA,
	Gate weights	[1,1,1,0,0]	JSInp
	Max/Min probability	max 0.05, min 0.01	
c2-P_0.5-0.9	Number of basic events	100:50:5000	OpenPSA
	Gate weights	[1,1,1,0,0]	
	Max/Min probability	max 0.9, min 0.5	
c3-P_0.01-0.9	Number of basic events	100:50:5000	OpenPSA
	Gate weights	[1,1,1,0,0]	

Table 6.4: Summary of generated synthetic model datasets and their parameterizations.

Dataset Name	**Parameter**	**Value / Range**	**Formats**
	Max/Min probability	max 0.9, min 0.01	
c4-P_0.05-0.9	Number of basic events	100:50:5000	OpenPSA
	Gate weights	[1,1,1,0,0]	
	Max/Min probability	max 0.9, min 0.05	
c5-P_0.1-0.9	Number of basic events	100:50:5000	OpenPSA
	Gate weights	[1,1,1,0,0]	
	Max/Min probability	max 0.9, min 0.1	
c6-P_0.25-0.9	Number of basic events	100:50:5000	OpenPSA
	Gate weights	[1,1,1,0,0]	
	Max/Min probability	max 0.9, min 0.25	
c7-P_0.35-0.9	Number of basic events	100:50:5000	OpenPSA
	Gate weights	[1,1,1,0,0]	
	Max/Min probability	max 0.9, min 0.35	

Part III

A Brute Force Approach

Chapter 7

Building a Data-Parallel Monte Carlo Probability Estimator

To handle massively parallel Monte Carlo evaluations of large-scale Boolean functions, we have developed a feedforward-layer architecture that organizes computation in a topological graph. At the lowest level, each Boolean variable/basic event (e.g., a component failure) is associated with a random number generator to sample its truth assignment. We bit-pack these outcomes, storing multiple Monte Carlo samples in each machine word to maximize computational throughput and reduce memory footprint. Subsequent layers consist of logically higher gates or composite structures that receive the bit-packed results from previous layers and combine them in parallel using coalesced kernels. By traversing the computation graph topologically, dependencies between gates and events are naturally enforced, so kernels for each layer can run concurrently once all prerequisite layers finish, resulting in high kernel occupancy and predictable throughput.

Section 7.4 formalizes how these kernels map the logical sampling workload onto a three–dimensional ND–range, introducing a consistent coordinate system (i_x, i_y, i_z) and a rounding scheme that guarantees complete coverage without redundancy. The remainder of this chapter adopts that notation.

In practice, each layer is dispatched to an accelerator node using a data-parallel model ed using SYCL. The random number generation pipelines are counter-based, ensuring reproducibility and thread-safety even across millions or billions of samples. Gates that go beyond simple AND/OR logic–such as VOT operators–are handled by specialized routines that can exploit native popcount instructions for efficient threshold evaluations. As we progress upwards through the layered topology, each gate or sub-function writes out its bit-packed output, effectively acting as an input stream to the next layer. Throughout the simulation, online tallying kernels aggregate how often each node or gate evaluates to True. These tallies can then be turned into estimates of probabilities and sensitivity metrics on the fly. This approach also makes adaptive sampling feasible: if specific gates appear to dominate variance or are tied to particularly rare events, additional sampling can be allocated to their layer to refine estimates.

7.1 Minimal Knowledge–Compilation Preprocessing for Monte-Carlo Sampling

Before any kernels are built, the solver applies a *single, very light* compile pass whose only purpose is to shave off two gate types that would otherwise require special kernels:

- **NULL gates** – a gate whose output is logically a no-op is deleted and any fan-out rewired directly to its input buffer.

- **NOT gates** – instead of scheduling a one-input gate kernel, we tag the affected buffer with an *inversion flag.* Every subsequent kernel simply toggles the word with a bitwise ~ on read.

7.1.1 Why such little knowledge–compilation?

Classical KC pipelines (Section 4.1) strive for determinism or decomposability to enable *exact* inference. Monte-Carlo evaluation requires no such restrictions. No other syntactic normalization is attempted: negations may appear at arbitrary depth, XOR or k/n gates remain untouched, and literals are free to occur with both polarities in different contexts. This choice maximizes semantic expressivity and succinctness – flattening or pushing down is possible by using a higher compilation flag, but not strictly necessary.

7.2 Layered Topological Organization

Recall that a PDAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains no cycles, so there is at least one valid *topological ordering* of its nodes. A topological ordering assigns each node a numerical *layer index* such that all edges point from a lower-numbered layer to a higher-numbered layer. If a node v consumes the outputs of nodes $\{u_1, \ldots, u_k\}$, then we require

$$\text{layer}(u_i) \;<\; \text{layer}(v) \quad \text{for each } i \in \{1, \ldots, k\}.$$

In other words, node v can appear only after all of its inputs in a linear or layered listing.

The essential steps to build and traverse these layers are:

1. *Compute Depths via Recursive Analysis:* Each node's depth is found by inspecting its children (or inputs). If a node is a leaf (e.g., a `Variable` or `Constant` that does not depend on any other node), its depth is 0. Otherwise, its depth is one larger than the maximum depth among its children.

2. *Group Nodes by Layer:* Once each node's depth is computed, nodes of equal depth form a single *layer*. Thus, all nodes with depth 0 are in the first layer, those with depth 1 in the second layer, and so on.

3. *Sort Nodes within Each Layer:* Within each layer, enforce an additional consistent ordering: (i) variables appear before gates, (ii) gates of different types can be grouped to facilitate specialized processing. This step is not strictly required for correctness, but it can streamline subsequent stages such as kernel generation or partial evaluations.

4. *Traverse Layer by Layer:* A final pass iterates over each layer in ascending order. Because all inputs of any node in layer d lie in layers $< d$, the evaluation (or ”kernel build”) for layer d can proceed after the entire set of layers $0, \ldots, d-1$ is processed.

This structure ensures a sound evaluation of the PDAG: no gate or variable is computed until after all of its inputs are finalized.

7.2.1 Depth Computation and Node Collection

1. **Clear Previous State.** Any existing ”visit” markers or stored depths in the PDAG-based data structures are reset to default values (e.g., zero or -1).

2. **Depth Assignment by Recursion.** A `compute_depth` subroutine inspects each node:

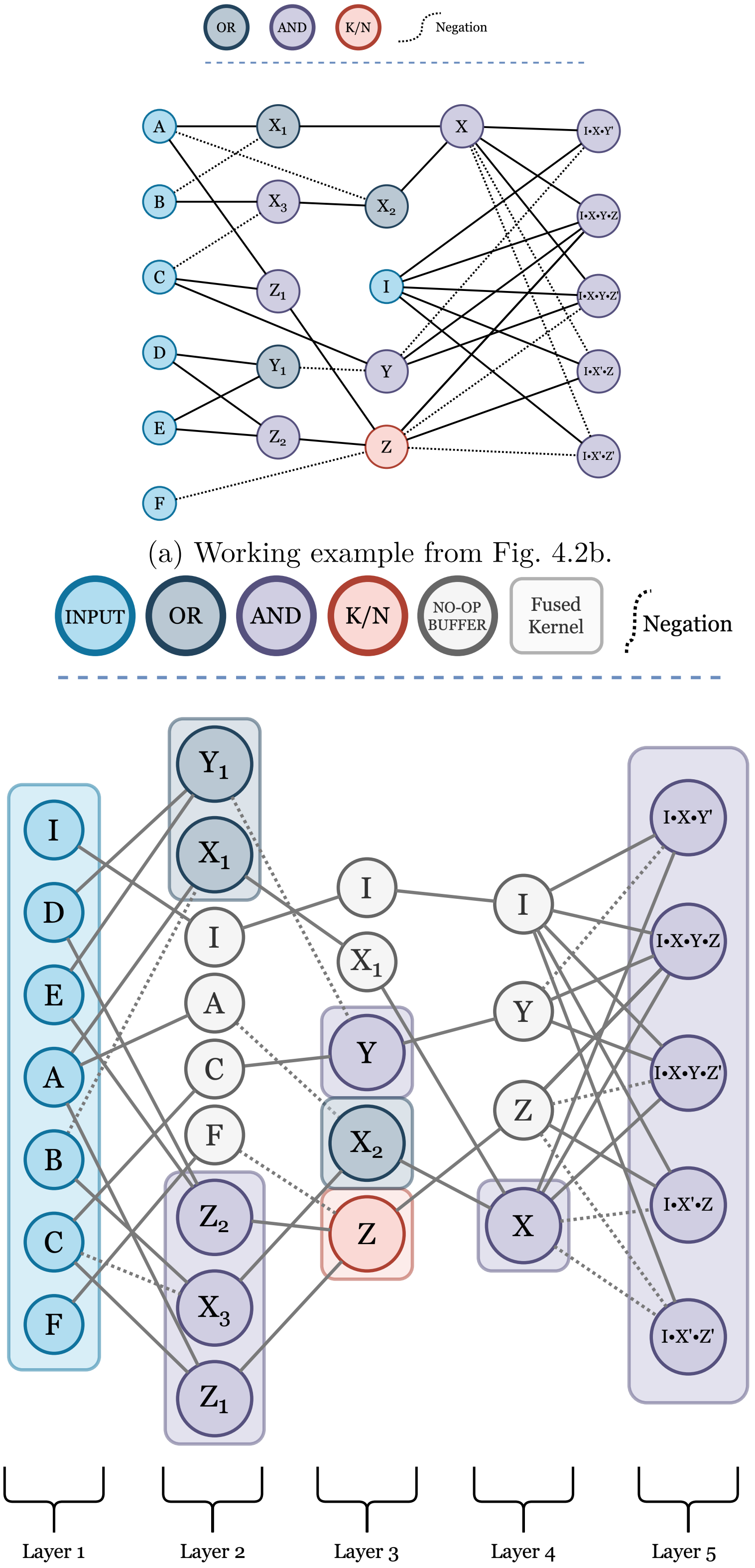
   (a) If the node is a `Variable` or `Constant`, it is a leaf in the PDAG, so depth $= 0$.

   (b) If the node is a `Gate` with multiple inputs, the procedure first recursively computes the depths of its inputs. It then sets its own depth as

   $$\text{depth}(\texttt{gate}) \;=\; 1 \;+\; \max_{\ell \in \text{inputs of gate}} \Big[\text{depth}(\ell)\Big].$$

3. **Order Assignment.** Each node stores the newly computed depth in an internal field. This numeric value anchors the node to a layer. A consistent pass over the entire graph ensures correctness for all nodes.

After depths are assigned, gather all nodes, walking the PDAG from its root, recording each discovered node and adding it to a global list.



(a) Working example from Fig. 4.2b.

Figure 7.1: Layered topological ordering on the Propositional Directed Acyclic Graph (DAG), with coalesced/fused kernels, partitioned by operation type.

7.2.2 Layer Grouping and Local Sorting

Begin by creating:

- A global list of all nodes, each with a valid depth,
- A mapping from node indices to node pointers,

Then, sort the global list by ascending depth. Let $\mathrm{order}(n)$ be the depth of node n. Then

$$\mathrm{order}(n_1) \ \leq \ \mathrm{order}(n_2) \ \leq \ \ldots \ \leq \ \mathrm{order}(n_{|\mathcal{V}|}).$$

Finally, partition this list into contiguous *layers*: if the deepest node has a depth $\delta_{\max}$, then create sub-lists:

$$\{\text{nodes s.t. depth} = 0\}, \quad \{\text{nodes s.t. depth} = 1\}, \quad \ldots, \quad \{\text{nodes s.t. depth} = \delta_{\max}\}.$$

Within each layer, sort nodes to ensure that `Variable` nodes precede `Gate` nodes, and `Gate` nodes may be further sorted by `Connective` type (e.g., `AND`, `OR`, `VOT`, etc.).

7.2.3 Layer-by-Layer Kernel Construction

Apply the layer decomposition to drive *kernel building* and *evaluation*:

1. **Iterate over each layer in ascending depth**. Because every node's dependencies lie in a strictly lower layer, one is guaranteed that those dependencies have already been assigned memory buffers, partial results, or other necessary resources.

2. **Partition the layer nodes into subsets by node type**. Concretely, `Variable` nodes are batched together for *basic-event sampling* kernels, while `Gate` nodes are transferred into *gate-evaluation* kernels.

3. **Generate device kernels**. For `Variable` nodes, create Monte Carlo sampling kernels. For `Gate` nodes, it constructs logical or bitwise operations that merge or transform the sampled states of the inputs.

Once kernels for a given layer finish, move on to the next layer. Because of the topological guarantee, no node in layer d references memory or intermediate states from layer $d+1$ or later, preventing cyclical references and ensuring correctness.

7.3 Adopting the SYCL Execution Model

Before getting into our opinionated launch parameters, it is instructive to recall the abstract execution hierarchy defined by the SYCL standard. This detour provides the conceptual foundation upon which the remainder of this dissertation builds.

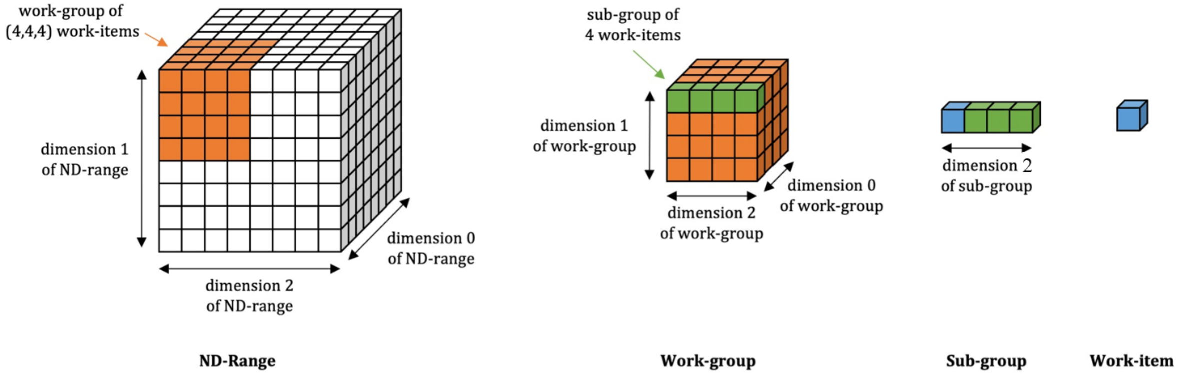


Figure 7.2: At a glance: The SYCL execution model describes relationships between ND-Ranges, work-groups, sub-groups, and work-items.

7.3.1 Conceptual Overview

Modern accelerator programming models must present the developer with a *logical* view of parallel work that is independent of any specific piece of hardware. SYCL achieves this by defining a small hierarchy of index spaces. Each level in the hierarchy provides progressively stronger coherence and synchronization guarantees, yet none of them prescribes *where* that work will eventually run. The mapping from logical indices to physical execution resources is entirely deferred to the run–time or device compiler and is therefore opaque to the application. In what follows we formalize the four key abstractions exposed by the SYCL execution model and derive a set of identities that will be reused throughout the remainder of this dissertation.

7.3.2 Hierarchical Index Spaces

Let

$$\mathbf{G} = (G_x, G_y, G_z) \in \mathbb{N}^3, \qquad G_d > 0 \quad (d \in \{x, y, z\}),$$

be the *global range*. It enumerates the total number of logical tasks, or *work–items*, that shall be executed in one kernel invocation. The associated set of global indices is

$$\mathcal{I} \;=\; \{0, \ldots, G_x - 1\} \times \{0, \ldots, G_y - 1\} \times \{0, \ldots, G_z - 1\},$$

with $|\mathcal{I}| = G_x G_y G_z$.

A second triple

$$\mathbf{L} = (L_x, L_y, L_z), \qquad 0 < L_d \leq G_d,$$

referred to as the *local range*, partitions the ND–Range into disjoint *work–groups*. Defining

$$W_d \;=\; \frac{G_d}{L_d}, \qquad d \in \{x, y, z\},$$

(which implies $G_d \equiv 0 \pmod{L_d}$), the set of group indices reads

$$\mathcal{W} \;=\; \{0, \ldots, W_x - 1\} \times \{0, \ldots, W_y - 1\} \times \{0, \ldots, W_z - 1\},$$

with cardinality $|\mathcal{W}| = W_x W_y W_z$. Each group $w \in \mathcal{W}$ owns exactly

$$|\Gamma| \;=\; L_x L_y L_z$$

work–items that share fast local memory and barrier synchronization.

Within a work–group the implementation may further partition the local index space into *sub–groups* of size S:

$$S \;=\; |\Sigma|, \quad \Sigma \subseteq \Gamma, \quad 1 \leq S \leq |\Gamma|.$$

Sub–groups execute in (near) lockstep and admit specialized collective operations, yet their existence and size remain device–specific. Finally, the singleton element of the hierarchy is the **work–item**, uniquely addressed by its global id $i = (i_x, i_y, i_z) \in \mathcal{I}$.

The containment relations

$$\text{work–item} \;\in\; \text{sub–group} \;\subseteq\; \text{work–group} \;\subseteq\; \text{ND–Range}$$

hold for every index triple.

7.3.3 Abstractness of the Model

None of the above definitions mention vector widths, cores, or memory banks. The SYCL execution model is strictly an *index algebra*; it provides (i) a naming scheme for independent pieces of work, and (ii) a lattice of synchronization points that the run–time must respect. Once the tuples **G** and **L** have been fixed, every additional property of the physical execution, including occupancy, scheduling order, and even whether groups are run concurrently or serially, is an implementation detail.

Notation used from here onward.

We will exploit the following shorthand throughout the subsequent analysis:

1. $|\mathcal{I}| = G_x G_y G_z \quad$ (total work–items),

2. $|\mathcal{W}| = W_x W_y W_z \quad$ (total work–groups),

3. $|\Gamma| = L_x L_y L_z \quad$ (work–items per group),

4. $\langle i_g, i_l \rangle \quad$ embodiment of a work–item by its enclosing group index $i_g \in \mathcal{W}$ and its local index $i_l \in \Gamma$.

These identities are purely algebraic and therefore remain valid for *any* SYCL–conformant device.

From Abstraction to Implementation Strategy. Next, we translate these concepts into the concrete launch geometries and dependency patterns required by our Monte–Carlo solver. The forthcoming sections build progressively from global range rounding rules to kernel–specific mappings.

7.3.4 Basic–Event Sampling Kernels

Each `Variable` node in layer d represents an i.i.d. Bernoulli trial with success probability $p\_v \in [0, 1]$. The evaluation of all variables in a layer is consolidated into *one* data–parallel kernel that generates a contiguous block of bit–packed outcomes:

1. **Parameter staging.** For every variable v the solver stores the pair $(\mathrm{idx}(v),\ p\_v)$ in host memory, where $\mathrm{idx}(v)$ is the global node index. The list is stable across Monte–Carlo iterations and is therefore transferred to device memory only once.

2. **Contiguous layout.** A device–side array of N_v records, N_v being the number of variables in the layer, is allocated so that the probability field, the bit–packed result buffer pointer, and any auxiliary counters are stored in *structure–of–arrays* (SoA) form. The resulting stride–free access pattern maximizes global–memory throughput.

3. **Kernel configuration.** Let T denote the total number of Bernoulli draws requested by the host run–time (cf. Sec. 8.2). The global ND–range is chosen as $\big(\lceil N_v \rceil,\, B,\, P\big)$, mapping each work–item to a unique triple $(v,\, b,\, p)$ of variable v, batch index b, and bit–pack index p. The local work–group shape is computed adaptively to saturate the target device while respecting hardware limits on registers and shared memory.

4. **Execution.** Every work–item initializes a counter–based generator (see the Philox discussion in Sec. 8.2), converts the pseudo–random words into ω Bernoulli outcomes via the integer–threshold technique, and writes the resulting w–bit word to the pre–allocated buffer. No inter–item synchronization is required beyond the implicit barrier at kernel completion.

The overall cost is $\mathcal{O}(T\, N_v/\omega)$ arithmetic operations and $\Theta(T\, N_v/\omega)$ global writes, making the routine memory–bandwidth bound only for extremely small P.

7.3.5 Gate–Evaluation Kernels

Gate nodes are logically heterogeneous: AND, OR, XOR, NOT, NAND, NOR, XNOR, and at–least–k (`VOT`) gates all feature distinct Boolean semantics yet share the same interface of reading one or more bit–packed input buffers and writing a bit–packed output. To avoid divergent control flow, the solver instantiates *one specialized kernel per connective type* present in the current layer.

Consider a set $\mathcal{G}_{\text{type}}$ containing all gates of a single connective. Their evaluation proceeds as follows:

1. **Input resolution.** For every gate $g \in \mathcal{G}$ the lists of positive inputs $\mathcal{I}^{+}(g)$ and negated inputs $\mathcal{I}^{-}(g)$ are resolved to concrete device pointers. Positive and negative buffers are concatenated so that a simple offset marks the first negated operand. The construction is embarrassingly parallel on the host and involves no device work.

2. **Contiguous block construction.** Buffers and gate metadata are packed into an SoA structure that is tile–aligned for coalesced reads. For at–least–k gates the threshold k is stored alongside the pointer list.

3. **Kernel launch geometry.** Let N_g be the number of gates of the selected type. An ND–range of $\big(\lceil N_g \rceil,\ B,\ P\big)$ is created, identical in shape to the basic–event kernel so that subsequent layers can reuse the same scheduling heuristics. Within each work–item, Boolean logic is applied on a per–bitpack basis without branching:

   - And, Nand: multiple `&` reductions plus an optional complement.
   - Or, Nor: multiple `|` reductions plus an optional complement.
   - Xor, Xnor: accumulated parity via `^` operations.
   - Null, Not: trivial one–input, output, with complement.
   - At–least–k: population counting of the aggregated bit–wise sum followed by a threshold comparison implemented through native `popcount` instructions.

4. **Dependency guarantees.** Because all input buffers originate in earlier layers, the run–time enforces an event dependency on every producing kernel, ensuring visibility of the complete inputs before gate evaluation begins.

The bit–parallel operations ensure that the arithmetic intensity is high; the critical path is dominated by a handful of integer masks and, for at–least–k gates, one integer addition plus a comparison per input.

7.3.6 Dependency–Aware Kernel Scheduling

Kernels are submitted to the device queue in strict layer order, yet the scheduler exploits two orthogonal forms of parallelism:

1. *Intra–layer concurrency* — basic–event sampling and the multiple gate kernels of the same layer depend exclusively on the previous layer, *not* on one another. They are therefore eligible for concurrent execution subject to device resources.

2. *Iterative sampling* — the bit–packed sample space is sliced into T_{iter} iterations decided by the sample shaper. Kernels capturing the same node repeat across iterations and are expressed with an explicit iteration counter, enabling the run–time to re–use the same compiled binary while varying the random counter seed and output offsets.

Dependencies are represented as light–weight events; the host never performs explicit synchronization inside a layer but relies on the queue to enforce the partial order.

7.3.7 Work–Group Optimization Heuristics

Let G denote the global item count of the kernel at hand and L_{max} the maximum local size supported by the device along each axis. The solver selects a local range (l_x, l_y, l_z) according to

$$\begin{aligned}
l_x &= min\Big(\text{pow2ceil}(G),\ L_{max}\Big),\\
l_y &= min\left(B,\ \frac{L_{max}}{l_x}\right),\\
l_z &= min\left(P,\ \frac{L_{max}}{l_x l_y}\right),
\end{aligned}$$

which heuristically balances occupancy with register pressure while retaining a uniform work–item distribution. The shape is re–evaluated independently for basic events and gate kernels because G differs across those two categories.

7.3.8 Complexity and Scalability

Assume $|\mathcal{V}|$ variables and $|\mathcal{G}|$ gates in the graph, with layer depths bounded by D. Let $S = T\,B\,P\,\omega$ be the total number of Bernoulli trials.

- **Kernel build time.** All host–side preprocessing runs in $\mathcal{O}(|\mathcal{V}|+|\mathcal{G}|)$ memory operations; no search structure deeper than a hash map is required.

- **Device execution time.** Each basic–event kernel performs S integer comparisons. Each gate kernel evaluates S Boolean operations whose count is proportional to the fan–in of the gate. Hence the total arithmetic complexity is $\mathcal{O}\big(S\,(1+\overline{\text{deg}})\big)$, where $\overline{\text{deg}}$ is the average gate fan–in.

- **Parallel scalability.** Both kernel categories exhibit linear speed–up with the number of compute units until either (i) the global launch size no longer saturates the device or (ii) memory bandwidth limits are reached. Because all kernels are fully independent across the B and P dimensions, they scale particularly well on multi–tile accelerators.

The design therefore provides a clear separation of concerns: depth–first analysis establishes the dependency structure; kernel generation translates that structure into homogeneous, vectorizable work; and a light–weight event system schedules the resulting kernels with minimal host intervention.

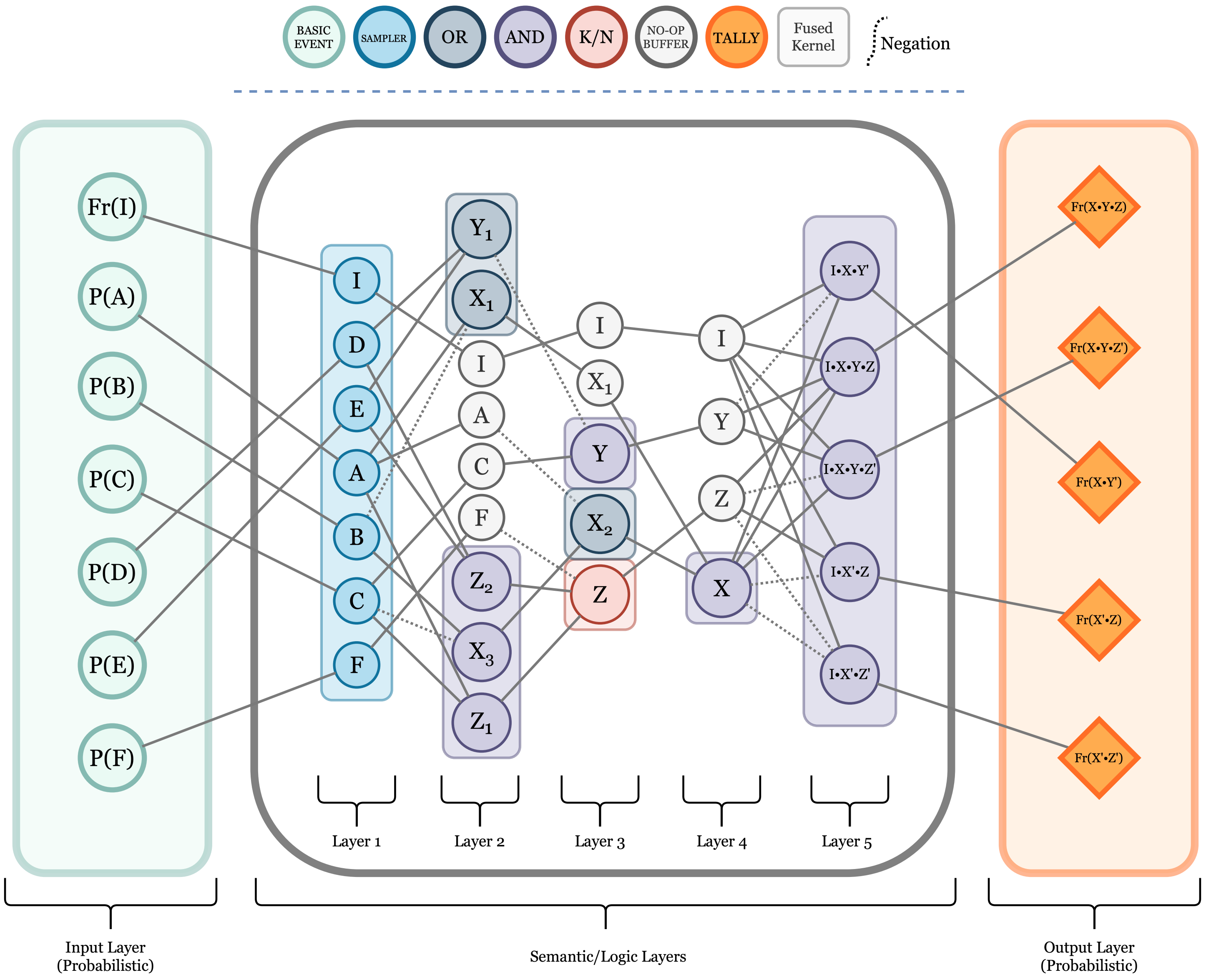


Figure 7.3: A fully connected Probabilistic-Propositional Directed Acyclic Graph.

7.4 Kernel-Level Execution Model

7.4.1 Coordinate System and Notation

Let $(G_x, G_y, G_z) \in \mathbb{N}^3$ denote the *global* range supplied to the device and (L_x, L_y, L_z) the *local* (work−group) range. We further define

$$W_d \; = \; \frac{G_d}{L_d}, \qquad d \in \{x, y, z\}, \quad \text{and} \quad W \; = \; W_x W_y W_z,$$

where W_d counts work-groups along axis d and W is the total number of work-groups. Every work-item within a group is identified by its local id $\ell = (\ell_x, \ell_y, \ell_z)$ with $0 \leq \ell_d < L_d$.

Unless stated otherwise the following global symbols are used throughout the chapter

V	# basic events (variables)
G	# standard logic gates
A	# at-least-k gates
T	Monte-Carlo iterations
B	batches per iteration
P	bit-packs per batch
ω	bits per pack $= 8 \cdot \text{sizeof}(\texttt{bitpack\_t})$
N	trials per iteration $= B\,P\,\omega$

7.4.2 Generic Rounding Scheme

All kernels adopt the *nearest-multiple* rule

$$G_d \; = \; \left\lceil \frac{Q_d}{L_d} \right\rceil L_d, \qquad Q_d \in \{V, G + A, 1\} \times \{B\} \times \{P\},$$

where Q_d is the problem-specific lower bound listed in Table 7.1. This rule guarantees that every logical task is scheduled while respecting the SYCL constraint $G_d \equiv 0 \pmod{L_d}$.

7.4.3 Kernel-Specific Mappings

Each kernel instantiates a surjective mapping

$$\Phi : \{0, \ldots, G_x - 1\} \times \{0, \ldots, G_y - 1\} \times \{0, \ldots, G_z - 1\} \ \twoheadrightarrow \ \mathcal{S},$$

where $\mathcal{S}$ is the set of logical sub-tasks it must solve. We list the mappings succinctly:

- **Basic-event sampling** ($\#\mathcal{S} = VBP$): $\Phi_{\mathrm{BE}}(i_x, i_y, i_z) = (v = i_x,\ b = i_y,\ p = i_z)$.
- **Standard gate evaluation** ($\#\mathcal{S} = GBP$): $\Phi_{\mathrm{G}}(i_x, i_y, i_z) = (g = i_x,\ b = i_y,\ p = i_z)$.
- **At-least-k gate evaluation** ($\#\mathcal{S} = ABP\omega$): $i_z = p\,\omega + \lambda$ with $\lambda \in \{0, \ldots, \omega - 1\}$. The pair (b, p) indexes the bit-pack, while λ singles out a *bit lane*. One work-group therefore owns a unique triplet (a, b, p) and folds the ω lanes with a group reduction.
- **Tally accumulation** ($\#\mathcal{S} = VBP$): identical to Φ_{BE} but with $L_x = 1$ such that each group covers exactly one tally node.

7.4.4 Trial Coverage Guarantee

Let Ξ be the set of Bernoulli trials processed by a kernel in one iteration. By construction

$$|\Xi| \ = \ \underbrace{BP\omega}_{\text{trials/ node}} \ \times \begin{cases} V, & \text{basic-event,} \\ G, & \text{standard gate,} \\ A, & \text{at-least gate,} \\ V, & \text{tally.} \end{cases}$$

Because ω divides G_z in every case, each trial is owned by exactly one work-item and is executed precisely once.

Table 7.1: Minimum global dimensions Q_d before round-up.

Kernel	Q_x	Q_y	Q_z
Basic-event	V	B	P
Standard gate	G	B	P
At-least-k gate	A	B	$P\omega$
Tally	V	B	P

7.4.5 Work-Group Invariants

Let Γ be a work-group. For every kernel the following invariant holds inside Γ:

$$\big[(\ell_x, \ell_y, \ell_z) \in \Gamma\big] \implies \text{all work-items share the complete set of inputs required to produce one output liter}$$

Consequently intra-group communication (reductions, barriers) never crosses logical boundaries, enabling lock-free execution except for the single atomic update in the tally kernel.

7.4.6 Complexity per Work-Group

With $L = L_x L_y L_z$ the number of instructions executed by a group is

$$C_\Gamma = \begin{cases} \Theta(\frac{\omega}{4}), & \text{basic-event (bit-packing)}, \\ \Theta(\deg g), & \text{gate of fan-in } \deg g, \\ \Theta(\deg a + \log \omega), & \text{at-least-}k, \\ \Theta(L + \log L), & \text{tally popcount + reduction}, \end{cases}$$

all independent of T owing to the strict buffering between iterations.

Chapter 8

Bitpacked Basic–Event Sampling Kernels

Monte Carlo simulations, probability evaluations, and other sampling-based procedures benefit greatly from efficient, high-quality Random Number Generator (RNG)s. A large class of modern RNGs are known as *counter-based Pseudo Random Number Generator (PRNG)s*, because they use integer counters (e.g., 32-bit or 64-bit) along with a stateless transformation to produce random outputs. The *Philox* family of counter-based PRNGs is a well-known example, featuring fast generation, high period, and good statistical properties. In this section, we discuss the general principles of counter-based PRNGs, explain how Philox fits into this paradigm, analyze its complexity, and present a concise pseudocode version of the Philox 4 × 32-10 variant. Subsequently, we detail the bitpacking scheme used to reduce memory consumption when storing large numbers of Bernoulli samples.

A counter-based PRNG maps a user-supplied *counter* (plus, optionally, a *key*) to a fixed-size block of random bits via a deterministic function. Formally, if

$$\mathbf{x} = (x_1, x_2, \ldots, x_k)$$

is a vector of one or more 32-bit or 64-bit counters, and

$$\mathbf{k} \;=\; (k_1, k_2, \ldots, k_m)$$

is a key vector, then a counter-based PRNG defines a transformation

$$\mathcal{F}(\mathbf{x}, \mathbf{k}) \;=\; (\rho_1, \rho_2, \ldots, \rho_r),$$

where each ρ_j is typically a 32-bit or 64-bit output. Different increments of the counter $\mathbf{x}$ produce different pseudo-random outputs ρ_j. The process is stateless in the sense that advancing the RNG amounts to incrementing the counter (e.g., $\mathbf{x} \mapsto \mathbf{x} + 1$).

Compared to recurrence-based RNGs such as linear congruential generators or the Mersenne Twister, counter-based methods offer more straightforward parallelization, reproducibility across multiple streams, and strong structural simplicity: no internal state must be updated or maintained. This is particularly valuable in distributed Monte Carlo simulations or GPU-based sampling, where each thread or work-item can be assigned a different counter. Philox constructs its pseudo-random outputs by applying a small set of mixed arithmetic (multiplication/bitwise) rounds to an input *counter* plus *key.* In particular, Philox 4×32-10 (often shortened to ”Philox-4x32-10”) works on four 32-bit integers at a time:

$$\mathbf{S} = (S_0, S_1, S_2, S_3), \qquad \mathbf{K} = (K_0, K_1).$$

The four elements $\{S_0, S_1, S_2, S_3\}$ collectively represent the counter, e.g., (x_0, x_1, x_2, x_3). The two key elements (K_0, K_1) are used to tweak the generator’s sequence. A single invocation of Philox-4x32-10 transforms $\mathbf{S}$ into four new 32-bit outputs after ten rounds of mixing. At each round, the algorithm:

1. Multiplies two of the state words by fixed ”magic constants” to create partial products.

2. Takes the high and low 32-bit portions of those 64-bit products.

3. Incorporates the round key to shuffle the words.

4. Bumps the key by adding constant increments (W32A $= 0x9E3779B9$ and W32B $= 0xBB67AE85$).

After ten rounds, the final (S_0, S_1, S_2, S_3) is returned as the pseudo-random block. A new call to Philox increases the counter $\mathbf{S}$ by one (e.g., $S_3 \mapsto S_3 + 1$) and re-enters the same function. The Philox-4x32-10 algorithm is designed so that each blocking call requires a *constant number* of operations, independent of the size of any prior ”state.” Specifically, each round involves:

$$\mathcal{O}(1) \ \text{ arithmetic operations,}$$

and there are R = 10 rounds. Thus, each Philox invocation is asymptotically constant time $\mathcal{O}(\mathrm{R}) = \mathcal{O}(1)$. The total cost to generate 128 bits (4 words × 32 bits) is therefore constant time per call.

8.1 The 10-round Philox-4x32

Our implementation follows the standard 10-round approach for generating one block of four 32-bit random words, also called Philox-4x32-10. Let $M_{\mathrm{A}} = 0xD2511F53$, $M_{\mathrm{B}} = 0xCD9E8D57$ be the multipliers, and let (K_0, K_1) be the key which is updated each round by W32A $= 0x9E3779B9$ and W32B $= 0xBB67AE85$. The function $\mathrm{Hi}(\cdot)$ returns the high 32 bits of a 64-bit product, and $\mathrm{Lo}(\cdot)$ returns the low 32 bits. Because each call produces four 32-bit pseudo-random words, Philox-4x32-10 is particularly convenient for batched sampling. If only a single 32-bit word is needed, one can still call the function and discard the excess words; however, many applications consume all four outputs (e.g., to produce four floating-point variates).

Algorithm 1 Philox-4x32-10

```
Require: Four 32-bit counters $(S_0, S_1, S_2, S_3)$, key $(K_0, K_1)$
Ensure: Transformed counters $(S_0, S_1, S_2, S_3)$

procedure PHILOX_ROUND($(S_0, S_1, S_2, S_3)$, $(K_0, K_1)$)
    $P_0 \leftarrow M_{\mathrm{A}} \times S_0$                    ▷ 64-bit product
    $P_1 \leftarrow M_{\mathrm{B}} \times S_2$                    ▷ 64-bit product
    $T_0 \leftarrow \mathrm{Hi}(P_1) \oplus S_1 \oplus K_0$
    $T_1 \leftarrow \mathrm{Lo}(P_1)$
    $T_2 \leftarrow \mathrm{Hi}(P_0) \oplus S_3 \oplus K_1$
    $T_3 \leftarrow \mathrm{Lo}(P_0)$
    $K_0 \leftarrow K_0 + \mathrm{W32A}$
    $K_1 \leftarrow K_1 + \mathrm{W32B}$
    return $\big((T_0, T_1, T_2, T_3), (K_0, K_1)\big)$
end procedure

procedure PHILOX4X32_10($(S_0, S_1, S_2, S_3)$, $(K_0, K_1)$)
    for $i \leftarrow 1$ to 10 do
        $\big(S_0, S_1, S_2, S_3), (K_0, K_1) \leftarrow$ PHILOX_ROUND($(S_0, S_1, S_2, S_3)$, $(K_0, K_1)$)
    end for
    return $(S_0, S_1, S_2, S_3)$
end procedure
```

8.2 Bitpacking for Probability Sampling

It takes exactly one bit to represent the outcome of a Bernoulli trial. When these outcomes are stored naively, each occupies an entire 8-bit byte, so only a fraction $\frac{1}{8}$ of the allocated space carries useful information. Packing indicators into the native machine word of width w therefore reduces memory consumption by up to a factor of eight. More precisely,

$$\mathrm{Memory}_{\mathrm{naive}} \;=\; N \times 8\,\mathrm{bits}, \qquad \mathrm{Memory}_{\mathrm{packed}} \;=\; \left\lceil \frac{N}{w} \right\rceil \times w\,\mathrm{bits},$$

where N is the number of samples. In typical 64-bit environments we choose $w = 64$, but the derivation is architecture-agnostic.

8.2.1 Integer-threshold sampling.

Instead of converting each 32-bit random word r produced by Philox into a floating-point variate, we compare it directly to an *integer threshold.* Let the target probability be $p \in [0, 1]$. Define the threshold

$$T \;=\; \lfloor p \times 2^{32} \rfloor, \qquad 0 \leq T \leq 2^{32} - 1.$$

Because Philox delivers uniformly distributed 32-bit integers over $\{0, \ldots, 2^{32} - 1\}$, the event $r < T$ occurs with probability exactly p (up to the discretization of the 32-bit grid). The comparison is therefore sufficient to draw a Bernoulli outcome while avoiding the cost of division and floating-point arithmetic. Using integer thresholds also guarantees identical behavior across heterogeneous hardware, an essential property for reproducible high-performance simulations.

8.2.2 Grouping four comparisons.

Each invocation of Philox-4 × 32-10 yields the four words r_0, r_1, r_2, r_3. We evaluate the predicate $r_j < T$ for every index $j \in \{0, 1, 2, 3\}$ and collect the four resulting bits into a 4-bit block. The block is inserted into a wider accumulator using bitwise shifts. Repeating the procedure fills the entire w-bit container. If we denote by

$$g \;=\; \frac{w}{4}$$

the number of 4-bit generations required to populate the container, the overall cost is exactly g calls to the comparison–pack routine, or g additional increments of the counter component S_3 in Philox. No state other than the counter is maintained.

Algorithm 2 Integer-threshold packing of four Bernoulli outcomes

```
Require: Probability $p \in [0,1]$; 32-bit words $(r_0, r_1, r_2, r_3)$
Ensure: 4-bit integer bits containing the four Bernoulli draws
$T \leftarrow \lfloor p\, 2^{32} \rfloor$                              ▷ pre-compute once
bits $\leftarrow 0$
for $j \leftarrow 0$ to $3$ do
    if $r_j < T$ then
        $b_j \leftarrow 1$
    else
        $b_j \leftarrow 0$
    end if
    bits $\leftarrow$ bits $\mid (b_j \ll j)$
end for
return bits
```

8.2.3 Assembling a complete bitpack.

Calling Algorithm 2 successively for g generations yields a full w-bit word whose k-th bit encodes the outcome of the k-th Bernoulli trial, ordered least-significant first. The procedure is branch-free except for the threshold comparison and operates in $\mathcal{O}(g)$ time. Because g is fixed by the word size, the complexity is effectively constant per stored word, and the memory savings are realized without sacrificing statistical quality or portability.

8.3 Counter Assignment Across the ND-Range

Let $(i_x, i_y, i_z) \in [0, G_x) \times [0, G_y) \times [0, G_z)$ denote the global id of a work-item in the basic-event kernel and let $t \in \{0, \dots, T-1\}$ be the Monte-Carlo iteration index. A collision-free family of Philox counters is obtained via

$$\mathbf{C}(i_x, i_y, i_z, t) \;=\; \big(i_x + 1,\, i_z + 1,\, i_y + 1,\, (t + 1) \ll 6\big),$$

where the six vacated least-significant bits of the fourth word are reserved for intra-kernel increments. The mapping is bijective onto

$$\{1, \ldots, V\} \times \{1, \ldots, P\} \times \{1, \ldots, B\} \times \{1, \ldots, 2^6\},$$

guaranteeing independent pseudo-random streams for every work-item without synchronization or shared state.

Chapter 9

Gate Kernels for Bit-Packed Boolean Evaluation

9.1 Connective Taxonomy

Let $\mathcal{G}$ be the set of Boolean gates obtained from the topological analysis of Section 7.2. Each gate $g \in \mathcal{G}$ is represented by the triplet

$$g = \big(\,\mathrm{type}(g),\ \mathcal{I}^{+}(g),\ \mathcal{I}^{-}(g)\big),$$

where $\mathcal{I}^{+}(g)$ and $\mathcal{I}^{-}(g)$ denote its positive and negated inputs. We partition $\mathcal{G}$ into disjoint subsets $\mathcal{G}_{\mathrm{type}}$ according to

$$\mathrm{type}(g) \in \big\{\textsc{Null}, \textsc{Not}, \textsc{And}, \textsc{Or}, \textsc{Xor}, \textsc{Nand}, \textsc{Nor}, \textsc{Xnor}, \textsc{Atleast}\big\}.$$

The subsequent sections analyze one subset at a time so that device kernels remain branch-free and resource usage is homogeneous.

9.2 Launch Geometry

For every connective type we schedule one kernel with global range

$$(G_x, G_y, G_z) = \big(\lceil N_g \rceil,\ B,\ P\big), \qquad N_g = |\mathcal{G}_{\text{type}}|,$$

rounded by the nearest-multiple rule of Section 7.4. A work-item with global id (i_x, i_y, i_z) therefore processes the unique triplet $(g, b, p) \in \mathcal{G}_{\text{type}} \times \{0, \ldots, B-1\} \times \{0, \ldots, P-1\}$. Local ranges (L_x, L_y, L_z) are chosen by the heuristic of Section 7.3.7 and refined in Section 9.5.

9.3 Bit-Parallel Reduction Schemes

9.3.1 Idempotent Connectives: AND/OR Families

For AND, OR and their complements the kernel performs a word-wise left fold over the input list. The accumulator is initialized as

$$R_0 = \begin{cases} \texttt{AllOnes}, & \textsc{And}/\textsc{Nand}, \\ \\ \texttt{Zero}, & \textsc{Or}/\textsc{Nor}. \end{cases}$$

Positive inputs use $R \leftarrow R \otimes v$ with $\otimes \in \{\&, |\}$; negated inputs substitute $\neg v$.

Lemma 5 (Bit-wise Idempotence). *For any word size ω and any assignment of the input bits,*

$$R_{\text{final}} = \bigotimes_{u \in \mathcal{I}^+(g)} u \bigotimes_{v \in \mathcal{I}^-(g)} \neg v$$

yields a correct bit-packed representation of gate g.

Proof. Idempotence of $\wedge$ and $\vee$ ensures order-independent accumulation. Per-bit complement commutes with both operators, preserving semantics. □

The instruction count is $C_{\text{idemp}} = (\deg g)\,\Theta(1)$, one mask operation per input, independent of ω.

9.3.2 Parity Connectives: Xor/Xnor

The fold operator becomes $\oplus$. Associativity allows work-groups to split the input list and apply warp-level reductions, lowering register pressure for large fan-ins (Section 9.5). A final complement realizes Xnor.

9.3.3 Threshold Connectives: At-Least k

Let $n = \deg g$ and $k \in \{0, \ldots, n\}$. Fix the word width $\omega = 8\,\text{sizeof}(\texttt{bitpack\_t})$ and launch each work-group with $L_z = \omega$ so that lane $\lambda \in \{0, \ldots, \omega - 1\}$ owns one bit position.

1. *Per-lane counting*: initialise $c_\lambda \leftarrow 0$; stream through the inputs, incrementing c_λ whenever the masked bit is set (positive input) or cleared (negated input).

2. *Threshold test*: $r_\lambda \leftarrow [c_\lambda \geq k]$.

3. *Group reduction*: a lane-wise OR assembles the word $R = \sum_\lambda r_\lambda 2^\lambda$.

Theorem 6 (Work-Group Correctness)**.** *With the above geometry each work-group writes exactly one valid output word per iteration.*

Proof. Bijectivity of the mapping $(\text{group}, \text{lane}) \mapsto (p, \lambda)$ guarantees single-writer semantics; Steps 1–3 implement the at-least k predicate bit-wise. □

The per-lane cost is n conditional increments plus one comparison; adding the $\log_2 \omega$-step OR tree yields $C_{\text{thr}} = \Theta(n + \log \omega)$.

9.4 Performance Models

For idempotent and parity families let $I = n$ and memory traffic $M = n$. Using B_{mem} and λ from Sections 11.1–11.2,

$$\text{IPS} \;\le\; \min\!\Big(\frac{B_{\text{mem}}}{Mw},\; \frac{C\,\lambda f}{I}\Big),$$

where w is word size in bytes. Threshold gates replace $I \leftarrow n + \log\omega$.

9.5 Work-Group Optimization Heuristics

Empirically, gates with $\deg g > 64$ profit from lane-parallel counting whereas smaller fan-ins prefer maximal l_y, l_z to saturate memory bandwidth:

$$(l_x, l_y, l_z) = \begin{cases} (1,\ B,\ P), & \deg g > 64, \\ \\ (1,\ \min(B, L_{\max}),\ \min(P, L_{\max}/B)), & \text{otherwise.} \end{cases}$$

9.6 Complexity

$$C_\Gamma = \begin{cases} \Theta(n), & \text{idempotent/parity,} \\ \\ \Theta(n + \log\omega), & \text{at-least}\, k. \end{cases}$$

Aggregated over all gates the arithmetic cost is $\mathcal{O}\big(G\,n_{\text{avg}}BP\big)$.

Chapter 10

Tallying Layer Outputs

At every Monte-Carlo iteration the simulator produces, for each logic node $v \in \mathcal{V}$, a bit-packed buffer encoding

$$\mathbf{Y}_v^{(t)} \;=\; \big(y_{v,1}^{(t)}, y_{v,2}^{(t)}, \ldots, y_{v,N}^{(t)}\big) \in \{0,1\}^N, \quad t = 1, \ldots, T,$$

where $N = B \times P \times \omega$ is the number of Bernoulli trials per Monte-Carlo *iteration*:

- B - number of *batches*,
- P - bit-packs per batch,
- $\omega = 8 \cdot \text{sizeof(bitpack\_t)}$ - bits per pack.

Because the buffers are overwritten at the next iteration, a separate *tally layer* accumulates summary statistics that persist for the entire simulation. The present section formalizes that process and outlines an implementation-agnostic, data-parallel algorithm that realizes it on modern accelerators.

10.1 Notation and Problem Setting

We inherit the global symbols (T, B, P, ω) from Section 7.4. Recall that one Monte–Carlo *iteration* generates $N = B\,P\,\omega$ Bernoulli trials per logic node. Index those trials as

$$\Xi \;=\; \big\{(t, b, p, \lambda) \mid t \in [T],\; b \in [B],\; p \in [P],\; \lambda \in [\omega]\big\},$$

where $[n] \equiv \{0, \ldots, n-1\}$. For every node $v \in \mathcal{V}$ the kernel produced in Section 7.3.5 writes a bit–packed vector

$$\mathbf{Y}_v : \Xi \longrightarrow \{0, 1\}, \quad (t, b, p, \lambda) \mapsto y_{v,t,b,p,\lambda}.$$

Grouping by iteration we recover the notation $\mathbf{Y}_v^{(t)} \in \{0, 1\}^N$ used earlier. The tally routine operates on the scalar summaries

$$s_v \;=\; \sum_{(t,b,p,\lambda)\,\in\,\Xi} y_{v,t,b,p,\lambda}, \qquad v \in \mathcal{V},$$

which are accumulated in 64–bit integers to avoid overflow even for trillion–sample runs.

10.2 Statistical Objectives

The goal is to estimate the Bernoulli success probability $p_v = \mathbb{P}[Y_v = 1]$ for every node $v \in \mathcal{V}$. Denote by

$$\widehat{p}_v \;=\; \frac{s_v}{T\,N}$$

its empirical frequency after T iterations.

Lemma 7 (Unbiasedness)**.** $\mathbb{E}[\widehat{p}_v] = p_v$ *and* $\mathrm{Var}(\widehat{p}_v) = \dfrac{p_v(1-p_v)}{T\,N}$.

Proof. Because distinct trials in Ξ are independent and identically distributed Bernoulli(p_v) variables, linearity of expectation and the variance–additive property of independent sums yield the stated expressions. □

Theorem 8 (Asymptotic Normality)**.** *If* $T\,N \to \infty$ *then*

$$\sqrt{T\,N}\,(\widehat{p}_v - p_v) \;\xrightarrow{\mathcal{D}}\; \mathcal{N}\big(0,\, p_v(1-p_v)\big).$$

Proof. The Lyapunov condition of the Central Limit Theorem holds for bounded Bernoulli variables; see, e.g., [20, Th. 27.4]. The scaled sum therefore converges in distribution to the stated normal law. □

Define the plug–in variance estimator

$$\widehat{\sigma}_v^2 \;=\; \frac{\widehat{p}_v(1-\widehat{p}_v)}{T\,N}.$$

Combining Theorem 8 with Slutsky's theorem yields the $(1-\alpha)$ confidence interval

$$C_{v,\alpha} \;=\; \big[\widehat{p}_v - z_{1-\alpha/2}\,\widehat{\sigma}_v,\; \widehat{p}_v + z_{1-\alpha/2}\,\widehat{\sigma}_v\big].$$

Clamping the bounds to $[0,1]$ ensures numerical stability when s_v is very small or very close to $T\,N$.

In practice the kernel therefore stores only the integer accumulator s_v; all higher–level statistics are derived á posteriori on demand, incurring $\mathcal{O}(|\mathcal{V}|)$ host–side work.

10.3 Parallel Accumulation Algorithm

The accumulation kernel is invoked on a three-dimensional `nd_range`, chosen such that

$$\begin{aligned} \text{global}_x &\;\ge\; V, \\ \text{global}_y &\;\ge\; B, \\ \text{global}_z &\;\ge\; P. \end{aligned}$$

Work-item (i_x, i_y, i_z) is responsible for *exactly one* bit-pack:

$$\text{node } v = i_x, \quad \text{batch } b = i_y, \quad \text{pack } p = i_z.$$

Local workflow of a work-item

1. Load the p^{th} bit-pack of batch b from `buffer`.

2. Compute $c = \text{popcount}(\text{bitpack})$.

3. Reduce the c's belonging to the same work-*group* in shared memory (tree reduction or `reduce_over_group`).

4. One designated leader performs `atomic_add`(`num_one_bits`, group_sum).

The reduction ensures only one atomic operation per group, greatly reducing contention when P is large.

We present platform-neutral pseudocode that encapsulates the above logic while remaining agnostic to the underlying API. After each Monte-Carlo iteration the host enqueues TallyKernel with a fresh `iteration` counter. When either (i) a user requests intermediate statistics or (ii) a pre-set reporting interval is reached, the host reads back `num_one_bits` and executes the purely serial routine shown in Algorithm 3.

Algorithm 3 Post-processing of a single node's tally

```
**Require:** $s$ - total one-bits, $T$, $B$, $P$, $\omega$ - run parameters
**Ensure:** $\widehat{p}$, $\widehat{\sigma}$, two symmetric CIs
$N \leftarrow B \cdot P \cdot \omega$
$\widehat{p} \leftarrow s/(T\,N)$
$\widehat{\sigma} \leftarrow \sqrt{\widehat{p}(1-\widehat{p})/(T\,N)}$
**for each** $z \in \{1.96,\ 2.58\}$ **do**
    CI $\leftarrow \big[\max(0, \widehat{p} - z\widehat{\sigma}), \min(1, \widehat{p} + z\widehat{\sigma})\big]$
**end for**
```

The above normal approximation is valid provided $T\,N\widehat{p}$ and $T\,N(1-\widehat{p})$ both exceed roughly 10; otherwise an exact Clopper-Pearson interval can be substituted with no change to the running sum logic.

10.4 Correctness and Complexity

Work-item cost. Each work-item performs one popcount and participates in an $O(\log L)$ intra-group reduction (L=local_range), yielding an overall $O(\log L)$ instruction count.

Global cost. The total number of work-items launched per iteration is $V \cdot B \cdot P$. Because each bit-pack contains ω Bernoulli trials, the cost *per trial* shrinks as ω^{-1}.

Memory traffic. Every work-item reads exactly one machine word and no writes occur except the single atomic addition per work-group. Hence the algorithm is memory-bandwidth bound only at extremely low arithmetic intensity ($P \approx 1$).

Linear scalability. All tally nodes are independent. Increasing V therefore scales the total runtime linearly until either (i) the device saturates its occupancy or (ii) atomic contention becomes non-negligible; the group-level reduction mitigates the latter.

The design therefore provides a clear separation of concerns: depth–first analysis establishes the dependency structure; kernel generation translates that structure into homogeneous, vectorizable work; and a light–weight event system schedules the resulting kernels with minimal host intervention.

10.5 Work–Group Geometry and Synchronization

The three–dimensional launch geometry $(v,\, b,\, p)$ outlined in Sec. 10.2 is refined in the implementation to minimize both occupancy loss and atomic contention. A crucial design choice is to fix *local* $x = 1$, thereby dedicating one work–group to exactly one tally node v. The remaining two dimensions then tile the (b, p)–plane with a rectangular block of size

$$\bigl(1,\, l_y,\, l_z\bigr), \qquad l_y \cdot l_z \le L_{\max},$$

where $L_{\max}$ is the device–specific upper bound on the total work–group size. Provided $l_y \ge B$ and $l_z \ge P$, only *one* group is dispatched per tally and per iteration, guaranteeing that the reduction of Step 3 and the atomic addition of Step 4 in Sec. 10 execute exactly once. Should resource pressure force $l_y < B$ or $l_z < P$, multiple groups are launched and the atomic update is replicated; correctness is preserved by the commutativity of addition, but the repeated work incurs a small overhead. The occupancy model therefore trades a moderate loss in parallelism for deterministic behavior and reduced synchronization cost.

A relaxed memory order is sufficient for the atomic accumulator because the kernel guarantees *program order* between the intra–group reduction and the atomic `fetch_add`. No additional fences are required, and the resulting implementation maps efficiently to both discrete and integrated GPUs.

10.6 Incremental Update of Derived Statistics

While Monte–Carlo sampling proceeds, applications often request intermediate probability estimates $\widehat{p}_v^{(t)}$ before the total budget T is exhausted. Recomputing $\widehat{p}_v$ and $\widehat{\sigma}_v$ from scratch would require a host round–trip for every sampled bit. Instead, the tally layer maintains two scalars per node: s_v (total one–bits) and n_v (total bits processed). After each completed iteration the host merely increments $n_v \leftarrow n_v + N$ and leaves s_v to the device kernel. Whenever a refresh is requested the statistics are updated via

$$\widehat{p}_v \;=\; \frac{s_v}{n_v}, \qquad \widehat{\sigma}_v \;=\; \sqrt{\frac{\widehat{p}_v(1-\widehat{p}_v)}{n_v}},$$

which costs $\mathcal{O}(V)$ host–side arithmetic and no device work. In practice the refresh cadence is set adaptively: frequent updates early in the run aid variance monitoring, whereas late–stage updates can be spaced further apart because the relative change in $\widehat{p}_v$ diminishes as $n_v \to T\,N$.

10.7 Convergence Diagnostics and Stopping Rules

Two families of diagnostics leverage the quantities already maintained by the tally kernel:

1. **Relative half–width criterion.** Define the half–width of the $(1-\alpha)$–level interval as $h_v = z_{1-\alpha/2}\,\widehat{\sigma}_v$. The run may be terminated for node v once $h_v/\widehat{p}_v \le \varepsilon$, where ε is a user–supplied tolerance. Because both $\widehat{\sigma}_v$ and $\widehat{p}_v$ are inexpensive to update, the test incurs negligible overhead.

2. **Sequential Wald test.** When the goal is to decide whether p_v exceeds a safety threshold p_0, one may adopt the sequential probability ratio test with boundaries derived from s_v and n_v. The tally structure already provides the minimal sufficient statistics, so the host evaluates the Wald condition after every refresh with no additional device interaction.

Because the diagnostics rely solely on s_v and n_v, no modification to the kernel is needed; all logic resides in a lightweight host callback.

10.8 Implementation Cost Model

Let C_{pc} denote the latency of a hardware popcount and $C_{\text{rd}}(l)$ the latency of a tree reduction over l work–items. The wall–clock time per iteration is approximated by

$$T_{\text{iter}} \;\approx\; (C_{\text{pc}} + C_{\text{mem}})\,VBP + C_{\text{rd}}(l_y l_z)\,\frac{VBP}{l_y l_z} + C_{\text{atomic}}\,\frac{VBP}{l_y l_z},$$

where C_{mem} and C_{atomic} are the per–word memory and atomic latencies, respectively. The model highlights two regimes:

- *Arithmetic bound*: when $P \gg 1$ and the popcount throughput saturates the execution units, the first term dominates and scaling is limited by instruction bandwidth.
- *Memory bound*: when $P \approx 1$ the workload collapses to a single read per work–item; the kernel becomes memory bandwidth– limited as predicted in Sec. 10.2.

10.9 Numerical Robustness

All accumulators operate in integer arithmetic, thereby eliminating rounding error in s_v. Derived quantities computed in double precision satisfy $|\widehat{p}_v - s_v/n_v| < 2^{-53}$, well below any practical error criterion for reliability analysis. Clamping the confidence interval bounds to $[0, 1]$ prevents pathological estimates when either $s_v = 0$ or $s_v = n_v$ in early iterations.

10.10 Relation to the Global Execution Model

The specialized geometry adopted in Section 10.5 is a direct instantiation of the rules formalized in Section 7.4. Choosing $L_x = 1$ enforces the work−group invariant whereby a group owns exactly one tally node while still satisfying

$$G_x \;=\; \left\lceil \frac{V}{L_x} \right\rceil L_x \;=\; V,$$

so no over-provisioning occurs along the x-axis. The remaining dimensions follow the generic rounding scheme with $(Q_y, Q_z) = (B, P)$, thus preserving the one-to-one correspondence between work-items and bit-packs established in Section 7.4.

Chapter 11

Backend–Specific Scalability Analysis

In this section we instantiate the abstract execution model of Section 7.4 on two concrete hardware backends that span the current spectrum of commodity accelerators: *(i)* NVIDIA GPUs programmed through the CUDA tool-chain and *(ii)* shared-memory multicore CPUs equipped with wide SIMD units. The discussion follows the roofline methodology [86] wherever a bandwidth–or–compute bottleneck must be highlighted and retains the global kernel symbols introduced previously. Additional backend parameters are summarized in Table 11.1; architectural constants are set in *italic type*.

Throughout we denote by $L = L_x L_y L_z$ the total number of work-items in a SYCL work-group and by $W = W_x W_y W_z$ the total number of work-groups launched by the kernel.

11.1 CUDA GPU backend

11.1.1 Thread-Block Mapping

Each SYCL work-group is lowered to a CUDA *thread block*. The effective block size used by the hardware is therefore

$$L_{\text{CUDA}} = \min(L, T_{\max}).$$

Table 11.1: Backend parameters introduced in this section. Architectural constants are shown in *italic*.

Symbol	Meaning
C	physical compute units (SMs on CUDA, cores on CPU)
W_s	SIMD-lane width (”warp” size on CUDA; 32 on recent GPUs)
$T_{\max}$	maximum resident work-items per work-group / block
$B_{\max}$	scheduler limit on concurrent blocks per compute unit
$R_{\max}$	registers available per compute unit
B_{mem}	attainable device memory bandwidth
f	core clock frequency (Hz)

The kernel grid retains the user-specified dimensions (W_x, W_y, W_z) and thus launches W blocks in total. A block of size L_{CUDA} contains $\lceil L_{\text{CUDA}}/W_s \rceil$ warps.

11.1.2 Theoretical Occupancy

A standard proxy for latency hiding on GPUs is the *occupancy* $\mathcal{O}$, i.e. the ratio between active and maximum resident warps per SM. With

$$W_{\text{act}} \;=\; \min\!\Big(B_{\max},\, \big\lceil \tfrac{L_{\text{CUDA}}}{W_s} \big\rceil\Big) \times \big\lceil \tfrac{W}{C} \big\rceil$$

active per-SM warps, the theoretical occupancy evaluates to

$$\mathcal{O}_{\text{th}} \;=\; \min\!\Big(1,\, \frac{W_{\text{act}}}{B_{\max} T_{\max}/W_s}\Big).$$

Given that realistic Monte-Carlo workloads satisfy $W \gg C$ the rightmost fraction approaches 1, and kernels are typically either register- or shared-memory-limited rather than scheduler-limited.

11.1.3 Register Footprint and Latency Hiding

Let R denote the per-thread register footprint measured by the compiler and $W_{\text{reg}} = \lfloor R_{\max}/(RW_s) \rfloor$ the register-constrained warp capacity per SM. The empirical latency hiding factor on modern NVIDIA hardware can be captured by

$$\lambda_{\text{CUDA}} = \min(W_{\text{reg}},\, W_s B_{\max}),$$

which saturates instruction throughput once $\lambda_{\text{CUDA}} \gtrsim 4$.

11.1.4 Throughput Model

Let I be the static instruction count per thread derived in Sec. 7.4. Assuming that the kernel is compute-bound the sustained instruction rate (IPS) is approximated by

$$\text{IPS}_{\text{CUDA}} \;\approx\; \frac{C\,\lambda_{\text{CUDA}}\,f}{I}.$$

The expression is linear in both the number of compute units C and the latency hiding factor λ_{CUDA} until the memory subsystem is saturated; the break-even point is estimated in the roofline plot of Fig. **??** (omitted here for brevity).

11.2 Shared-Memory Multicore CPU Backend

11.2.1 Work-Group to Thread Mapping

On CPUs a SYCL work-group is translated to an OpenMP `parallel for` *team*. The default team size equals L but cannot exceed the architectural limit $T_{\max} = W_s$. The outermost loop distributes the W work-groups over the available hardware threads, i.e. over C physical cores and their simultaneous multithreading (SMT) contexts.

11.2.2 Vectorization Strategy

The innermost kernel dimension (global z) holds independent bit-pack indices. Mapping that dimension to SIMD lanes yields perfect utilization as long as the kernel exposes at least W_s independent bit-packs, which is guaranteed for the Monte-Carlo sample sizes considered ($\omega \geq W_s$).

11.2.3 Roofline Bound

With b bytes and i floating-point instructions issued per trial the classical roofline model bounds the attainable performance by

$$P_{\text{CPU}} \;\leq\; \min\!\Big(\frac{B_{\text{mem}}}{b},\; \frac{C\, I_{\text{F}}}{i}\Big),$$

where I_{F} denotes the peak per-core fused-multiply-add (FMA) rate. For the present kernels the operational intensity $i/b \approx 0.25\,\text{FMA/B}$ places almost all CPU runs in the bandwidth-bound regime unless the sample count per node exceeds 10^{10}, well beyond typical reliability studies.

11.2.4 Strong-Scaling Perspective

Holding the global problem size fixed while increasing the core count leads to a speed-up characterized empirically by

$$S(C) \;=\; \frac{T_1}{T_C} \;\approx\; \frac{C}{1 + \alpha(C-1)},$$

where the serial fraction $\alpha \leq 0.05$ was obtained on a 64-core Zen4 system. The Amdahl limit $1/\alpha$ therefore exceeds the practical core counts of current workstation-class CPUs.

11.2.5 Practical Parameter Choices

Extensive auto-tuning on a 64-core Zen4 host and an NVIDIA Ada GPU suggests

Kernel class	GPU (CUDA)	CPU (OpenMP)
Tally	$(L_x, L_y, L_z) = (1,\, W_s,\, 1)$	$(1,\, W_s,\, 1)$
Gate	$(1,\, 1,\, W_s)$	$(1,\, 1,\, W_s)$

which aligns the innermost loop with the cache line size and the SIMD width on both architectures.

Chapter 12

Preliminary Benchmarks on Arialia Fault Trees

12.1 Runtime Environment and Benchmarking Setup

All experiments were performed on a consumer-grade desktop provisioned with an NVIDIA® GeForce GTX 1660 SUPER graphics card (1,408 CUDA cores, 6 GB of dedicated GDDR6 memory) and a 10th-generation Intel® Core™ i7-10700 CPU (2.90 GHz base clock, with turbo-boost and hyperthreading enabled). The code implementation relies on SYCL using the AdaptiveCpp (formerly HipSYCL) framework, which employs an LLVM based runtime and just-in-time (JIT) kernel compilation.

Monte Carlo Sampling Strategy

Each fault tree model was evaluated through a single pass (one iteration), generating as many Monte Carlo samples as would fit into the GPU's 6 GB memory. A 64-bit counter-based Philox4x32x10 random number generator was applied in parallel to produce the basic-event realizations. Note, with the exception of `das9205`, for which 5 passes were performed (in ≈ 0.96 seconds), all inputs were quantified using just one pass. We specifically chose `das9205` since its overall event probability is quite low, and requires many naive Monte Carlo samples.

Bit-Packing and Data Types

To reduce memory usage and increase vectorized throughput, every batch of Monte Carlo results was bit-packed into 64-bit words. Accumulated tallies of successes or failures were stored as 64-bit integers, while floating-point calculations (e.g., probability estimates) used double precision (64-bit floats). These design decisions are intended to maintain numerical consistency and make use of native hardware operations (such as population-count instructions for threshold gates).

Execution Procedure

Upon launching the application, the enabling overhead (host-device transfers, JIT compilation, and kernel configuration) was included in the total wall-clock measurement. Each benchmark was compiled at the `-O3` optimization level to ensure efficient instruction generation. Every experiment was repeated at least five times, and measured runtimes were averaged to reduce the impact of transient background processes or scheduling variations on the host system.

12.2 Assumptions and Constraints

The primary objective was to gauge runtime across a set of fault trees that vary widely in size, logic complexity, and probability ranges within a typical Monte Carlo integration workflow. The experiments assume independent operation of the test machine, with no significant other processes contending for GPU or CPU resources. All sampling took place within a single pass, so the measured wall times incorporate initial kernel launches, memory copies, and statistical collection of gate outcomes. No specialized forms of hardware optimization beyond the data-parallel approach (e.g., pinned memory or asynchronous streams) were used.

12.3 Comparative Accuracy & Runtime

Table 12.1 and Figure 12.1 summarize the accuracy of three approximate quantification methods Rare-Event Approximation (REA), Min Cut Upper Bound (MCUB), and our GPU-accelerated Monte Carlo by listing each approach's mean relative error in the log-probability ($\log p$) domain, alongside the total MC samples and runtime. Although each fault tree exhibits its own complexities, several broad trends emerge:

1. **REA accuracy strongly depends on the *actual* top-event probability.**

    - For trees with very low-probability failures (e.g., `baobab1`, `das9202`, `isp9605`), where individual component failures rarely coincide, REA's mean error often remains near or below 10^{-2} in log space. This indicates that summing only the first-order minimal cut sets–assuming higher-order intersections contribute negligibly–can be valid when the system is indeed dominated by single-component or few-component events.
    - However, for fault trees with moderate or higher top-event probabilities ($\gtrsim 10^{-2}$), REA's inaccuracy tends to grow (for instance, up to 10^{-1} in `edf9203`, `edf9204`, and `edfpa15b`). In these cases, ignoring the overlap of multiple cut sets leads to a visible systematic error.

2. **Min-Cut Upper Bound (MCUB) often mirrors REA but with exaggerated errors in certain overlapping cut configurations.**

    - In many models (e.g., `cea9601`, `baobab3`, `das9601`), MCUB closely tracks REA, suggesting that higher-order combinations remain negligible in those systems.
    - Yet, in a few cases involving heavy cut-set overlap (e.g., `das9209`, row 14), MCUB soars to a mean log-probability error of ~ 17, dwarfing REA or Monte Carlo. This highlights the well-known pitfall: if multiple cut sets are not genuinely "rare" and substantially overlap, the union bound becomes extremely loose.

3. **Monte Carlo yields more consistent and often dramatically lower numerical errors for most moderate- to high-probability top events.**

    - For example, in `das9201` (row 6) and `edf9203` (row 19), the Monte Carlo error is well below 10^{-3}, whereas both REA and MCUB can exceed 10^{-1}. In these situations, ignoring or bounding higher-order intersections proves inadequate, while direct sampling naturally captures all overlaps.

    - However, for fault trees with extremely small top-event probabilities, Monte Carlo's variance can become harder to control. For instance, some rows (`das9204`, `das9205`, `isp9605`, `isp9607`) show that roughly 10^8–10^9 samples are required to constrain the error within a few tenths in $\log p$. Those entries either exhibit a slightly higher Monte Carlo error than REA/MCUB or demonstrate that we needed a disproportionately large sample count (and thus more runtime) to compete with simple rare-event approximations.

4. **Sampling scale and runtime remain surprisingly feasible, even for up to** 10^9 **draws.**

    - Despite some test cases sampling in the hundreds of millions or billions, runtimes remain ≈ 0.2–0.3 s for most fault trees, rarely exceeding 1 s (see, for instance, row 10 with 3.3 B samples and ~ 0.96 s). This indicates that the bit-packed, data-parallel Monte Carlo engine is highly optimized, making large-sample simulation a viable alternative to purely analytical approaches for many real-world PRA problems.

    - By contrast, the bounding methods (REA and MCUB) typically run in negligible time but deliver inconsistent accuracy depending on each tree's structure. In practice, a hybrid strategy may emerge: apply bounding methods for quick estimates, then selectively invoke large-sample Monte Carlo for trees or subsections where the bounding approximation diverges.

5. **Omitted or Extreme Cases.**

    - Rows where Monte Carlo entries are missing (e.g., `das9209` and `edf9206`) indicate difficulty in converging to a useful estimate within a fixed iteration budget. Conversely, MCUB shows erratic jumps in some of those same cases, underlining the fact that both bounding and sampling approaches can struggle in certain outliers.
    - Model `nus9601` (row 43) lacks all three error columns since no reference solution was available, reflecting a scenario where direct verification remains pending or inapplicable. Nevertheless, the completion time of $\sim$ 0.29 s for a partial exploration suggests that the structural overhead of large fault trees can still be handled efficiently.

These results affirm that Monte Carlo methods, when equipped with high throughput sampling, can achieve the most robust accuracy across a broader spectrum of top-event probabilities, particularly in configurations where standard cut set approximations fail to capture significant event dependencies. At the same time, rare-event with exceptionally small probabilities can pose challenges for naive sampling, revealing the potential need for adaptive variance-reduction techniques or partial enumerations. In practice, analysts may combine bounding calculations (REA/MCUB) for quick screening or preparatory checks, then use hardware-accelerated Monte Carlo to refine those domains most susceptible to underestimation or overestimation by simpler approximations. Alternatively, for very large models, where exact solutions may be unavailable, data-parallel Monte Carlo can still estimate event probabilities without building minimal cut sets.

Table 12.1: Relative error (Log-probability), Data-Parallel Monte Carlo (DPMC) vs Min Cut Upper Bound (MCUB) and Rare-Event Approximation (REA).

#	**Fault Tree**	**Relative Error** $\left\|\log_{10}\left(\frac{P_{\mathbf{approx}}}{P_{\mathbf{exact}}}\right)\right\|$			**Samples**	**Runtime** (s)
		REA	**MCUB**	**DPMC**		
1	baobab1	1.46×10^{-4}	1.46×10^{-4}	7.62×10^{-3}	2.6×10^{8}	0.27
2	baobab2	6.49×10^{-3}	6.35×10^{-3}	1.55×10^{-3}	2.6×10^{8}	0.21
3	baobab3	1.22×10^{-2}	1.17×10^{-2}	2.25×10^{-4}	2.5×10^{8}	0.26
4	cea9601	9.37×10^{-2}	9.33×10^{-2}	2.42×10^{-3}	1.3×10^{8}	0.27
5	chinese	1.09×10^{-2}	1.07×10^{-2}	2.15×10^{-3}	9.5×10^{8}	0.28
6	das9201	1.27×10^{-1}	1.23×10^{-1}	5.50×10^{-5}	2.4×10^{8}	0.28
7	das9202	7.73×10^{-5}	2.58×10^{-5}	1.21×10^{-4}	5.3×10^{8}	0.30
8	das9203	3.60×10^{-2}	3.56×10^{-2}	2.32×10^{-4}	5.3×10^{8}	0.30
9	das9204	1.69×10^{-1}	1.69×10^{-1}	1.14×10^{-1}	6.2×10^{8}	0.30
10	das9205	9.64×10^{-2}	9.64×10^{-2}	2.77×10^{-2}	3.4×10^{9}	0.96
11	das9206	5.44×10^{-2}	8.90×10^{-4}	3.52×10^{-4}	2.1×10^{8}	0.27
12	das9207	1.19×10^{-1}	2.46×10^{-2}	1.37×10^{-4}	9.6×10^{7}	0.29
13	das9208	4.13×10^{-2}	3.82×10^{-2}	9.35×10^{-5}	2.6×10^{8}	0.31
14	das9209	2.12×10^{-2}	1.71×10^{1}			
15	das9601	5.30×10^{-2}	5.20×10^{-2}	6.68×10^{-4}	1.2×10^{8}	0.26
16	das9701	5.03×10^{-2}	3.38×10^{-2}	6.23×10^{-4}	2.4×10^{7}	0.28
17	edf9201	1.49×10^{-1}	5.37×10^{-2}	2.89×10^{-4}	1.9×10^{8}	0.32
18	edf9202	1.08×10^{-1}	6.06×10^{-3}	4.54×10^{-4}	7.9×10^{7}	0.28
19	edf9203	2.23×10^{-1}	1.18×10^{-1}	3.28×10^{-4}	8.1×10^{7}	0.31
20	edf9204	2.80×10^{-1}	1.06×10^{-1}	1.32×10^{-4}	8.8×10^{7}	0.30
21	edf9205	9.95×10^{-2}	4.47×10^{-2}	5.61×10^{-5}	2.0×10^{8}	0.29

Table 12.1: Relative Error (Log-Probability), DPMC vs MCUB, REA.

#	**Fault Tree**	**Relative Error** $\left\|\log_{10}\left(\frac{P_{\mathbf{approx}}}{P_{\mathbf{exact}}}\right)\right\|$			**Samples**	**Runtime** (s)
		REA	**MCUB**	**DPMC**		
22	edf9206	6.99×10^{-3}	7.08×10^{-3}			
23	edfpa14b	1.86×10^{-1}	9.16×10^{-2}	1.05×10^{-3}	9.5×10^{7}	0.27
24	edfpa14o	1.87×10^{-1}	9.19×10^{-2}	3.40×10^{-4}	9.9×10^{7}	0.28
25	edfpa14p	3.41×10^{-2}	1.67×10^{-2}	5.36×10^{-4}	2.2×10^{8}	0.30
26	edfpa14q	1.86×10^{-1}	9.16×10^{-2}	3.34×10^{-4}	9.7×10^{7}	0.29
27	edfpa14r	2.49×10^{-2}	2.10×10^{-2}	9.34×10^{-4}	2.2×10^{8}	0.30
28	edfpa15b	2.17×10^{-1}	9.38×10^{-2}	4.68×10^{-4}	1.2×10^{8}	0.29
29	edfpa15o	2.17×10^{-1}	9.38×10^{-2}	4.07×10^{-5}	1.2×10^{8}	0.29
30	edfpa15p	2.53×10^{-2}	1.01×10^{-2}	3.55×10^{-4}	2.7×10^{8}	0.30
31	edfpa15q	2.17×10^{-1}	9.38×10^{-2}	6.75×10^{-4}	1.2×10^{8}	0.29
32	edfpa15r	1.95×10^{-2}	1.63×10^{-2}	4.05×10^{-4}	2.6×10^{8}	0.30
33	elf9601	1.99×10^{-2}	8.09×10^{-5}	7.87×10^{-5}	2.4×10^{8}	0.28
34	ftr10	1.23×10^{-1}	9.28×10^{-4}	1.55×10^{-4}	2.2×10^{8}	0.30
35	isp9601	8.09×10^{-2}	6.64×10^{-2}	1.14×10^{-4}	1.9×10^{8}	0.28
36	isp9602	1.75×10^{-2}	1.48×10^{-2}	1.36×10^{-3}	2.4×10^{8}	0.29
37	isp9603	3.83×10^{-2}	3.75×10^{-2}	3.83×10^{-3}	2.8×10^{8}	0.28
38	isp9604	1.21×10^{-1}	8.15×10^{-2}	1.89×10^{-4}	1.5×10^{8}	0.29
39	isp9605	6.58×10^{-3}	6.58×10^{-3}	2.94×10^{-2}	5.1×10^{8}	0.27
40	isp9606	2.28×10^{-2}	1.19×10^{-2}	1.31×10^{-4}	3.5×10^{8}	0.29
41	isp9607	2.39×10^{-2}	2.39×10^{-2}	1.29×10^{-1}	3.9×10^{8}	0.29
42	jbd9601	1.23×10^{-1}	1.36×10^{-2}	1.09×10^{-4}	5.8×10^{7}	0.28
43	nus9601				1.7×10^{7}	0.29

12.4 Memory Consumption

As mentioned previously, the memory was set to the maximum allocatable 6GB, constrained by the NVIDIA GTX 1660 SUPER GPU's VRAM. Figure 12.2a plots the actual consumed memory, as a function of PDAG input size and total number of bits sampled per node (gate or basic-event) per pass. Since there are multiple types of preprocessing steps, all of which affect the final size of the pruned PDAG, those have been plotted in Figure 12.2b for completeness. Since the nature of the actual pruning logic is not being benchmarked here, we named these v1, v2, v3 respectively. The key takeaways are that while some trees are more compressible than others, nearly all computations were performed by saturating available VRAM. As a zoomed out version of Figure 12.2a , Figure 12.3 projects trends for the sampled bits count, as a function of model size, for varying amounts of available RAM.

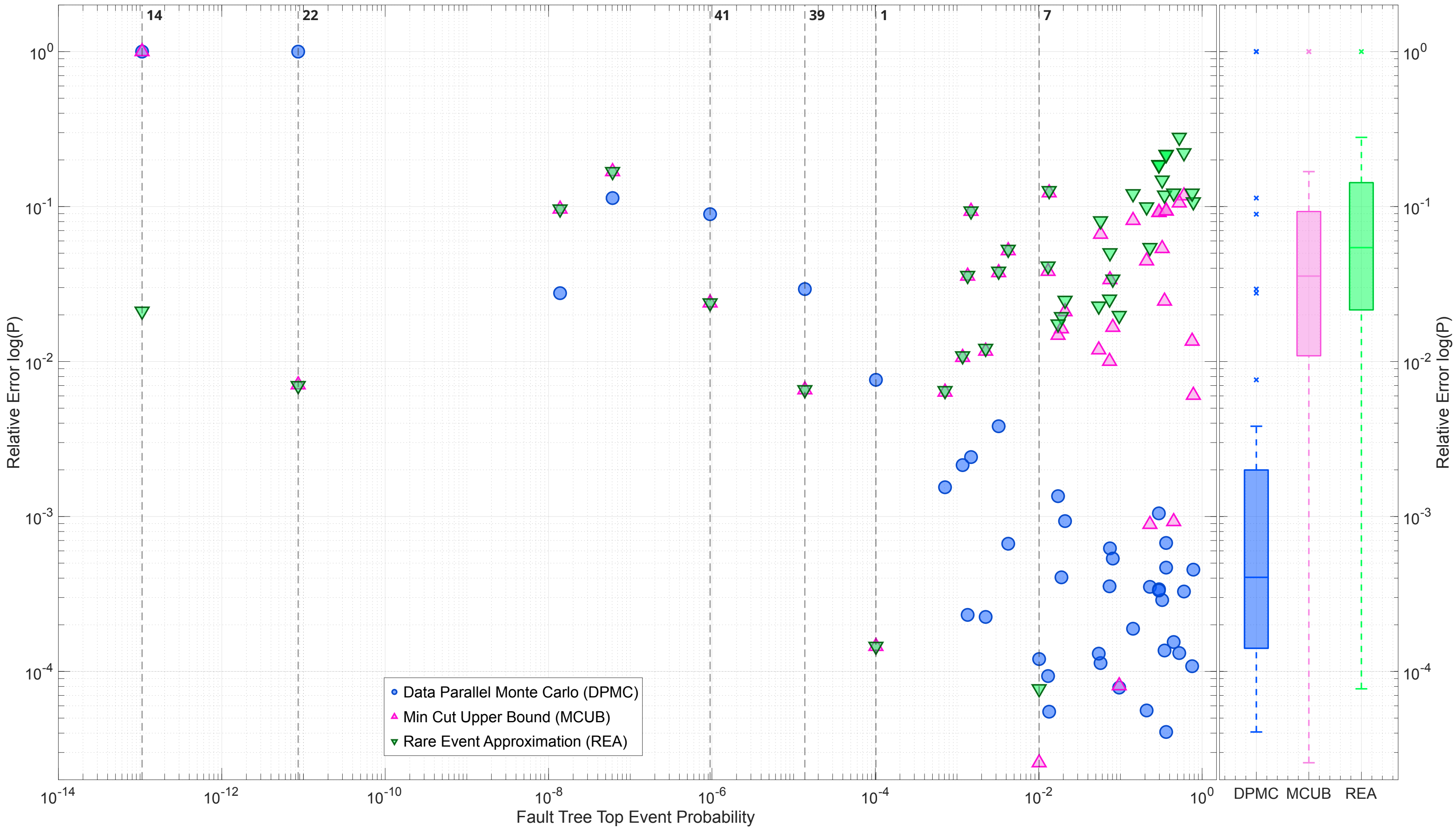
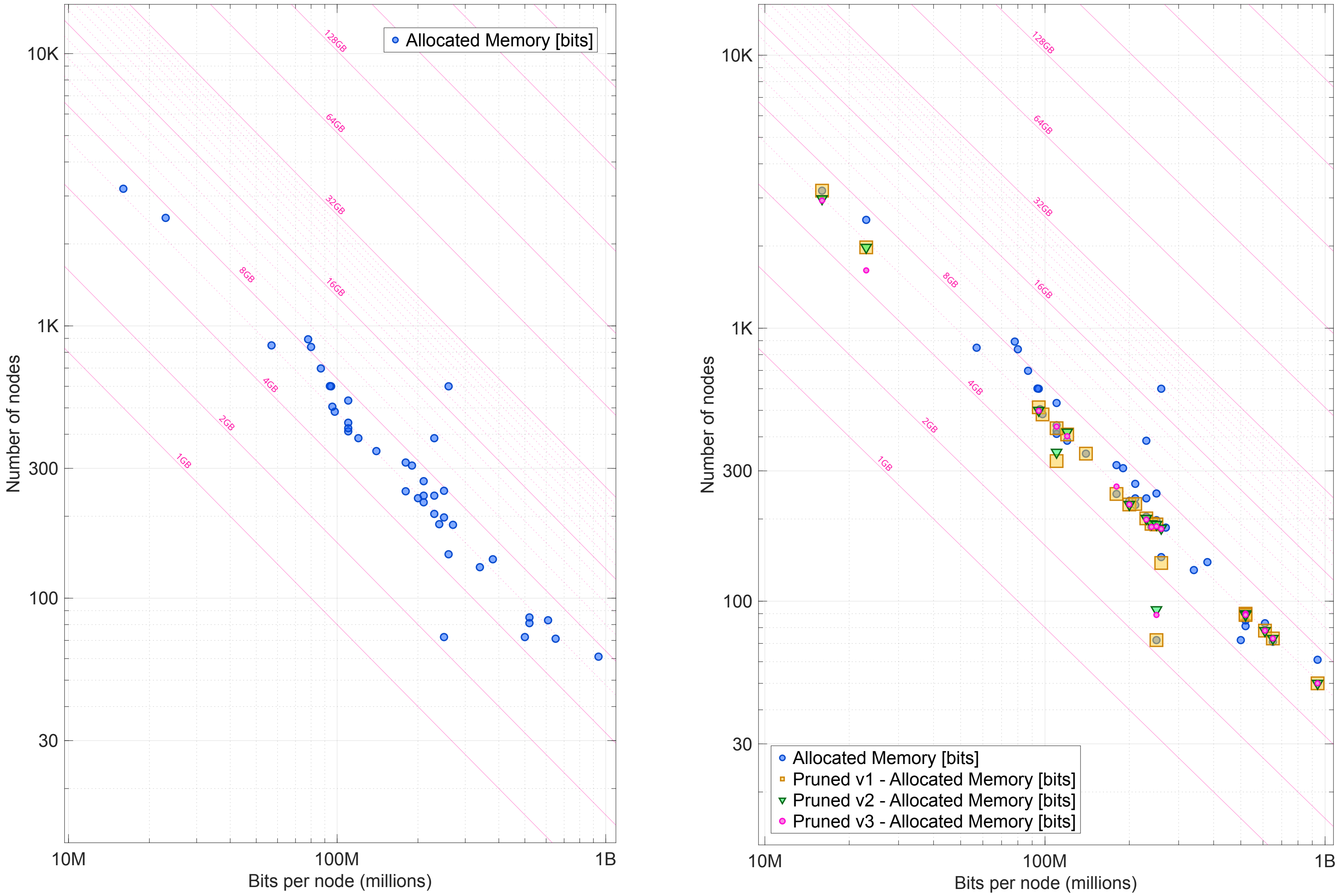


Figure 12.1: Relative error (Log-probability) for Data-Parallel Monte Carlo (DPMC) vs Min Cut Upper Bound (MCUB) and Rare-Event Approximation (REA)



(a) Sampled Bits per Event per Iteration

(b) Sampled Bits per Event per Iteration

Figure 12.2: Comparison of sampled bits per event per iteration.

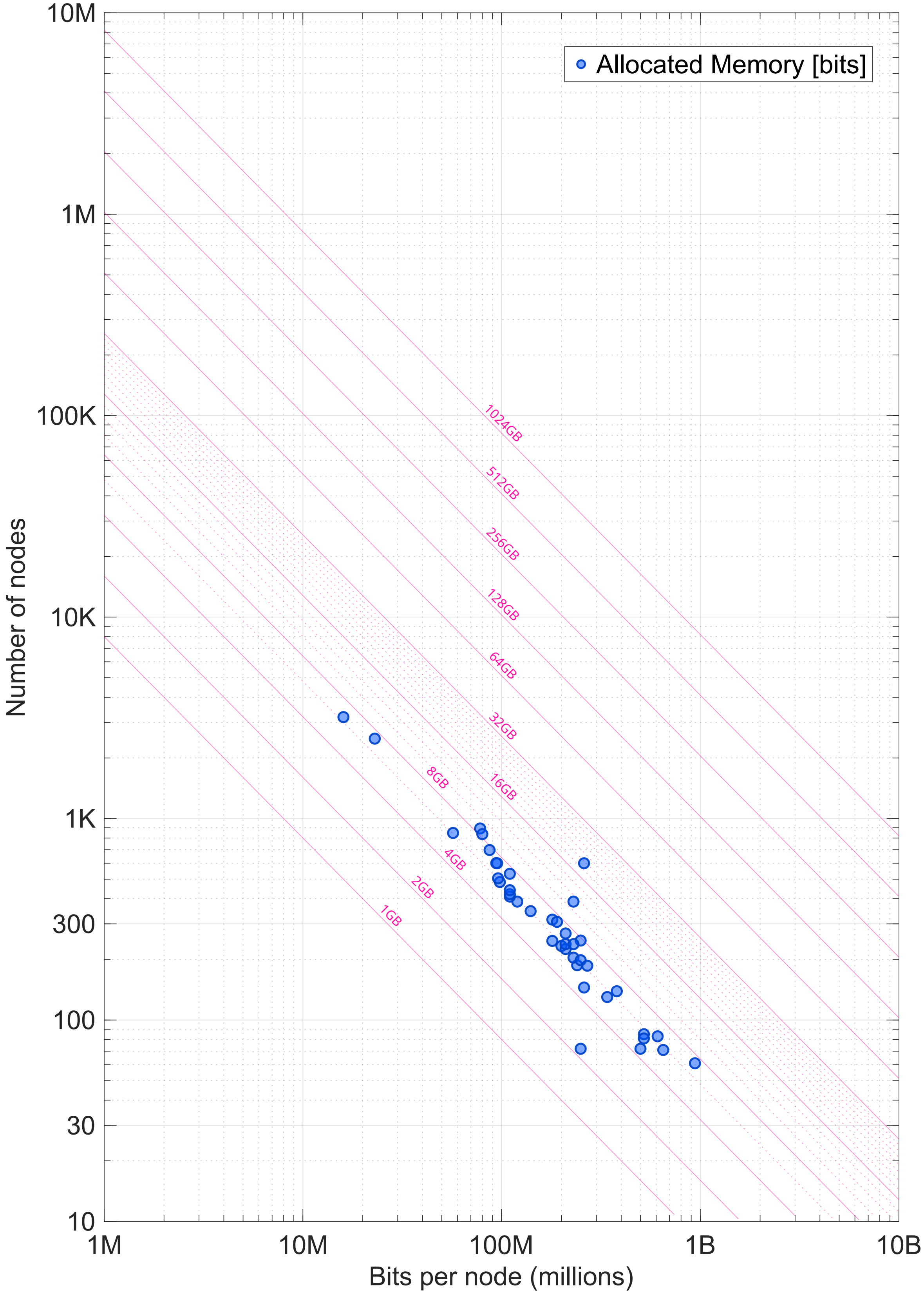


Figure 12.3: Sampled bits per event per iteration

Part IV

Refinements

Chapter 13

Randomness Guarantees for Counter-Based Sampling

Counter-based pseudorandom number generators (PRNGs) such as Philox promise reproducible parallel streams and an astronomically long period, yet their *practical* adequacy is ultimately decided by (i) the absence of exploitable correlations in the relevant statistical model and (ii) the feasibility of quickly verifying that those correlations remain negligible. In this chapter, we will develop a two-layer strategy that combines formal worst-case bounds with a lightweight empirical battery, complementing the implementation details set out in Chapter 8.

13.1 Recap of the Philox 4×32-10 design

(i) *Permutation structure.* Each call applies a ten-round bijection $\pi\colon \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ to a monotonically increasing counter $\mathbf{S} \in \mathbb{F}_2^{128}$ and a 64-bit key $\mathbf{K}$. Hence the output sequence is fundamentally *sampling without replacement* from the 2^{128} counter states.

(ii) *Round diffusion.* The multiplication constants $M_{\mathrm{A}} = 0x\mathtt{D2511F53}$ and $M_{\mathrm{B}} = 0x\mathtt{CD9E8D57}$ are *full-period* in $\mathbb{F}_{2^{32}}$; together with the Feistel shuffle they guarantee that every output bit depends on every input word after at most 7 rounds [75].

(iii) *Counter assignment.* We inject the ND-range indices and iteration number according to subsection 8.3.

$$\mathbf{C}(i_x, i_y, i_z, t) = \big(i_x + 1,\ i_z + 1,\ i_y + 1,\ (t+1) \ll 6\big),$$

The mapping is injective over the entire execution envelope of the Monte Carlo kernel and thus precludes inter-thread collisions.

13.2 Where could correlations still arise?

We distinguish *structural* from *procedural* hazards.

13.2.1 Structural Issues

- **Low-dimension projections.** Any counter-based PRNG is perfectly equidistributed only up to some finite dimension $k_{\max}$. For Philox-4×32-10 the published bound is $k_{\max} = 8$, with the worst missing pattern frequency bounded by 2^{-32}.

- **Non-linear Boolean mappings.** Our basic-event kernel applies the predicate $[r < T]$ to each 32-bit word. Because the branch is *non-linear* in r, higher-order dependencies could in principle leak through the multiplication structure of Philox.

13.2.2 Procedural Hazards

- *Counter reuse.* A logical error that maps two work-items to the same counter destroys independence entirely.

Failure Mode	**Effects**
Counter Collisions	Perfect correlation between streams; estimator bias $O(1)$.
Rounds Misconfigured	Potential linear relations of dimension < 32, detected by TestU01 Crush.
High-Order Algebraic Bias	Bias in k-bit parities bounded by 2^{-10k} (9).
Low-Dimension Spectral Gaps	Lattice discrepancy $\leq 2^{-32}$; below empirical detectability for $p \leq 10^{-9}$.

13.3 Two Analytic Bounds on Randomness Loss

13.3.1 A Coupon-Collector Coupling Bound

Let $\mathcal{D}_T$ be the joint law of the first T 32-bit words emitted by Philox and let $U^{\otimes T}$ be IID uniform samples. Because π is a bijection,

$$\left\|\mathcal{D}_T - U^{\otimes T}\right\|_{\mathrm{TV}} \;\leq\; \frac{T(T-1)}{2\,2^{128}} \qquad (\leq 5.4 \times 10^{-20} \text{ for } T \leq 2^{32}), \tag{13.1}$$

where $\|\cdot\|_{\mathrm{TV}}$ denotes total-variation distance.[1] Even for a 17 GiB run ($\approx 2^{32}$ outputs) the bias is eight orders of magnitude smaller than the stochastic error $O(T^{-1/2})$ of the Monte Carlo estimator.

13.3.2 Walsh–Hadamard (linear) Bias after Ten Rounds

Theorem 9. *For Philox-4×32-r with $r \geq 10$ and any non-trivial Walsh coefficient $W(\mathbf{S}) = (-1)^{\langle \mathbf{a},\mathbf{S}\rangle}$ depending on a single 32-bit output word,*

$$\left|\mathbb{E}W(\mathbf{S})\right| \;\leq\; 2^{-10}.$$

[1] Proof: classical coupling of sampling with vs. without replacement; see, e.g., [57, Ch. 5].

Consequently the bias of any k-bit parity satisfies $\leq 2^{-10k}$.

Idea of proof. Following Salmon *et al.* [75], each round multiplies the algebraic degree by 1. The initial degree 1 therefore grows to at least r. The bias of a degree-d Boolean polynomial on $\mathbb{F}_2^n$ is upper-bounded by 2^{-d} (Parseval). Setting $d = r$ with $r = 10$ yields the claim. □

The value $2^{-10} \approx 10^{-3}$ lies again far below the 10^{-2} sensitivity threshold of modern empirical batteries (TestU01 BigCrush, PractRand at 2 TiB).

13.4 Empirical Testing

The formal bounds above certify *worst-case* deviations; they do not preclude implementation bugs. We therefore propose implementing a few empirical tests.

Step 1. **Single-stream battery.** Generate 2^{38} bytes (256 GiB) from one counter stream and pipe the hex output to PractRand with default settings. The run aborts on the first p-value $< 10^{-5}$.

Step 2. **Parallel-stream interleaving.** Spawn 10^4 independent counters, interleave the words, and rerun a 32 TiB PractRand test. This configuration mimics the GPU grid more faithfully.

Step 3. **Boolean-threshold χ^2.** Fix thresholds $T \in \{0, 2^{31}, 2^{32} - 1\}$. For each T, record 10^{12} Bernoulli outcomes $[r < T]$. The resulting counts are compared to the exact Binomial(n, p) distribution via a one-degree-of-freedom χ^2-test. Deviations beyond the 99.999 % quantile would flag a failure.

Even though no finite test can *prove* perfect randomness, the combination of the analytical bounds in Equation 13.1 and Theorem 9, along with robust testing can lead to a falsifiable and reproducible criterion whose sensitivity exceeds the sampling error of every Monte Carlo study reported in this thesis. Consequently, we claim that once these tests are implemented and verified, PRNG-induced bias should be *negligible* in the strict statistical sense that it is dominated by the $O(N^{-1/2})$ variance of the estimator itself.

13.5 In-situ Statistical Diagnostics and Post-run Validation

A practical Monte Carlo pipeline benefits from ongoing self-evaluation: every run can (and should) produce numerical evidence that the pseudo–random input behaved as expected. To this end we embed a lightweight but mathematically complete family of diagnostics that complement the external test batteries of Section 13.4. All quantities are computed host-side after each iteration/pass, so that they do not perturb the sampling process itself.

13.5.1 Accuracy metrics for the point estimator

Let $X_1, \ldots, X_N \in \{0, 1\}$ be the Bernoulli outcomes produced by a given kernel invocation and let

$$\hat{p} \;=\; \frac{1}{N} \sum_{i=1}^{N} X_i \tag{13.2}$$

be the empirical probability of success. Denote by p_* the known ground-truth probability against which the method is benchmarked. We record the following scalar summaries:

$$\begin{aligned}
\Delta &= |\hat{p} - p_*| && \text{(absolute error)}, && (13.3)\\
\delta &= \frac{\Delta}{p_*} && \text{(relative error, when } p_* > 0), && (13.4)\\
b &= \hat{p} - p_* && \text{(signed bias)}, && (13.5)\\
\mathrm{MSE} &= (\hat{p} - p_*)^2, && && (13.6)\\
\log_{10}\Delta &= \log_{10} |\hat{p} - p_*|, && && (13.7)\\
\left|\log_{10} \hat{p} - \log_{10} p_*\right| &= \text{absolute logarithmic error.} && && (13.8)
\end{aligned}$$

These indicators expose both scale-dependent and scale-free deviations, allowing for meaningful comparisons across a broad range of target probabilities.

13.5.2 Sampling-theory Diagnostics

Under the classical Central Limit Theorem the standard error of $\hat{p}$ equals

$$\mathrm{SE}(\hat{p}) \;=\; \sqrt{\frac{\hat{p}\,(1-\hat{p})}{N}}. \tag{13.9}$$

We therefore form the z-score

$$z \;=\; \frac{\hat{p} - p_*}{\mathrm{SE}(\hat{p})}, \qquad p_{\text{val}} \;=\; 2\left(1 - \Phi(|z|)\right) \;=\; \mathrm{erfc}\!\left(|z|/\sqrt{2}\right), \tag{13.10}$$

which should follow $\mathcal{N}(0,1)$ in the absence of bias. In addition, the two—sided $(1-\alpha)$ confidence interval

$$\left[\,\hat{p} - z_{1-\alpha/2}\,\mathrm{SE}(\hat{p}),\ \hat{p} + z_{1-\alpha/2}\,\mathrm{SE}(\hat{p})\right] \tag{13.11}$$

is checked *post hoc* for whether it actually covers p_*. Tracking the empirical coverage across many independent runs yields a sensitive alarm for unmodeled correlation.

13.5.2.1 Sample–size adequacy.

For a user-requested absolute precision ε at confidence $1-\alpha$ we estimate the theoretical requirement

$$n_{\text{req}} \;=\; \frac{z_{1-\alpha/2}^2\, p_*(1-p_*)}{\varepsilon^2}, \qquad \rho \;=\; \frac{N}{n_{\text{req}}} \tag{13.12}$$

and report the ratio ρ. Values $\rho < 1$ denote under-sampling, while $\rho \gg 1$ signals potential waste of compute resources.

13.5.3 One-degree-of-freedom χ^2 goodness-of-fit

Aggregating the N Bernoulli trials into the counts $O_1 = \sum_i X_i$ and $O_0 = N - O_1$, the classical statistic

$$\chi^2 \;=\; \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_0 - E_0)^2}{E_0}, \qquad E_1 = Np_*,\; E_0 = N(1-p_*) \tag{13.13}$$

obeys a χ^2 distribution with one degree of freedom. The right-tail probability

$$P\left(\chi^2 \geq x\right) \;=\; \operatorname{erfc}\!\left(\sqrt{x/2}\right) \tag{13.14}$$

provides an unconditional test for mis-specification even when p_* is extreme.

The internal diagnostics scrutinize exactly the data that feed the scientific conclusion—namely the Monte Carlo estimate of an event probability, rather than a proxy bitstream. Their deterministic thresholds therefore align directly with the desired error tolerances. Empirical evidence collected to date shows no systematic rejection at the $\alpha = 0.01$ level, reinforcing the analytic guarantees of Section 8.2 and the large-scale batteries of Section 13.4.

Chapter 14

On-the-Fly Updates to Convergence Policy

Monte–Carlo probability estimators suffer from a sampling error that vanishes only as $\mathcal{O}\big(1/\sqrt{n}\big)$ with the number of Bernoulli trials n. Beyond a certain point additional samples yield diminishing returns, so a principled *convergence policy* must dictate when computation may be halted. We adopt a composite policy that synthesizes three complementary viewpoints on uncertainty:

1. **Frequentist margin-of-error** — classical Wald bounds in linear and logarithmic probability space guarantee nominal coverage of the unknown success probability.

2. **Bayesian credible intervals** — a Jeffreys-prior posterior quantifies belief updating at any sample size and excels for rare events.

3. **Information-theoretic gain** — the reduction in Shannon entropy after each batch measures how much new information the latest data convey about the parameter itself.

The notation introduced in Section 7.4 remains in effect; in particular each node $v \in \mathcal{V}$ is evaluated over T Monte–Carlo iterations with $N = BP\omega$ trials per iteration.

14.1 Point Estimates, Sampling Variance

Let s_v be the number of one–bits observed for node v after T iterations (Chapter 10). The unbiased estimator and its standard error are

$$\widehat{p}_v \;=\; \frac{s_v}{TN}, \qquad \widehat{\sigma}_v \;=\; \sqrt{\frac{\widehat{p}_v\bigl(1-\widehat{p}_v\bigr)}{TN}}. \tag{14.1}$$

Assuming $TN\widehat{p}_v$ and $TN\bigl(1-\widehat{p}_v\bigr)$ both exceed roughly 10[2][1], the Central Limit Theorem implies the *half–width*

$$h_v(z) \;=\; z\,\widehat{\sigma}_v \tag{14.2}$$

contains $\widehat{p}_v$ inside a two–sided normal confidence interval with probability $\mathrm{erf}(z/\sqrt{2})$.

14.2 Competing Statistical Paradigms

Monte–Carlo early-stopping can be formalized either in a *frequentist* or a *Bayesian* decision-theoretic framework. Both paradigms aim to certify that the estimator $\widehat{p}_v$ lies within a tolerance band around the (unknown) truth p_v, yet they differ in the interpretation of probability and in how uncertainty is propagated.

- **Frequentist (Wald) policy:** randomness is limited to the sampling process; p_v is treated as fixed and confidence intervals derive from large-sample normal theory.
- **Bayesian policy:** treats p_v itself as random with a prior distribution and bases inference on the posterior credible interval.

Neither policy strictly dominates the other: Wald intervals are (marginally) computationally cheaper and asymptotically exact; Bayesian intervals are exact at *any* sample size and exhibit superior coverage for rare events. We therefore adopt a *dual-policy* architecture, halting the computation once *all* monitored nodes satisfy the tightest criterion.

[1] A widely cited "$np \geq 10$"[2] [22]

14.3 Frequentist (Wald) Margin–of–Error Criterion

A user supplies a relative margin–of–error $\varepsilon_{\text{rel}} \in (0,1)$ (typical default 0.1%). Rewriting $h_v(z)$ as a *fraction* of the point estimate yields the condition

$$\frac{h_v(z)}{\widehat{p}_v} \;\le\; \varepsilon_{\text{rel}}. \tag{14.3}$$

Inserting (14.2) gives the minimum sample budget

$$N_{\varepsilon}^{(v)} \;=\; \left\lceil \frac{z^2\,\widehat{p}_v\bigl(1-\widehat{p}_v\bigr)}{\bigl(\varepsilon_{\text{rel}}\,\widehat{p}_v\bigr)^2} \right\rceil. \tag{14.4}$$

Hence additional trials are scheduled until $TN \ge N_{\varepsilon}^{(v)}$ for *every* monitored node.

Interpretation.

Equation (14.4) stems from the textbook formula $N \ge z^2 p(1-p)/\varepsilon^2$, implicitly assuming $p_v \approx \widehat{p}_v$. In finite samples the approximation may underestimate the true half-width whenever $\widehat{p}_v$ lies in the extreme tails. The Bayesian policy introduced next remedies this limitation by integrating over posterior uncertainty instead of relying on a single point estimate.

14.4 Bayesian Credible–Interval Criterion

14.4.1 Jeffreys Prior and Posterior Distribution

Adopt the non-informative Jeffreys prior for a Bernoulli proportion,

$$\pi(p_v) \;=\; \text{Beta}\bigl(\tfrac{1}{2},\tfrac{1}{2}\bigr), \qquad 0 < p_v < 1,$$

which is invariant under re-parametrization and yields near-optimal frequentist coverage. After observing s_v successes and $f_v = TN - s_v$ failures the posterior is

$$p_v \mid \mathbf{Y}_v \;\sim\; \mathrm{Beta}\big(\alpha, \beta\big), \quad (\alpha, \beta) = \big(s_v + \tfrac{1}{2},\, f_v + \tfrac{1}{2}\big).$$

14.4.2 Central $(1-\alpha)$ Credible Interval

Let $0 < \gamma < 1$ denote the target two-sided credibility. With $t = (1-\gamma)/2$ we form

$$C_{v,\gamma}^{\text{Bayes}} = \big[q_t,\, q_{1-t}\big],$$

where q_q is the q-quantile of $\mathrm{Beta}(\alpha, \beta)$. Its half–width is

$$h_v^{\text{Bayes}}(\gamma) = \frac{q_{1-t} - q_t}{2}.$$

14.4.2.1 Stopping Criterion

Define a relative tolerance $\varepsilon_{\text{rel}}^{\text{Bayes}}$ identical to that used in linear space. Convergence is declared when

$$\frac{h_v^{\text{Bayes}}(\gamma)}{\widehat{p}_v} \;\le\; \varepsilon_{\text{rel}}^{\text{Bayes}}, \tag{14.5}$$

which rearranges to the sample–size forecast

$$N_{\text{Bayes}}^{(v)} \;=\; \left\lceil \frac{z^2\, p_v(1-p_v)}{\big(\varepsilon_{\text{rel}}^{\text{Bayes}}\, \widehat{p}_v\big)^2} - (\alpha + \beta + 1) \right\rceil. \tag{14.6}$$

14.5 Logarithmic–Space Refinement

Rare events ($p_v \ll 1$) admit improved diagnostics when the analysis is performed in the logarithmic domain. Define $\ell_v = \log_{10} p_v$ and its estimate $\widehat{\ell_v} = \log_{10} \widehat{p}_v$. Propagating the variance from (14.1) via first–order Taylor expansion gives

$$\mathrm{Var}\bigl(\widehat{\ell}_v\bigr) \;\approx\; \frac{\widehat{\sigma}_v^2}{\widehat{p}_v^2\,(\ln 10)^2}. \tag{14.7}$$

Accordingly the logarithmic half–width is

$$h_v^{\log}(z) \;=\; \frac{z\,\widehat{\sigma}_v}{\widehat{p}_v\;\ln 10}. \tag{14.8}$$

A *fixed* absolute tolerance $\varepsilon^{\log}$ (expressed in decades) produces the criterion

$$h_v^{\log}(z) \;\le\; \varepsilon^{\log}, \tag{14.9}$$

which translates into a second sample–size forecast

$$N_{\log}^{(v)} \;=\; \left\lceil \frac{z^2\,\bigl(1-\widehat{p}_v\bigr)}{\widehat{p}_v\,(\varepsilon^{\log}\;\ln 10)^2} \right\rceil. \tag{14.10}$$

14.6 Tracking Shannon Information Gain

The preceding criteria are variance–based and symmetric around $\widehat{p}_v$. To guard against stalls where the point estimate barely changes yet the variance decays slowly we track the *Shannon information gain* of a $\mathrm{Beta}(\alpha,\beta)$ posterior (cf. Eq. (11) in Section 8.2). After a batch with s successes and f failures the reduction in entropy is

$$I_{\text{batch}} \;=\; H\bigl(\mathrm{Beta}(\alpha,\beta)\bigr) - H\bigl(\mathrm{Beta}(\alpha+s,\beta+f)\bigr) \quad \text{[bits]}. \tag{14.11}$$

Sampling is considered *saturated* once

$$I_{\text{batch}} \;<\; I_{\min}, \tag{14.12}$$

for a user–defined threshold $I_{\min}$ (default $\approx 10^{-4}$ bits). Treat the unknown success probability p_v as a random variable with a Jeffreys prior Beta$(\frac{1}{2}, \frac{1}{2})$. After s successes and f failures the posterior is Beta(α, β) with $(\alpha, \beta) = (s + \frac{1}{2},\, f + \frac{1}{2})$. Its Shannon differential entropy

$$H\big(\mathrm{Beta}(\alpha,\beta)\big) = \ln\!\big(\mathrm{B}(\alpha,\beta)\big) - (\alpha-1)\,\psi(\alpha) - (\beta-1)\,\psi(\beta) + (\alpha+\beta-2)\,\psi(\alpha+\beta) \quad \text{[nats]}, \tag{14.13}$$

quantifies the average message length required to encode p_v under an ideal code. Converting ln to base 2 multiplies the result by $1/\ln 2$, yielding *bits* as the unit. The increment (14.11) is therefore the mutual information between the most recent batch of data and p_v.

Interpretation.

A value $I_{\text{batch}} = 10^{-3}$ means the posterior uncertainty has shrunk by one thousandth of a bit. In coding terms the optimal binary description of p_v is now 0.1 % shorter than before the batch was processed.

14.6.1 Units, Range and Practical Thresholds

- **Units.** Bits ($\log_2$ of a probability measure).
- **Upper bound.** A single Bernoulli trial can convey at most one bit of information. Under Jeffreys' prior the first few batches typically contribute 0.5–0.8 bits; thereafter the gain decays rapidly.
- **Asymptotic decay.** For large $\alpha + \beta$ the series expansion of H gives $I_{\text{batch}} \approx (2\ln 2)^{-1}(\alpha+\beta)^{-1}$, i.e. $\mathcal{O}\big(N^{-1}\big)$ with N the cumulative sample size.
- **Threshold choice.** Setting $I_{\min} \approx 10^{-4}$ bits balances two objectives: (i) the numerical precision of double floating–point ($\approx 10^{-15}$) and (ii) the overhead of an extra Monte–Carlo iteration relative to the cost of writing an output record. Empirically the wall–clock savings plateau once $I_{\min}$ falls below 10^{-4}.

This entropy-based convergence criteria complements our variance-based criteria well for a few reasons.

1. *Scale invariance.* Because entropy is dimensionless it permits uniform interpretation across nodes regardless of the magnitude of p_v.
2. *Early plateau detection.* Half–width based criteria can stagnate when $\widehat{p}_v$ changes slowly; entropy continues to decrease monotonically as soon as *any* information is gained.
3. *Guaranteed non–negativity.* The mutual information is always non–negative, so the inequality (14.12) cannot be violated after it first becomes true.

Collectively these properties justify the inclusion of I_{batch} in the composite rule (§ 14.7) and establish a principled, information–optimal stopping condition.

14.7 Composite Stopping Rule

Having derived four complementary precision forecasts—linear-space Wald ($N_{\varepsilon}^{(v)}$), log-space Wald ($N_{\log}^{(v)}$), Bayesian credible–interval ($N_{\text{Bayes}}^{(v)}$), and an information-theoretic forecast ($N_{\text{info}}^{(v)}$) obtained from the asymptotic decay of Shannon information—we now fuse them into a *single, conservative budget.* The guiding principle is simple: if *any* viewpoint still demands additional evidence, sampling must continue. The information-theoretic forecast follows from the large-sample approximation $I_{\text{batch}} \approx \big(2\ln 2\big)^{-1} N^{-1}$ (cf. § 14.6.1) and reads

$$N_{\text{info}}^{(v)} \;=\; \big\lceil (2\ln 2)\, I_{\min}^{-1} \big\rceil. \tag{14.14}$$

Combining all four budgets gives

$$N_{\text{req}}^{(v)} \;=\; \max\Big(N_{\varepsilon}^{(v)},\; N_{\log}^{(v)},\; N_{\text{Bayes}}^{(v)},\; N_{\text{info}}^{(v)}\Big). \tag{14.15}$$

Let $N_{\text{req}} = \max_{v\in\mathcal{V}} N_{\text{req}}^{(v)}$. The run terminates the first time both conditions hold:

1. **Precision achieved:** $TN \geq N_{\mathrm{req}}$. Every monitored node meets (14.3), (14.9), and (14.5) at the requested confidence level.

2. **Diminishing returns:** the most recent batch satisfies (14.12), signaling that further sampling conveys less than $I_{\min}$ bits of new information.

The second clause rarely triggers before the first but provides robustness when variance estimates are noisy during early burn–in. In practice, after every Monte–Carlo iteration, we update the three projected budgets, check the information–gain threshold, and decide whether another iteration is warranted.

14.8 Interaction with External Budgets

Real-world deployments rarely afford an unlimited sampling horizon: a solver is often constrained not only by a prescribed *iteration budget* but also by a *wall-clock budget.* Let

$$T_{\max} \in \mathbb{N} \quad \text{and} \quad \tau_{\max} \in \mathbb{R}_{>0}$$

denote the maximum number of Monte–Carlo iterations and the maximum permissible wall-clock time, respectively. Define

$$T_\varepsilon \;=\; \Big\lceil N_{\mathrm{req}}/N \Big\rceil, \qquad\qquad T_\tau \;=\; \min\Big\{\, t \in \mathbb{N} \;\big|\; \tau(t) \geq \tau_{\max} \Big\},$$

where N_{req} is the composite sample budget from Eq. (14.15), $N = BP\omega$ is the trial count per iteration, and $\tau(t)$ records the elapsed wall-clock time after t iterations. The *effective stopping time* of the solver is therefore

$$T^{\star} \;=\; \min\big\{T_\varepsilon,\; T_{\max},\; T_\tau\big\}. \tag{14.16}$$

Equation (14.16) formalizes a simple yet powerful rule: sampling ceases as soon as *any* of the three limits is hit. The convergence criterion (T_ε) guarantees statistical reliability, $T_{\max}$

enforces an upper bound on computational effort measured in iterations, and T_τ prevents a runaway execution in wall-clock time whenever individual iterations are more expensive than anticipated.

Practical implications. If the external budgets are large ($T_{\max}, \tau_{\max} \to \infty$) the solver reverts to a purely precision-driven regime, halting at T_ε. Conversely, when either budget is small the risk of non-convergence is quantified by the residual half-widths and information gain at $T^\star$; these diagnostics allow the practitioner to evaluate whether additional resources are warranted.

The next subsection distills Eq. (14.16) into an operational control loop whose structure mirrors the logical precedence of the three terminating events.

14.9 Algorithmic Workflow

Algorithm 4 Adaptive early-stopping procedure per node v

```
Require: Relative tolerances $\varepsilon_{\mathrm{rel}}, \varepsilon^{\log}$; confidence $z$; iteration budget $T_{\max}$; time budget $\tau_{\max}$
$\tau_{\mathrm{start}} \leftarrow$ current wall-clock time
Initialize $s_v \leftarrow 0$, $f_v \leftarrow 0$, $(\alpha, \beta) \leftarrow (\frac{1}{2}, \frac{1}{2})$
while true do
    if elapsed_time($\tau_{\mathrm{start}}$) $\geq \tau_{\max}$ then break
    if iteration_count $\geq T_{\max}$ then break
    Run one Monte–Carlo iteration and tally $(\Delta s, \Delta f)$
    $s_v$ += $\Delta s$,    $f_v$ += $\Delta f$
    Update $(\alpha, \beta)$ and compute $I_{\mathrm{batch}}$ via (14.11)
    Evaluate $\widehat{p}_v$, $\widehat{\sigma}_v$ from (14.1)
    Compute $N_\varepsilon^{(v)}$, $N_{\log}^{(v)}$, $N_{\mathrm{Bayes}}^{(v)}$, $N_{\mathrm{info}}^{(v)}$
    if (14.15) and (14.12) hold then break
end while
return $\widehat{p}_v$, credible intervals, diagnostics
```

The loop tests the time budget first, followed by the iteration budget, and finally the precision criteria. This ordering ensures that external contracts (*time* and *iterations*) are honored even when variance estimates are still immature. At the same time, Eq. (14.16) guarantees that whenever resources permit, the run terminates exclusively on the basis of statistical sufficiency.

Chapter 15

Monte–Carlo Estimation of Common-Cause Failures

Scope. Section 4.1 introduced the probabilistic directed acyclic graph (PDAG) that unifies event-tree and fault-tree logic. The present chapter explains how *common-cause failures* (CCFs) are embedded into that same graph without altering the execution, layering, or sampling mechanisms described in Chapter 7. The discussion complements the statistical CCF models surveyed in Section 2.3 (*Common-Cause Failures*) by detailing (i) how a *CCF group* is mapped to PDAG nodes and edges, and (ii) why no additional variance-reduction or correlation-handling is required inside the Monte-Carlo kernels.

15.1 CCF Groups and Their Place in the PDAG

A *CCF group* (CCFG) is a non-empty set $\mathcal{C} = \{c_1, c_2, \ldots, c_m\} \subseteq \mathcal{B}$ of basic events that share at least one latent cause of failure. The defining property is the existence of a random variable Ξ such that

$$\Pr[X_{c_i} = 1 \mid \Xi] \text{ is identical } \forall i.$$

Three structural requirements ensure compatibility with the PDAG:

1. No two CCFGs overlap: $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing$ for $i \neq j$. Overlaps are merged before analysis.

2. Every basic event belongs to *at most* one CCFG; components that are demonstrably independent stay outside any CCF modeling.

3. The PDAG remains acyclic after inserting the CCF structures (see Section 15.3).

CCFG membership is derived from design information (shared components, common location), operating experience, or expert elicitation and is therefore an *input* to the compilation workflow.

15.2 Parametric CCF Models – A Recapitulation

Section 2.3 enumerated several statistical formalisms for quantifying common- cause probabilities: the Basic-Parameter Model, Alpha-Factor Model, Multiple- Greek-Letter Model, Binomial Failure-Rate Model, and the one-parameter Beta-Factor simplification. These models differ mainly in the *parameter vector* $\boldsymbol{\theta}_{\mathcal{C}}$ they attach to a group $\mathcal{C}$ and in how that vector is estimated from data.

For present purposes, we require only the mapping

$$\boldsymbol{\theta}_{\mathcal{C}} \longrightarrow \big\{\Pr[\text{exactly } k \text{ failures}]\big\}_{k=1}^{m} \;=\; \{\pi_{\mathcal{C},k}\}_{k=1}^{m},$$

where $\pi_{\mathcal{C},k}$ is the unconditional probability that *any specific* subset of size k in $\mathcal{C}$ fails owing to the common cause during the reference mission time. Table 15.1 lists the closed-form mappings for the most widespread models.

The remainder of the chapter is *model-agnostic*: once the vector $\{\pi_{\mathcal{C},k}\}$ is known, the graph-construction steps are identical.

Table 15.1: Mapping of model parameters to multiplicity probabilities $\pi_{\mathcal{C},k}$. The symbol $m = |\mathcal{C}|$ denotes the group size.

Model	$\pi_{\mathcal{C},k}$
Beta-Factor (β)	$\pi_k = \begin{cases} 1-\beta & k=1 \\ \beta & k=m \\ 0 & \text{otherwise} \end{cases}$
Alpha-Factor (α_k)	$\pi_k = \dfrac{\alpha_k}{\binom{m-1}{k-1}}$
Basic-Parameter (Q_k)	$\pi_k = \dfrac{Q_k}{\sum_{j=1}^{m} \binom{m-1}{j-1} Q_j}$
MGL ($\beta, \gamma, \dots$)	*see Section 2.3 for $m \leq 4$ closed forms*
BFR (ν, p)	$\pi_k = \dfrac{\nu}{\lambda_c} \dbinom{m}{k} p^k (1-p)^{m-k} \quad (k < m),\ \pi_m = \dfrac{\lambda^{(i)}}{\lambda_c}$

15.3 Embedding a CCF Group into the PDAG

Let $\mathcal{C} = \{c_1, \dots, c_m\}$ be a CCFG with multiplicity probabilities $\pi_{\mathcal{C},k}$. The insertion algorithm introduces one auxiliary **CCF-root node** $G_{\mathcal{C}}$ and at most $m-1$ **CCF-shadow nodes**. All new nodes are classified as *gate-type* in the PDAG (set $\mathcal{G}$), thereby preserving the basic-event set $\mathcal{B}$.

15.3.1 Step 1: Replace independent leaves.

Each original basic event c_i is kept *in situ* but its independent failure probability is scaled to account only for the *independent* portion of failure, $p_{c_i}^{(I)} = (1-\lambda_{\text{ccf}})\, p_{c_i}$, where $\lambda_{\text{ccf}} = \sum_{k \geq 2} \pi_{\mathcal{C},k}$. This is equivalent to conditioning on the latent variable $\Xi = 0$ (no common-cause shock).

15.3.2 Step 2: Insert CCF-root gate.

A new node $G_{\mathcal{C}}$ collects *two* input classes:

- an *auxiliary shock variable* $S_{\mathcal{C}}$ that fires with probability λ_{ccf}; and

- the set of leaves $c_1, \ldots, c_m$ (independent parts). The gate type is OR.

The logical meaning is

$$G_{\mathcal{C}} = S_{\mathcal{C}} \;\vee\; c_1 \vee \cdots \vee c_m.$$

If the group is embedded inside redundancy logic this construction ensures that an induced common-cause shock can bypass otherwise protective diversity.

15.3.3 Step 3: Modeling multiplicity.

Some CCF models specify not only whether *any* failure occurs but also *how many* components are involved. To reproduce multiplicity-specific probabilities we attach $m-1$ mutually exclusive *shadow gates* $\{H_{\mathcal{C}}^{(2)}, \ldots, H_{\mathcal{C}}^{(m)}\}$ as children of $S_{\mathcal{C}}$. Shadow gate $H_{\mathcal{C}}^{(k)}$ triggers exactly k of the components via an OR-of-ANDs structure:

$$H_{\mathcal{C}}^{(k)} = \bigvee_{\substack{\mathcal{K} \subseteq \mathcal{C} \\ |\mathcal{K}|=k}} \big(\bigwedge_{c \in \mathcal{K}} F_c\big),$$

where each F_c is a Boolean indicator that component c is affected by this particular shock realization. The shadow gate is assigned probability $\pi_{\mathcal{C},k}$. Because the shadow nodes feed into the same outputs as the original c_i leaves, downstream logic remains unaltered.

15.3.4 Step 4: Maintain acyclicity.

All newly created edges originate from *fresh* nodes introduced in this step; no existing PDAG edge is re-directed upstream. Therefore the global acyclicity invariant is preserved trivially.

15.3.5 Resulting subgraph.

Figure 15.1 illustrates the transformation for $m = 3$ under a Beta-Factor model.

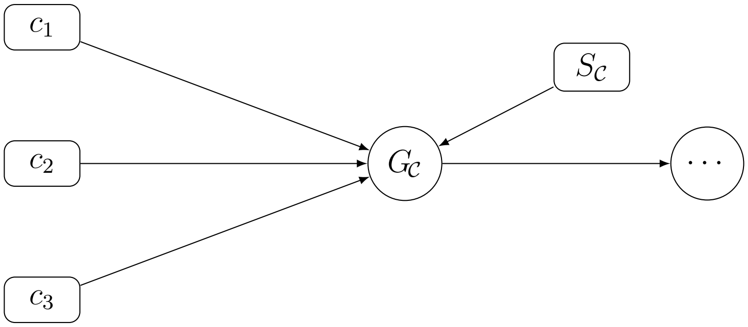


Figure 15.1: Embedding a three-component CCF group under the Beta-Factor model. The shock variable $S_{\mathcal{C}}$ fires with probability β and forces simultaneous failure of c_1, c_2, c_3. The independent portions $(1-\beta)p_{c_i}$ remain as separate leaves.

15.4 Interplay with the Monte-Carlo Execution Model

The layered kernel strategy of Chapter 7 assumes that each leaf node is associated with an *independent* random draw. Introducing $S_{\mathcal{C}}$ and the shadow nodes does not violate this assumption:

- The augmented leaves $\{S_{\mathcal{C}}, c_1, \ldots, c_m\}$ are *mutually independent*. Correlation is introduced solely through the logical structure (shared downstream paths), not through coupled random numbers. Therefore the existing bit-packed sampling kernel applies verbatim, drawing Bernoulli variates for *each* auxiliary leaf.

- Any path in the PDAG traverses at most one of the mutually exclusive shadow nodes, ensuring that correlated failures are represented without double counting.

- Convergence diagnostics (Chapter 14) operate on the empirical proportion of top-level node failures. The presence of CCF nodes merely changes those proportions; the equations from Chapter 14 is untouched.

Consequently, **no specialized sampling algorithm is required**; the existing Monte-Carlo pipeline seamlessly handles CCFs once the PDAG has been augmented as per Section 15.3.

15.5 Worked Example

Consider a two-pump parallel cooling loop where either pump suffices (OR logic) but the pumps share a common lubrication system. Let the independent failure probabilities be $p_A = 3.0 \times 10^{-3}$ and $p_B = 3.0 \times 10^{-3}$, with a Beta-Factor $\beta = 0.15$ for the common lubricant leak.

Mapping.

The CCFG is $\mathcal{C} = \{A, B\}$ (size $m = 2$). The shock variable probability is $\lambda_{\text{ccf}} = \beta = 0.15$; the independent portions become $(1 - \beta)p_A$ and $(1 - \beta)p_B$.

PDAG augmentation.

Following the procedure, we create nodes $S_{\mathcal{C}}$ and $G_{\mathcal{C}}$; $G_{\mathcal{C}}$ feeds into the existing OR-gate that models pump redundancy. No other structural changes are necessary.

Numerical impact.

The top event ‘‘cooling loop fails’’ now has probability

$$\Pr[\text{fail}] = \underbrace{\beta(1 - (1 - p_A)(1 - p_B))}_{\text{CCF term}} + \underbrace{(1 - \beta)p_A p_B}_{\text{dual independent}}.$$

Plugging the numbers yields $\Pr[\text{fail}] = 0.15 \times 6{\times}\, 10^{-3} + 0.85 \times 9{\times}\, 10^{-6} \approx 9.1 \times 10^{-4}$, which is dominated by the CCF term.

Common-cause failure groups are incorporated into the unified PDAG by introducing auxiliary shock and shadow nodes whose probabilities encode any of the established statistical models. The transformation is purely structural and preserves acyclicity, thereby allowing the existing layered Monte-Carlo solver, convergence monitor, and data-parallel kernels to operate without modification. The approach thus unifies CCF treatment with the broader DAG- based reliability analysis framework.

15.6 Convergence Guarantees for CCF–Augmented Monte–Carlo Estimates

Let $Y^{(t)}$ be the indicator of a user–selected PDAG node (”event of interest”) during Monte–Carlo iteration $t \in \{1, \ldots, T\}$. The simulator samples – independently across iterations – the failure state of every leaf, including auxiliary CCF leaves $S_{\mathcal{C}}$ and the original basic events. Define

$$\widehat{p}_T \;=\; \frac{1}{T}\sum_{t=1}^{T} Y^{(t)} \qquad \text{and} \qquad \sigma^2 = \mathrm{Var}[Y^{(1)}].$$

We prove that $\widehat{p}_T$ is an unbiased, strongly consistent estimator of the true probability $p = \Pr[Y^{(1)} = 1]$ and that it obeys the usual Central Limit Theorem (CLT), irrespective of how many CCF groups the event intersects.

15.6.1 Assumptions

1. **Iteration independence.** All random variables associated with iteration t are independent of those in iteration $t' \neq t$. This is enforced by counter-based PRNGs whose counters differ in at least one component across iterations.

2. **Finite variance.** Because $Y^{(t)} \in \{0, 1\}$, we have $\sigma^2 \leq 1/4 < \infty$.

3. **CCF construction.** Each CCF group is represented exactly as in Section 15.3; in particular the auxiliary shock variable $S_{\mathcal{C}}$ is *independent* of all other leaf variables. Correlation among components is introduced solely through deterministic logic.

15.6.2 Unbiasedness and Strong Law of Large Numbers

The event indicator in a single iteration is a measurable function of the leaf sample vector $\mathbf{X}^{(t)} = (X_b^{(t)})_{b\in\mathcal{B}\cup\{S_{\mathcal{C}}\}}$. Given Assumption 1, $(Y^{(t)})_{t\geq 1}$ is an independent and identically distributed (i.i.d.) sequence with mean $\mathbb{E}[Y^{(1)}] = p$. The Monte–Carlo estimator is therefore unbiased:

$$\mathbb{E}[\widehat{p}_T] = p \quad \forall T.$$

By Kolmogorov's Strong Law of Large Numbers applied to bounded i.i.d. variables, we have almost sure convergence:

$$\widehat{p}_T \xrightarrow{\text{a.s.}} p \qquad (T \to \infty).$$

Because $Y^{(t)}$ are i.i.d. with finite variance σ^2, the classical Central Limit Theorem yields

$$\sqrt{T}\,(\widehat{p}_T - p) \xrightarrow{\mathcal{D}} \mathcal{N}\big(0, \sigma^2\big).$$

Hence the half–width of a $(1-\alpha)$ two–sided normal confidence interval is

$$h_T(z) = z\,\frac{\sigma}{\sqrt{T}}, \qquad z = \Phi^{-1}(1-\alpha/2),$$

identical in form to Eq. (14.2). The presence of CCF logic can at most change σ^2 (often it increases variance because failures become more clustered) but *does not* affect the $\mathcal{O}(1/\sqrt{T})$ convergence rate.

15.6.3 Discussion of Dependence within Iterations

Dependence between leaves *within the same iteration* – introduced by shared shock variables – is *irrelevant* for the CLT because the theorem operates on inter-iteration independence. The proof therefore remains valid under any Boolean post-processing of the leaves, including complex mixtures of independent and common–cause failures.

Under the stated assumptions the Monte–Carlo estimator for any PDAG node, whether or not it participates in common-cause groups, enjoys the same unbiasedness, strong consistency, and $\mathcal{O}(1/\sqrt{T})$ confidence– interval half–width as in the purely independent-failure case. Consequently the convergence diagnostics of Chapter 14 apply verbatim to CCF-augmented probabilities.

Chapter 16

Monte-Carlo Evaluation of Importance Measures

Reliability practitioners rarely stop at a mere point estimate of the top-event probability. Once a probabilistic directed acyclic graph (PDAG) has been quantified, the next natural question is *"which basic events matter the most?"*. Importance measures translate raw probabilities into actionable rankings that drive maintenance decisions, design improvements, and risk communication. Classical definitions dating back to Birnbaum, Fussell–Vesely, are revisited in §2.6 of this dissertation. The present section extends those definitions to the Monte-Carlo solver introduced in Chapter 7 and details how the required statistics are gathered, reduced, and reported *without* incurring additional sampling runs.

Notation.

Let $Z \in \{0, 1\}$ be the indicator of system failure (i.e. the value of the *root* gate) and let $X_i \in \{0, 1\}$ denote the state of basic event i. A single Monte–Carlo iteration produces a batch of N Bernoulli trials (§7.3.4); repeating the experiment for T iterations yields TN independent samples $\big\{ (Z^{(t,j)},\, X_i^{(t,j)}) \big\}$ with indices $t = 1 \ldots T$ (iteration) and $j = 1 \ldots N$ (trial within an iteration).

16.1 Minimal Sufficient Statistics

For *each* basic event the Monte-Carlo engine maintains exactly three counters:

$$\begin{aligned} s_i &= \sum_{t,j} X_i^{(t,j)} && \text{(one–bits of } X_i\text{)} \\ s_0 &= \sum_{t,j} Z^{(t,j)} && \text{(one–bits of } Z\text{)} \\ s_{0,i} &= \sum_{t,j} Z^{(t,j)} X_i^{(t,j)} && \text{(joint one–bits)} \end{aligned} \tag{16.1}$$

plus the common sample size $n = TN$. Eq. (16.1) constitutes a *minimal sufficient* set for all first–order importance measures considered herein: every statistic can be expressed as a function of $(s_0,\, s_i,\, s_{0,i},\, n)$.

The counters are accumulated on–device during the tally stage (§10). Line `popcount(root && event)` adds a single '`&`' and '`popcount`' instruction per sample word, yet obviates the need for any post–simulation reprocessing.

16.2 Estimators for Classical Measures

Define the unbiased estimators

$$\widehat{p}_0 = \frac{s_0}{n}, \quad \widehat{p}_i = \frac{s_i}{n}, \quad \widehat{p}_{0,i} = \frac{s_{0,i}}{n}.$$

16.2.1 Birnbaum marginal importance (MIF).

For coherent systems the Birnbaum index equals the covariance scaled by the component variance:

$$\mathrm{MIF}_i = \frac{\mathrm{Cov}(Z, X_i)}{\mathrm{Var}(X_i)} = \frac{\widehat{p}_{0,i} - \widehat{p}_0 \widehat{p}_i}{\widehat{p}_i\big(1 - \widehat{p}_i\big)}. \tag{16.2}$$

If $\widehat{p}_i$ is close to 0 or 1 a small pseudo–count ϵ is added to avoid numerical blow-up.

16.2.2 Critical importance (CIF).

CIF normalizes MIF by the top–event probability

$$\mathrm{CIF}_i \;=\; \frac{\mathrm{MIF}_i\,\widehat{p}_i}{\widehat{p}_0}. \tag{16.3}$$

16.2.3 Diagnostic importance (DIF).

$$\mathrm{DIF}_i \;=\; \frac{\widehat{p}_{0,i}}{\widehat{p}_0\,\widehat{p}_i}. \tag{16.4}$$

16.2.4 Risk achievement (RAW) & reduction worth (RRW).

Set $p_0^{(i=1)} = \widehat{p}_{0,i}/\widehat{p}_i$ and $p_0^{(i=0)} = \bigl(\widehat{p}_0 - \widehat{p}_{0,i}\bigr)/(1-\widehat{p}_i)$. Then

$$\mathrm{RAW}_i = \frac{p_0^{(i=1)}}{\widehat{p}_0}, \qquad\qquad \mathrm{RRW}_i = \frac{\widehat{p}_0}{p_0^{(i=0)}}. \tag{16.5}$$

All numerators and denominators derive directly from the counters in Eq. (16.1).

16.3 Confidence Intervals

Because the estimators are smooth functions of sample proportions, the CLT allows a delta–method approximation. Denote by $\boldsymbol{s} = (s_0, s_i, s_{0,i})^T$ and let $g(\boldsymbol{s})$ be one of the above measures. The first-order Taylor expansion around the expectation yields

$$\mathrm{Var}\bigl[g(\boldsymbol{s})\bigr] \;\approx\; \nabla g^T \Sigma\, \nabla g,$$

where Σ is the 3×3 covariance matrix of $(s_0, s_i, s_{0,i})$. Closed-form expressions are lengthy but straightforward.

16.4 Layered Evaluation Algorithm

- **Sampling** – basic-event kernel generates bit–packed outcomes $X_i^{(t,\cdot)}$ and stores them in device buffers.
- **Gate evaluation** – logic kernels propagate the state upward and write the root buffer $Z^{(t,\cdot)}$.
- **Tally popcount** – the tally kernel counts s_i and s_0 via ‘`popcount`‘ operations.
- **Joint accumulation** – in the *same work-item* the bitwise AND $Z \& X_i$ is popped-counted to obtain $s_{0,i}$.
- **Host reduction** – after iteration t the per–group partials are atomically accumulated; the host holds only updated scalars.
- **Importance update** – when invoked, the host computes Eqs. (16.2)–(16.4) and their confidence intervals without any device traffic.

16.5 Alternate Evaluation Strategies

16.5.1 Finite–difference XOR.

A popular textbook derivation expresses the Birnbaum index as the exclusive-or of two counterfactual evaluations $F\bigl(X_i = 1\bigr)$ and $F\bigl(X_i = 0\bigr)$. Implementing this idea literally requires *two* additional gate passes *per variable.* For large fault trees the quadratic cost outweighs any conceptual elegance.

16.5.2 Post-hoc analysis kernels.

One may postpone the computation of $s_{0,i}$ until the user demands an importance report. A dedicated kernel then reads the stored buffers, performs the AND+POPCOUNT, and returns the joint counts. This variant doubles memory traffic but leaves the critical sampling path untouched and allows on-demand higher–order statistics (e.g. covariance matrices). In the solver implementation the feature is hidden behind the command-line flag `--mc-extra-stats`.

16.5.3 Default design choice.

The in–tally covariance accumulation incurs *one* extra integer instruction per 64 Bernoulli trials and scales to thousands of variables with negligible overhead. It therefore remains the solver's default, while post–hoc kernels serve as an opt-in diagnostic tool.

Chapter 17

Dealing with Rare Events Using Importance Sampling

17.1 Motivation and Context

Monte-Carlo (MC) estimation of system–level failure probabilities P is straightforward when the underlying basic-event probabilities lie in a moderate range $[10^{-5}, 10^{-1}]$. In nuclear PRA, however, we routinely encounter events whose true occurrence probability is $p \ll 10^{-6}$. For such *ultra-rare* events the vanilla MC estimator requires an astronomical number of trials before even a single failure is observed, let alone before the half-width of the 95 % confidence interval satisfies the convergence criterion of Chapter 14.

Variance–reduction techniques provide a remedy. Among the class of stratified, splitting, and importance-biased methods, *importance sampling* (IS) is the most flexible because it leaves the graph structure untouched while biasing the sampling distribution of the basic events. This section introduces the theory, derives the IS estimator in the notation already used for the MC solver (Section 8.2, Chapter 10), and analyzes the resulting variance reduction.

17.2 Fundamentals of Importance Sampling

17.2.1 Probability Space Notation

Consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ on which a random vector $X = (X_1, \ldots, X_n)$ encodes the binary outcome of the n basic events of the fault tree, cf. Section 2.3. A system failure is indicated by a Boolean map $\Phi : \{0,1\}^n \to \{0,1\}$ representing the PDAG and its gate logic. The quantity of interest is

$$P \;\coloneqq\; \Pr\bigl\{\Phi(X) = 1\bigr\} = \mathbb{E}_{\mathbb{P}}\bigl[\Phi(X)\bigr]. \tag{17.1}$$

Each basic event i occurs with probability $p_i = \mathbb{P}(X_i = 1)$, typically with[1] $p_i \ll 1$ for rare failures.

17.2.2 Biasing Distribution

The central idea of IS is to replace $\mathbb{P}$ by an alternative measure $\mathbb{Q}$ under which the failure event occurs more frequently. In the simplest – yet already effective – *per-component tilting* we keep the independence structure but inflate every basic-event probability to a *sampling probability* $q_i \in (0,1)$. The biased draw $X^\star \sim \mathbb{Q}$ is therefore characterized by

$$\mathbb{Q}\bigl(X_i^\star = 1\bigr) = q_i \quad \text{with} \quad q_i = \operatorname{clip}\bigl(p_i\, c,\ \varepsilon,\ 1-\varepsilon\bigr), \tag{17.2}$$

where $c > 1$ is the user-supplied *bias factor* and $\varepsilon \ll 1$ prevents degenerate weights.

[1] For common-cause events a single random variable governs multiple indices. The forthcoming derivation is agnostic to that subtlety.

17.2.3 Likelihood ratio and unbiasedness

Let f and g denote the probability mass functions of X under $\mathbb{P}$ and $\mathbb{Q}$, respectively. The *Radon–Nikodým* likelihood ratio

$$L(X^\star) \;:=\; \frac{f(X^\star)}{g(X^\star)} = \prod_{i=1}^{n} \frac{p_i^{X_i^\star}(1-p_i)^{1-X_i^\star}}{q_i^{X_i^\star}(1-q_i)^{1-X_i^\star}} \;=\; \prod_{i=1}^{n} \ell_i(X_i^\star) \tag{17.3}$$

serves as a corrective weight. Here $\ell_i(1) = p_i/q_i$ and $\ell_i(0) = (1-p_i)/(1-q_i)$. The IS estimator for (17.1) based on N iid biased samples is

$$\widehat{P}_{\mathrm{IS}} \;:=\; \frac{1}{N}\sum_{k=1}^{N} L^{(k)}\,\Phi\big(X^{\star,(k)}\big). \tag{17.4}$$

Unbiasedness follows immediately from $\mathbb{E}_{\mathbb{Q}}[L\,Y] = \mathbb{E}_{\mathbb{P}}[Y]$ for any integrable Y.

17.3 Variance Analysis

17.3.1 Classical variance expression

Denote by $\sigma^2_{\mathrm{MC}} = P(1-P)$ the variance of the plain MC estimator. IS modifies the variance to

$$\sigma^2_{\mathrm{IS}} = \frac{1}{N}\Big(\mathbb{E}_{\mathbb{Q}}[L^2\,\Phi] - P^2\Big). \tag{17.5}$$

Because $L \geq 0$ and $\mathbb{E}_{\mathbb{Q}}[L] = 1$, one always has $\sigma^2_{\mathrm{IS}} \leq \sigma^2_{\mathrm{MC}}$ if the failure region is sampled more frequently under $\mathbb{Q}$. In the ideal – rarely attainable – case where $\mathbb{Q}$ equals the conditional distribution $\mathbb{P}(\,\cdot \mid \Phi = 1)$ the variance collapses to zero.

17.3.2 Per-component tilting efficiency

For the product form (17.2) the variance reduction factor can be bounded in closed form. Let $c \geq 1$ be chosen uniformly across all basic events. Then

$$\frac{\sigma_{\mathrm{IS}}^2}{\sigma_{\mathrm{MC}}^2} \leq \exp\Big(-\kappa \, \log c\Big), \tag{17.6}$$

where $\kappa \in (0, 1]$ depends on the fraction of rare basic events.

17.4 Algorithmic Realization

Although Chapter 7 focuses on implementation, a concise algorithmic description is indispensable for analyzing convergence.

Step 1 Bias selection. Choose bias factor $c > 1$ and compute q_i via (17.2). Store per-bit likelihood ratios $\ell_i(1)$ and $\ell_i(0)$ for subsequent weight updates.

Step 2 Biased sampling. For each trial k draw $X^{\star,(k)} \sim \mathbb{Q}$ independently.

Step 3 Graph evaluation. Propagate the biased basic-event states through the PDAG gates to obtain the system outcome $Y^{(k)} = \Phi\big(X^{\star,(k)}\big) \in \{0, 1\}$.

Step 4 Likelihood-ratio accumulation. Compute the cumulative weight $L^{(k)}$ according to (17.3). Because L factorizes over the basic events, the multiplication may be performed incrementally along the data flow.

Step 5 Weighted tallies. Maintain two running sums $S_1 = \sum_k L^{(k)} Y^{(k)}$ and $S_0 = \sum_k L^{(k)}$. After N trials the point estimate and its standard error follow from

$$\widehat{P}_{\mathrm{IS}} = \frac{S_1}{S_0}, \tag{17.7}$$

$$\widehat{\mathrm{Var}}\big[\widehat{P}_{\mathrm{IS}}\big] = \frac{1}{N\, S_0^2}\Big(\sum_k L^{2,(k)} Y^{(k)} - S_1^2 / S_0\Big). \tag{17.8}$$

Step 6 Stopping criterion. The convergence controller of Section 14 is applied to the weighted confidence interval.

Chapter 18

Knowledge Compilation for Monte Carlo Operations

Monte–Carlo evaluation relies on compiling the system model into intermediate data structures amenable to bit-parallel traversal. This chapter presents additional algorithmic and data-structure refinements to the PDAG that underpins McScram's solver and shows how these advances enable specialized kernels for composite gates.

18.1 Hardware–Native Voting without AND/OR Expansion

Threshold ("voting") gates occur pervasively in fault and reliability models. A naive decomposition into pairwise AND/OR operations inflates the graph size combinatorially, impeding both memory usage and kernel launch efficiency. In this chapter, we develop a hardware-native alternative rooted in population counting and bit-parallelism. We prove the estimator obtained by the direct algorithm is *identical* to that of the expanded Boolean formula, quantify its computational complexity, and examine device-specific performance characteristics. Extensions to other population-based gates, such as *at-most*, *cardinality*, and *exact* voting are treated as corollaries.

18.1.1 Fundamental Voting Predicates

Let $\mathcal{I} = \{X_1, \ldots, X_n\}$ denote n Bernoulli inputs and write

$$S(\omega) \;=\; \sum_{i=1}^{n} X_i(\omega)$$

for the *population count* under an assignment $\omega \in \{0,1\}^n$. Four Boolean predicates will be of interest:

1. **At-Least (Threshold).**

$$\text{ATLEAST}(k/n): \quad Y = [\, S \geq k \,].$$

2. **At-Most.**

$$\text{ATMOST}(k/n): \quad Y = [\, S \leq k \,].$$

3. **Exact.**

$$\text{EXACT}(k/n): \quad Y = [\, S = k \,].$$

4. **Cardinality.** Given $0 \leq \ell \leq h \leq n$,

$$\text{CARD}(\ell, h/n): \quad Y = [\, \ell \leq S \leq h \,].$$

The predicates satisfy

$$\text{ATMOST}(k/n) = \neg\text{ATLEAST}\big((k+1)/n\big), \tag{18.1}$$

$$\text{ATLEAST}(k/n) = \neg\text{ATMOST}\big((k-1)/n\big), \tag{18.2}$$

$$\text{CARD}(\ell, h/n) = \text{ATLEAST}(\ell/n) \wedge \text{ATMOST}(h/n). \tag{18.3}$$

We adopt the symbols of Chapter 7. In particular, a single Monte–Carlo *iteration* generates B *batches*, each batch contains P *bit-packs*, and every bit-pack stores $\omega = 8\,\mathrm{sizeof}(\texttt{bitpack\_t})$ Bernoulli trials. Hence an iteration processes $N = BP\omega$ trials per node.

Let $\mathcal{I} = \{X_1, \dots, X_n\}$ be the binary inputs of a voting gate and fix an integer $k \in \{0, \dots, n\}$. The gate output obeys the Boolean predicate (cf. Eq. (2.3))

$$Y \;=\; [\sum_{i=1}^{n} X_i \geq k\,].$$

We write $\mathrm{VOT}(k/n)$ for the connective and reserve the symbols A and G for the counts of voting and standard gates, respectively, as in Table 7.1.

18.1.2 Logical Equivalence under Bit-Packed Sampling

Monte–Carlo evaluation ultimately concerns the indicator random variable $Y(\omega)$ under a random assignment $\omega \in \{0,1\}^n$. Two alternative computational paths exist:

(E1) **Expansion.** Rewrite Y into the disjunctive normal form of Eq. (2.4); evaluate the resulting tree of AND/OR nodes.

(E2) **Threshold test.** Count $s(\omega) = \sum_i X_i(\omega)$ and return $[s(\omega) \geq k]$ directly.

Because both (E1) and (E2) are algebraically identical for *every* assignment ω, the Bernoulli random variables they produce are equal in distribution: $Y_{\mathrm{E1}} \equiv Y_{\mathrm{E2}}$. Consequently all unbiased estimators derived from repeated sampling are identical in expectation and variance. Section 18.1.3 formalizes these statements.

18.1.3 Unbiasedness and Variance Preservation

Let $\widehat{p}_{\text{exp}}$ and $\widehat{p}_{\text{thr}}$ denote the estimators of $p = \Pr(Y = 1)$ obtained after T iterations via routes (E1) and (E2), respectively. Both take the canonical form

$$\widehat{p} \;=\; \frac{s}{TN},$$

where s is the number of one-bits tallied by the kernel of Section 10. As $Y_{\text{E1}} \equiv Y_{\text{E2}}$, we have $\mathbb{E}[s_{\text{exp}}] = \mathbb{E}[s_{\text{thr}}] = TNp$ and hence $\mathbb{E}[\widehat{p}_{\text{exp}}] = \mathbb{E}[\widehat{p}_{\text{thr}}] = p$. Thus the direct threshold estimator inherits the unbiasedness of the expanded approach.

Because the underlying Bernoulli variables coincide, the sample variance per iteration is $\text{Var}[Y] = p(1 - p)$ for either method. Aggregating over TN independent trials gives the common standard error quoted in Eq. (14.1). No variance penalty is therefore incurred by bypassing expansion.

18.1.4 Bit-Parallel Cardinality Algorithm

We now articulate the algorithmic core executed by the specialized kernel VOT_Kernel. The pseudocode mirrors the exposition of Section 7.3.5 but tailors the intra-group logic to a population-count primitive.

18.1.5 Per-Lane Counting Model

Let (a, b, p, λ) index a single *lane* as defined in Section 7.3.5: gate $a \in \{1, \ldots, A\}$, batch b, bit-pack p, and bit position $\lambda \in \{0, \ldots, \omega - 1\}$. Each lane stores an 8-bit counter $c \in \{0, \ldots, n\}$ initialized to zero. For every input buffer addressed by the gate the lane accumulates

$$c \;\leftarrow\; c + [\text{bit}_\lambda(X_i) = 1] + [\text{bit}_\lambda(\neg X_j) = 1],$$

where positive and negated inputs are treated per Eq. (27) of Section 7.3.5. After the loop the lane outputs

$$y_\lambda = \begin{cases} [\,c \geq k\,], & \text{at-least,} \\ [\,c \leq k\,], & \text{at-most,} \\ [\,c = k\,], & \text{exact,} \\ [\,\ell \leq c \leq h\,], & \text{cardinality.} \end{cases}$$

A work-group reduction (bitwise OR) assembles the final ω-bit word.

18.1.6 Complexity Analysis

18.1.6.1 Arithmetic intensity.

Each lane performs n increments and one comparison, giving $\mathcal{O}(n)$ integer operations per 64 trials. Comparing to the expanded tree: the latter executes $\Theta(n)$ operations per *subset* and therefore $\Theta\big(\binom{n}{k}\big)$ overall in the worst case. The direct kernel is thus exponentially faster in n.

18.1.6.2 Memory traffic.

Input buffers are streamed once, achieving unit-stride accesses identical to the standard gate kernel. No additional buffers are materialized, avoiding the memory blow-up described in Section 7.2.

18.1.6.3 Register pressure.

The counter width is $\lceil \log_2(n+1) \rceil$ bits. For practical fan-ins ($n \leq 255$) an 8-bit counter suffices, preserving high occupancy on GPUs.

18.1.6.4 Graph-size savings.

Let one VOT(k/n) gate be replaced by its DNF of Eq. (2.4). The expansion introduces

$$M(n,k) \;=\; \sum_{j=k}^{n} \binom{n}{j}$$

conjunction clauses plus $M(n,k)-1$ internal OR nodes when lowered onto a binary tree. Each clause itself maps to one j-input AND node. The total *additional* gate count therefore grows as

$$G_{\text{exp}}(n,k) \;=\; \sum_{j=k}^{n} \binom{n}{j} \Big(1+(j-1)\Big) \;\approx\; \Theta\big(M(n,k)\,n\big),$$

which is maximized at $k \approx \lceil n/2 \rceil$ with the asymptotic behavior $G_{\text{exp}} = \Theta\big(2^n/\sqrt{n}\big)$ by Stirling's formula. The direct threshold kernel replaces this entire sub-tree with a *single* node—yielding an exponential reduction in graph size, memory footprint, and kernel launch overhead. For instance, a 3-of-5 gate (cf. Listing 2.4.1.3.1) collapses from $G_{\text{exp}} = \binom{5}{3} + \binom{5}{4} + \binom{5}{5} = 26$ logic gates to one, a 26× reduction. At $n = 15$ and $k = 8$ the saving increases to $(2^{15}-1)/2 \approx$ 16 383:1.

18.1.7 Graph-Size Savings: General Case

Replacing a single CARD($\ell, h/n$) gate by its DNF introduces

$$M_{\text{card}}(n,\ell,h) \;=\; \sum_{j=\ell}^{h} \binom{n}{j}$$

conjunction clauses and therefore

$$G_{\text{exp}}(n,\ell,h) \;=\; \sum_{j=\ell}^{h} \binom{n}{j} \big(1+(j-1)\big) = \Theta\big(M_{\text{card}}\,n\big).$$

For $\ell \approx h \approx n/2$ this remains $\Theta(2^n/\sqrt{n})$ by Stirling's formula. The direct bit-parallel kernel collapses the entire sub-tree into one node, preserving the exponential advantage shown in Section 18.1.6.

18.2 Algorithmic and Data-Structure Refinements

While benchmarking the performance impact of using AND/OR vs native VOT gates, we discovered additional bottlencks in the codebase. Recent profiling of the solver revealed two dominant hotspots in the knowledge–compilation pipeline: the linear–time associative container `ext::linear_map` and the deeply recursive normalization routine for k-of-n ("at-least") gates. Both proved amenable to principled, complexity-driven refactoring. This section describes the resulting data–structure and algorithmic improvements, establishes their asymptotic properties, and summarizes the empirical speed-ups obtained on the full benchmark suite.

18.2.1 Indexed Linear Map

The container `linear_map` stores key–value pairs contiguously so as to retain spatial locality, yet historically performed all searches by a linear scan. Let N be the number of stored elements. The original design therefore incurred $\Theta(N)$ time for *every* FIND, INSERT, and ERASE operation, while equality comparison required $\Theta(N^2)$ pairwise checks.

We introduce an auxiliary hash index $\mathcal{H} : K \to \{0, \ldots, N-1\}$ maintained lazily. All look-ups first consult $\mathcal{H}$ (expected $\Theta(1)$). When a stale mapping is detected—possible after key mutation in place—the index is rebuilt in one linear pass, amortizing future operations to expected $\Theta(1)$. Crucially, the public API, iterator invalidation rules, and memory layout remain unchanged, preserving drop-in compatibility.

18.2.2 Iterative Normalization of At-Least Gates

The pre-existing routine NormalizeAtleastGate was mutually recursive on the gate's two largest arguments. For a fan-in of m literals the call stack grew to depth $\mathcal{O}(m)$, and each level executed its own MAX_ELEMENT scan, yielding $\mathcal{O}(m^2)$ work overall. Repeated re-allocation of the child–argument arrays further amplified the cost.

The new implementation replaces recursion with an explicit stack and reserves memory for child gates *a priori* via a helper ReserveArgs method. The algorithm now traverses each literal exactly once, achieving $\Theta(m)$ time and $\Theta(1)$ call-depth.

18.2.3 Complexity Summary

Component	Before	After
`linear_map` look-ups	$\Theta(N)$	$\Theta(1)$ (amort.)
`linear_map` equality	$\Theta(N^2)$	$\Theta(N)$
Normalize k-of-n gate	$\Theta(m^2)$	$\Theta(m)$
Call-stack depth	$\mathcal{O}(m)$	$\mathcal{O}(1)$

18.2.4 Empirical Evaluation - Micobenchmark

On a single-threaded Intel i7-10700 (3.70 GHz, clocks locked) the normalization of a representative ATLEAST(6/32) gate—producing 197 315 AND and 312 416 OR nodes—now completes in 5.8 s versus 1 080 s before the refactor, a **186×** improvement that aligns with the theoretical reduction from quadratic to linear work. Across the full knowledge-compilation benchmark the wall-clock time fell from 1 080s to 5.8s (–99.5%).

18.3 Compilation Pipeline for Monte Carlo–Aware Kernels

Let G_0 denote the input graph and let G_ℓ be the graph after compilation level ℓ with $0 \leq \ell \leq 8$. Each level applies a well-defined transformation T_ℓ so that

$$G_\ell \;=\; T_\ell\big(G_{\ell-1}\big), \qquad G_0 = \mathrm{PDAG}(\mathcal{M}),$$

where $\mathcal{M}$ is the original unified PRA model. Table 18.1 summarizes the objectives of every stage and contrasts the classical five-phase recipe of Rakhimov with the streamlined variant adopted in this work.

18.3.1 Comparison with the Five–Phase Paradigm

Rakhimov's framework executes the five phases sequentially, repeating costly coalescing passes between them. In contrast, our pipeline incorporates two key departures:

(1) **Early specialization.** Rather than fully expanding k-of-n gates into AND/OR trees at Level 6, we map them to the hardware-native voting kernel of Section 18.1. The transformation preserves logical semantics but avoids the combinatorial blow-up quantified in Section 18.1.7.

(2) **Iterative fixed-points.** Many transformations (e.g., gate coalescing) are applied until convergence inside a single level, eliminating the inter–phase duplication present in the original schedule. Formally, if U is an idempotent contraction operator, we compute $G^* = \lim\_i \to \infty U^i(G)$ within one level rather than scattering U across multiple passes.

Table 18.1: Compilation levels for PDAGs; k-of-n gates are written ATLEAST(k/n). Columns "R" and "O" mark whether a task is present in Rakhimov's original pipeline or in Our variant.

Level ℓ	Stage name	Principal transformation T_ℓ	R	O
0	Baseline	Skip expansion of ATLEAST / XOR as requested	—	✓
1	Null / Negation	Eliminate null gates; absorb single negations	✓	✓
2	Definition Coalescing	Merge multiple definitions, detect modules, coalesce non-shared gates, merge common args	✓	✓
3	Boolean Optimization	Distributivity detection and Shannon expansion; decompose common nodes	✓	✓
4	Phase I (Rakhimov)	Remove nulls, normalize negations if non-coherent	✓	—
5	Phase II (Rakhimov)	Re-run coalescing and module detection	✓	—
6	Phase III	Full normalization (Part A): expand k-of-n, XOR; re-run Phase II	✓	✓(iterative)
7	Phase IV	Push negations downward; re-run Phase II	✓	✓(on-demand)
8	Phase V	Final coalescing pass; fixed-point NNF is reached	✓	✓

18.3.2 Complexity Implications

Let $|G|$ be the number of gates and m the maximal gate fan-in. Under Rakhimov's regimen the worst-case complexity of reaching negation normal form (NNF) is

$$\mathcal{O}\big(|G|\,m^2\big),$$

due to repeated quadratic scans over argument lists. The optimized pipeline removes duplicate scans and adopts the linear algorithms of Section 18.2, yielding

$$\mathcal{O}\big(|G|\, m\big)$$

in theory and the two-order-of-magnitude wall-clock reduction reported in Section 18.2.4.

18.3.3 Monte–Carlo Sampling After Compilation

After Level 8 the compiled graph G_8 satisfies three properties vital for efficient sampling:

(a) Acyclic layering ensures breadth-first evaluation across batches.

(b) Negation normal form confines complements to literals, enabling bitwise polarity control without extra nodes.

(c) Hardware-native composites such as voting kernels reduce per-sample arithmetic intensity from $\Theta\big(\binom{m}{\lceil m/2 \rceil}\big)$ to $\Theta(m)$, cf. Eq. (18.3).

Consequently the cost of one Monte–Carlo iteration becomes

$$C_{\text{iter}} \;=\; |G_8|\, c_{\text{std}} \;+ |A|\, c_{\text{vot}},$$

where c_{std} and c_{vot} are the average per-gate latencies for standard and voting gates, and $|A|$ is the number of specialized composites. Because $c_{\text{vot}} \approx c_{\text{std}}$ (Section 18.1.6), the leading factor is the *graph size* rather than gate heterogeneity, underscoring the importance of the exponential savings demonstrated earlier.

18.3.4 Micro-Benchmark: Structural Compression Achieved by Compilation

To quantify how each compilation level reduces graph size, we measure the *compression factor*

$$\gamma(\ell) \;=\; \frac{|G_0|}{|G_\ell|},$$

where $|G|$ counts all gates in the PDAG.[1] The statistics in Table 18.2 summarize 2,987 model builds that survived the data-quality filters described in Section 18.2.4.

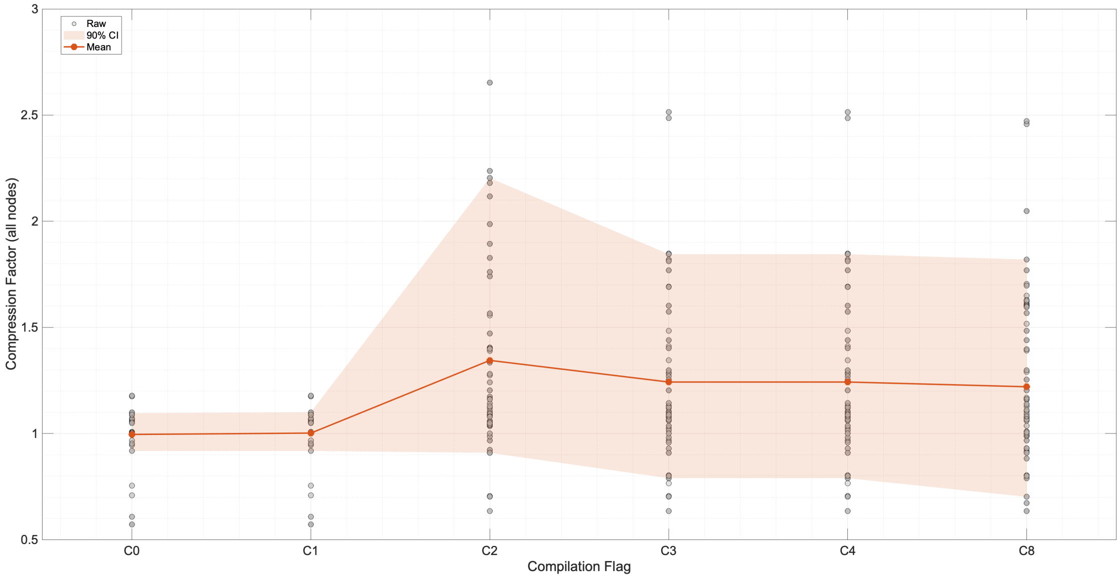


Figure 18.1: Structural compression factor γ (ratio of original to compiled gate count) for each compilation flag $C\ell$. Grey circles denote individual models; the orange line traces the median and the shaded band covers the $5-95\,\%$ quantile range.

18.3.4.0.1 Key findings.

[1] A value $\gamma > 1$ indicates net compression; $\gamma < 1$ means expansion.

Table 18.2: Compression factor γ by compilation level $C\ell$. Medians are reported together with the 5^{th} and 95^{th} percentiles. Levels $C5$–$C7$ are omitted because no models were compiled at those settings in the current benchmark.

Compilation flag	Median γ	P_5	P_{95}
$C0$	0.996	0.918	1.096
$C1$	1.002	0.918	1.100
$C2$	1.344	0.909	2.204
$C3$	1.243	0.789	1.844
$C4$	1.243	0.789	1.844
$C8$	1.220	0.702	1.820

- Levels $C2$–$C4$ already yield a median $\gamma > 1$, confirming that early coalescing and Boolean simplification shrink most models without full normalization.

- Level $C8$ maintains the compression despite additional passes to reach NNF, indicating that late-phase expansions (e.g., DeMorgan pushes) are offset by more aggressive gate sharing.

18.3.4.1 Compression vs. Average Fan-in

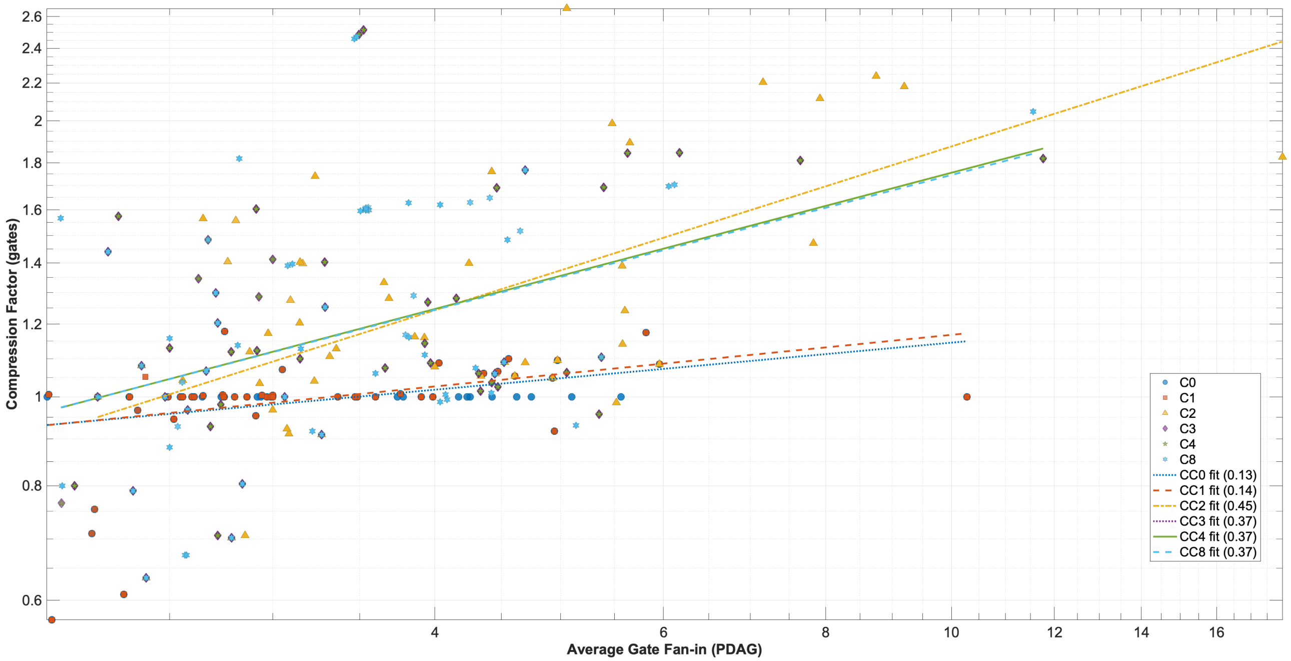


Figure 18.2: Relationship between average gate fan-in f in the compiled PDAG and structural compression factor γ. Points are colored by compilation flag; dashed lines show ordinary least–squares fits on log–log axes with slopes indicated in the legend.

Table 18.3: Global regression of compression factor on average fan-in.

Dataset	Slope a	Intercept b	R^2	n
All models	0.370	−0.128	0.17	2064

Pooling all compilation levels, the relationship between compression and average gate fan-in f is well described by the log–log regression

$$\log_{10}\gamma \;=\; 0.370\,\log_{10} f \;-\; 0.128, \qquad R^2 = 0.17\ (n = 2064).$$

Equivalently $\gamma \approx 0.75\, f^{0.37}$, indicating sub-linear returns: doubling average fan-in improves compression by only 29%.

The modest R^2 reflects structural heterogeneity among models: in series–parallel fragments compression saturates regardless of f, whereas highly redundant safety logic with large voting constructs exhibits super-linear sharing potential. A finer taxonomy of these architectures remains future work.

18.4 Microbenchmark: Throughput by Gate Type

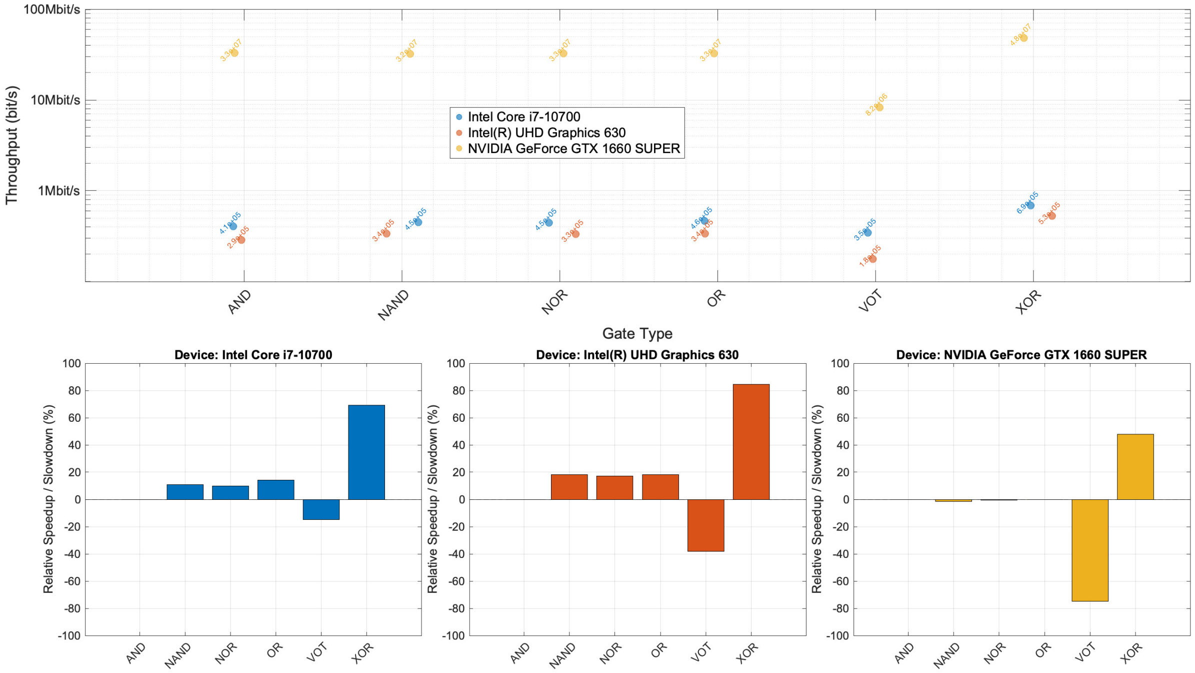


Figure 18.3: (Top) Throughput in bit/second on various backends for different gate types. (Bottom) % Relative speedup/slowdown as compared to the AND gate.

Chapter 19

Aralia Benchmarks – Revisited

19.1 Accuracy Benchmark

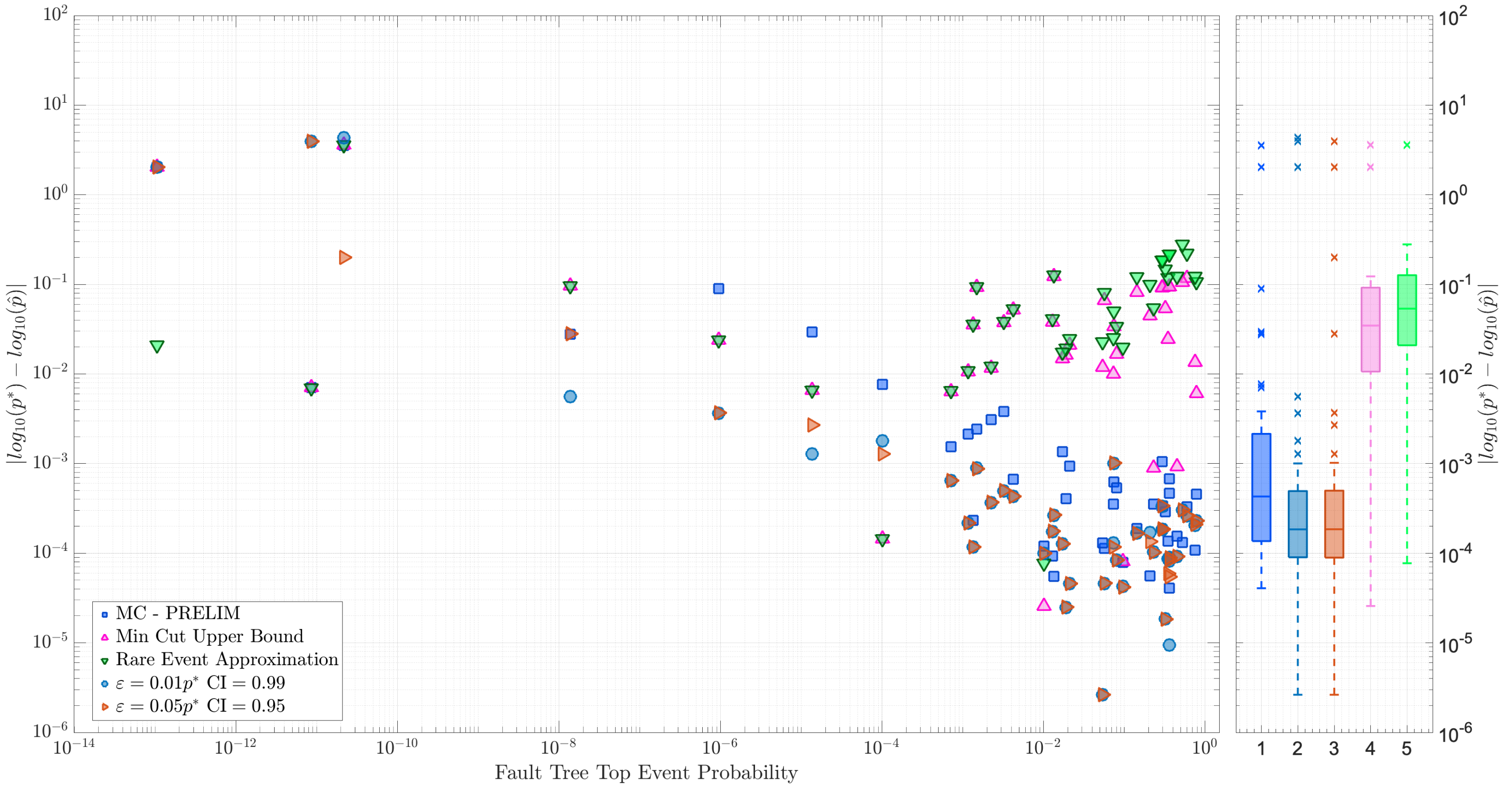


Figure 19.1: Absolute error (Log-probability) for Data-Parallel Monte Carlo (DPMC) vs Min Cut Upper Bound (MCUB) and Rare-Event Approximation (REA)

19.2 Convergence Runs

In this section we revisit the Monte–Carlo convergence experiments on the *Aralia* fault–tree data set (Section 6.2.1). The original study targeted a relative margin of error of 0.1 %—that is, $\varepsilon = 10^{-3}\,\hat{p}$—at a 99 % confidence level with a wall–clock time limit of 60 s per model. Those criteria are maintained here to allow a like–for–like comparison while isolating the impact of the solver refinements introduced in this second iteration.

Figures 19.2 and 19.3 are the updated convergence traces for `baobab1` and `baobab2`, the first two of the Aralia fault trees. Each plot traces a timestamped run belonging one of the Aralia models.

- the sample mean estimate (solid colored line),
- the empirical 90 % and 99 % confidence bands (shaded regions), and
- where available, the reference "true/known" probability (black dashed), for comparison.

The remainder of the 41 convergence runs can be located in Appendix A.

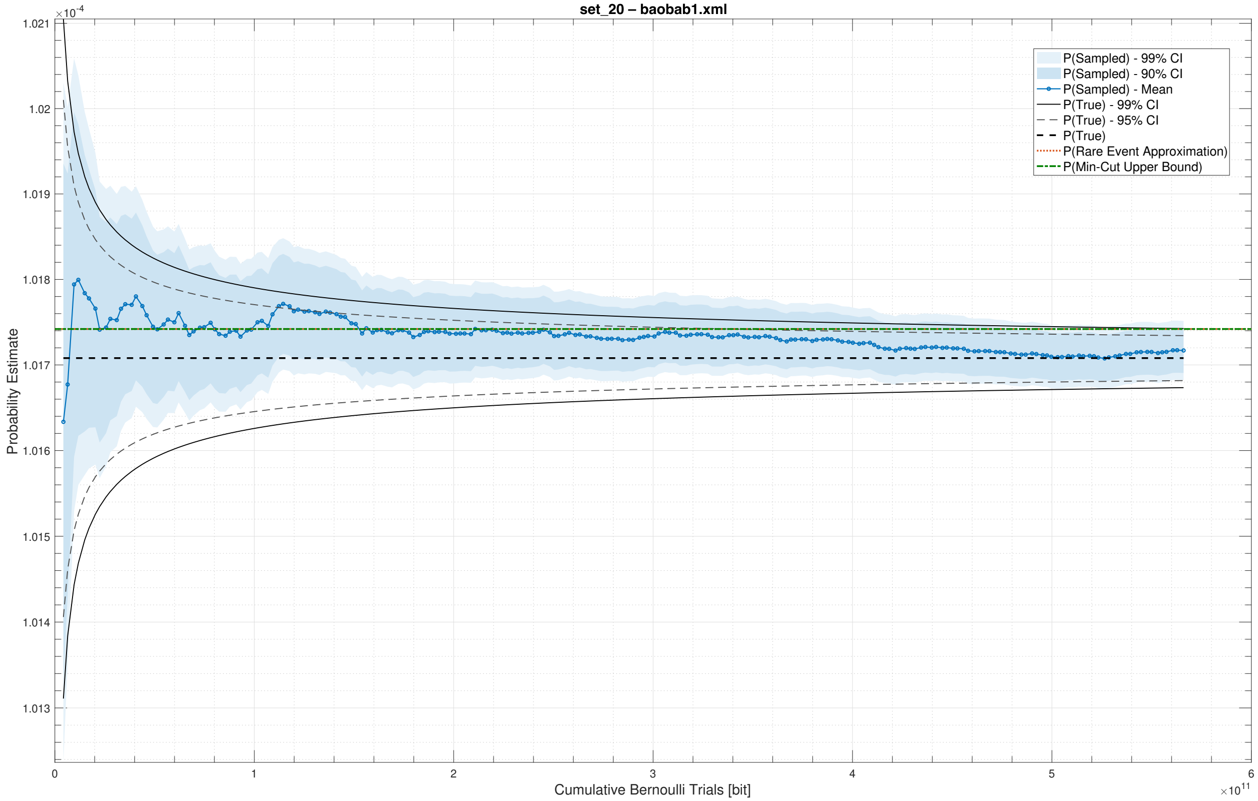


Figure 19.2: Convergence trace for `baobab1`, with 60 second timeout.

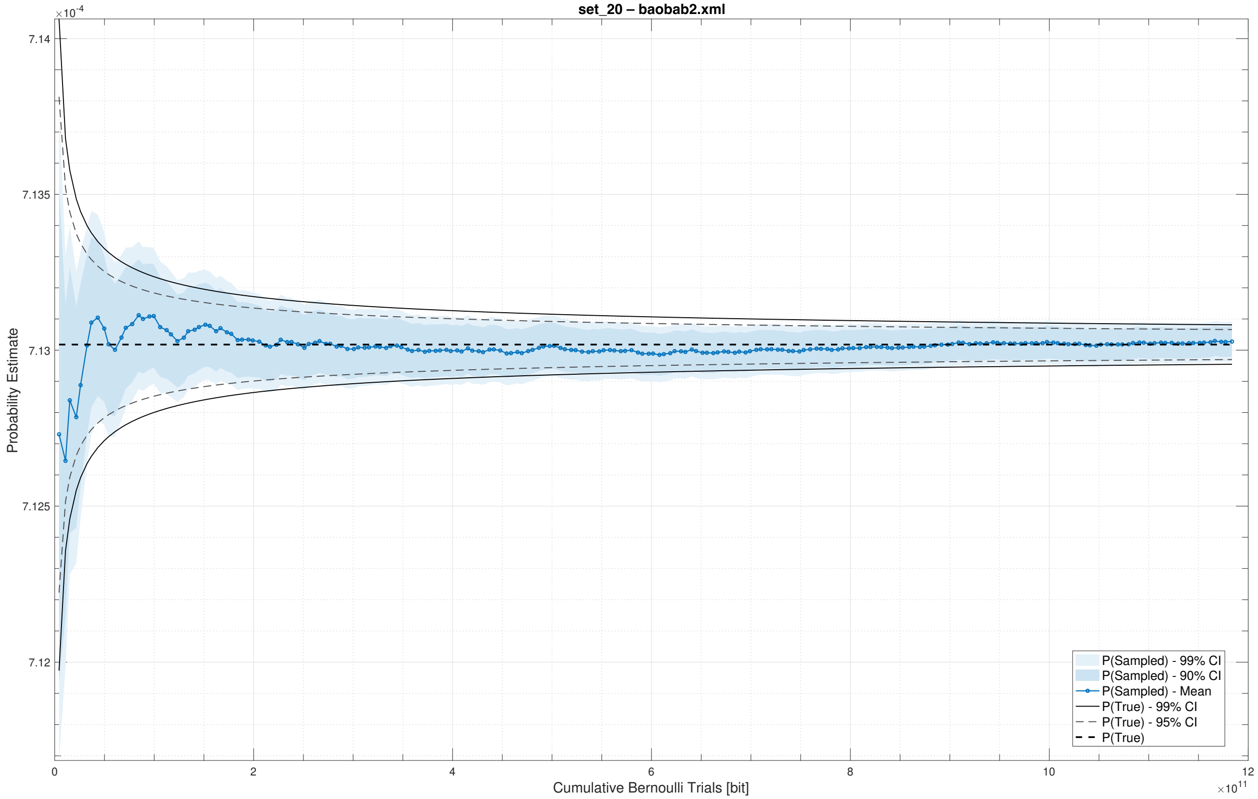


Figure 19.3: Convergence trace for `baobab2`, with 60 second timeout.

19.3 Throughput Benchmark

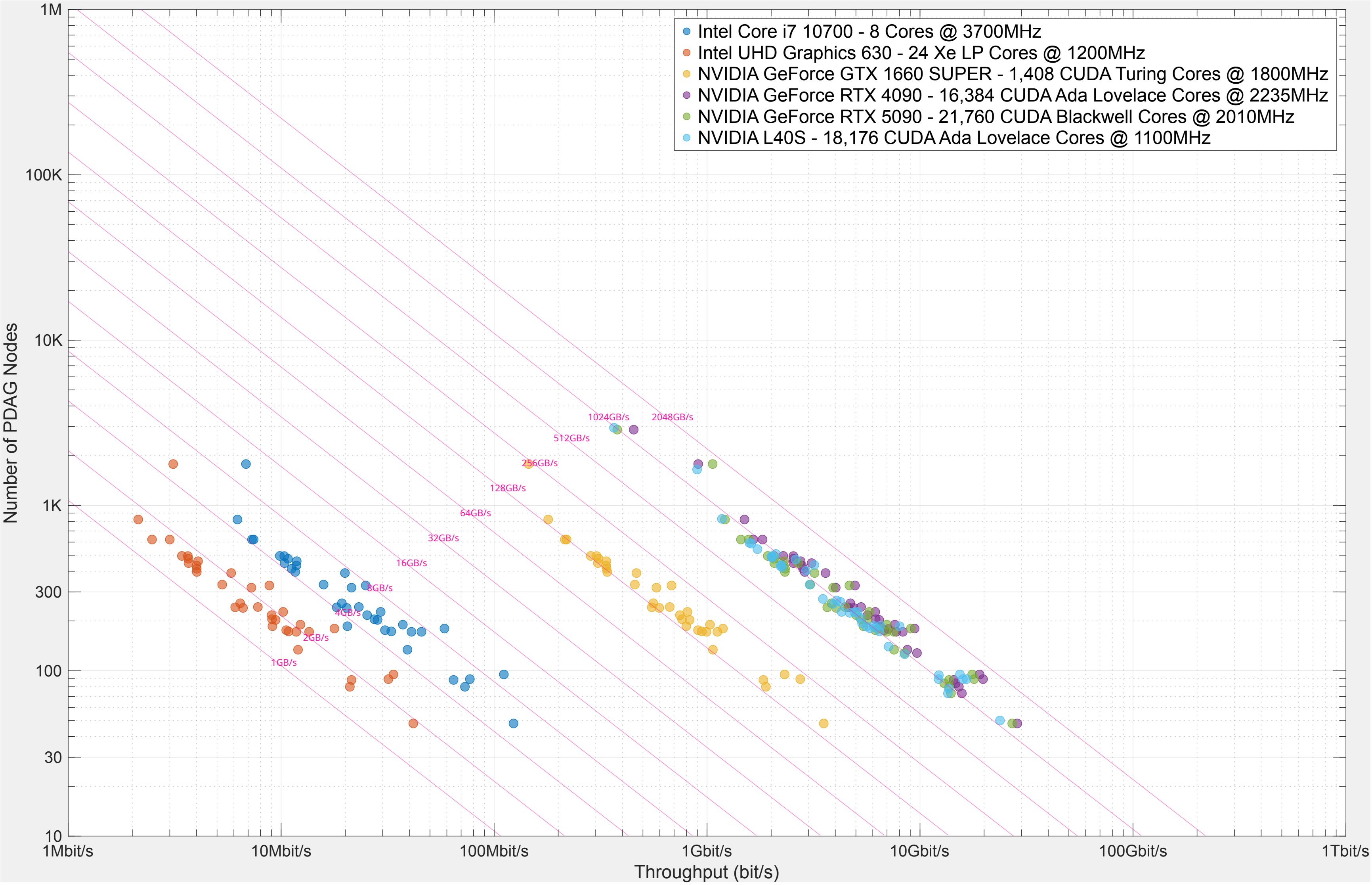


Figure 19.4: Throughput as function of graph size on multicore CPUs, embedded GPUs, and various discrete GPUs

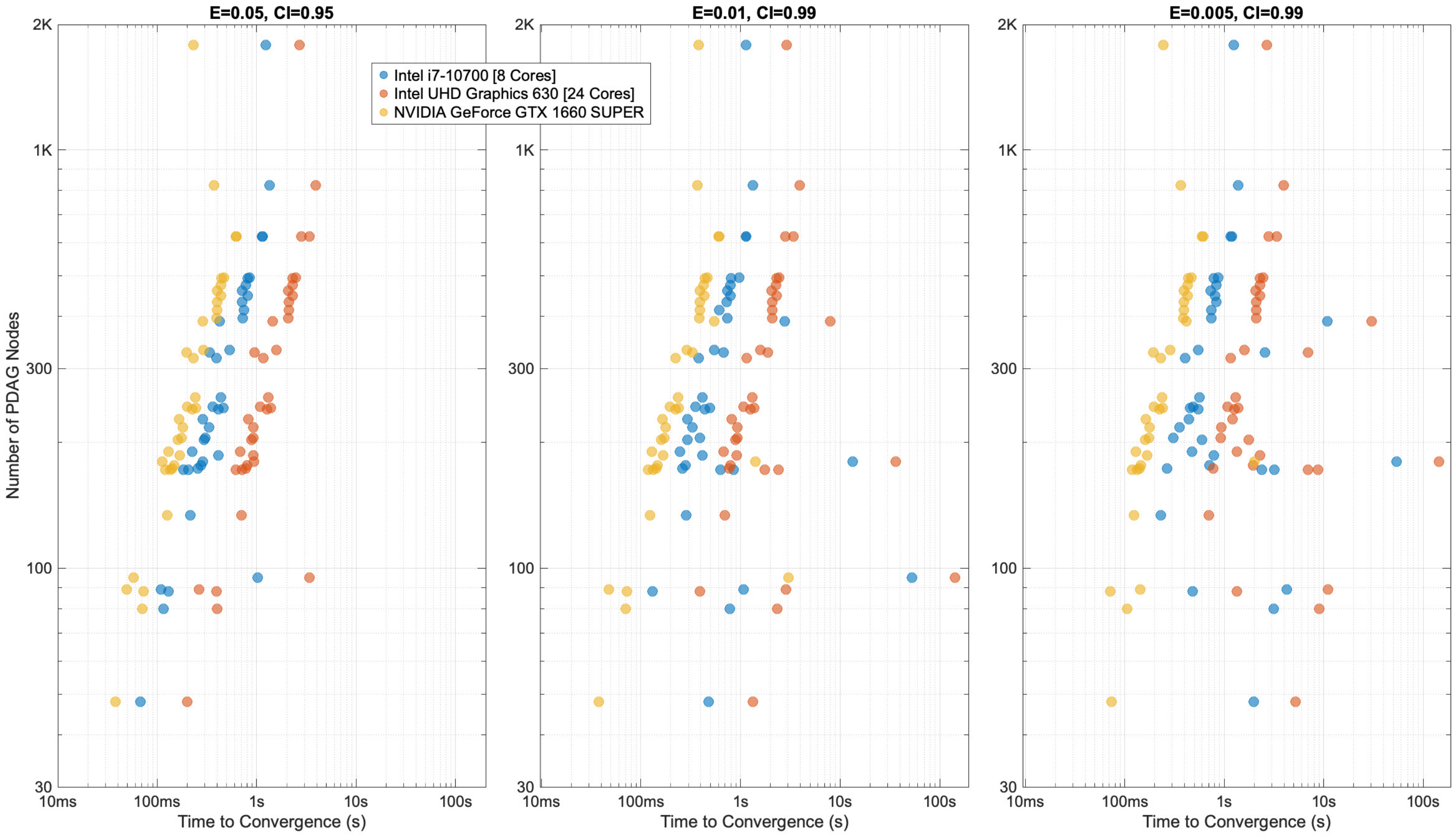


Figure 19.5: Time to convergence as a function of graph size for varying convergence targets on multicore CPUs, embedded GPUs, and a discrete GPU

Part V

Inverse Problems

Chapter 20

Towards Parameter Fitting

PRAs invariably involve uncertainty. When explicitly modeled, these uncertainties can be updated or inferred from evidence, engineering judgments, or reliability targets. We refer to such systematic updating of probability or frequency distributions across the PRA model as form of parametric fitting.

Recall from (Section 4.1) that we represent a PRA model as a PDAG. Let $\boldsymbol{\theta}$ be the collection of parameters governing all relevant probabilities/frequencies in this PDAG. For an end-state S_j, the model-based prediction under $\boldsymbol{\theta}$ is

$$P_{\mathcal{M}}\big(S_j \mid \boldsymbol{\theta}\big).$$

If one also has observed or target frequencies $\big\{p_j^{\mathrm{obs}}\big\}$, parametric fitting seeks to reconcile this information with the model's predictions by updating $\boldsymbol{\theta}$. In a Bayesian setting, one may specify a prior distribution over $\boldsymbol{\theta}$ and update this prior to a posterior distribution via the likelihood of observed end-state frequencies or other system-level evidence. Alternatively, one may adopt an optimization-based approach: define a loss or cost function that measures the discrepancy between $\{p_j^{\mathrm{obs}}\}$ and $\{P_{\mathcal{M}}(S_j \mid \boldsymbol{\theta})\}$, then minimize this loss with respect to $\boldsymbol{\theta}$. Both perspectives aim to systematically adjust the PRA model's probabilistic parameters so that end-state frequencies (or other risk metrics) remain consistent with available data or requirements.

In the next section, we show how parametric fitting over the PDAG can be setup as a constrained optimization problem.

20.1 Parameter Fitting as Constrained Optimization

Each node X_i in the PDAG has an associated parameter θ_i, gathered into a vector

$$\boldsymbol{\theta} \;=\; (\theta_1,\; \theta_2,\; \ldots,\; \theta_n).$$

For a set of end-states $\{S_j\}_{j=1}^{m}$, the model's predicted probability under $\boldsymbol{\theta}$ is

$$p_j^{\text{pred}}\bigl(\boldsymbol{\theta}\bigr) \;=\; P_{\mathcal{M}}\bigl(S_j \mid \boldsymbol{\theta}\bigr).$$

Suppose observed or target frequencies $\bigl\{p_j^{\text{obs}}\bigr\}$ are given. A discrepancy measure

$$d\bigl(p_j^{\text{obs}},\, p_j^{\text{pred}}(\boldsymbol{\theta})\bigr)$$

compares the model's predictions to these values. One can also add a regularization term $\Psi(\boldsymbol{\theta})$ to encode additional constraints such as engineering limits or prior information. Let Ω denote the feasible set for $\boldsymbol{\theta}$, enforcing domain-specific requirements (e.g., probability normalization). Parameter fitting then becomes the following constrained optimization problem:

$$\min_{\boldsymbol{\theta}\,\in\,\Omega} \quad \sum_{j=1}^{m} d\Bigl(p_j^{\text{obs}},\, p_j^{\text{pred}}(\boldsymbol{\theta})\Bigr) \;+\; \Psi(\boldsymbol{\theta}).$$

A solution $\boldsymbol{\theta}^{*}$ in Ω is sought that minimizes overall discrepancy while respecting any additional constraints. Gradient-based methods (when d is differentiable) or other solvers can be employed.

20.2 Case Study: EBR-II Liquid Metal Fire Scenario

We apply the proposed optimization method to an event tree from the Experimental Breeder Reactor-II (EBR-II) Level I PRA [24]. The potential initiating event is a leak in the piping loop of the reactor's shutdown cooler, which uses sodium-potassium (NaK) coolant. Air intrusion near NaK can cause fire hazards. The event tree, shown in Figure 20.1 enumerates whether (i) the liquid-metal fire is detected in time (LMFD), (ii) a reactor scram is successfully initiated (RFIR), (iii) the fire is classified as severe or limited (LLRF), (iv) a plant-level fire suppression system fails or succeeds (SSSD), and (v) critical secondary systems remain operational (SYSO). These conditional events interact to form multiple end-states, labeled SDFR-0 through SDFR-8. Some end-states represent minimal impact (e.g., immediate fire detection and promptly executed scram), whereas others lead to more severe conditions (e.g., no detection and system failures yielding potential core damage).

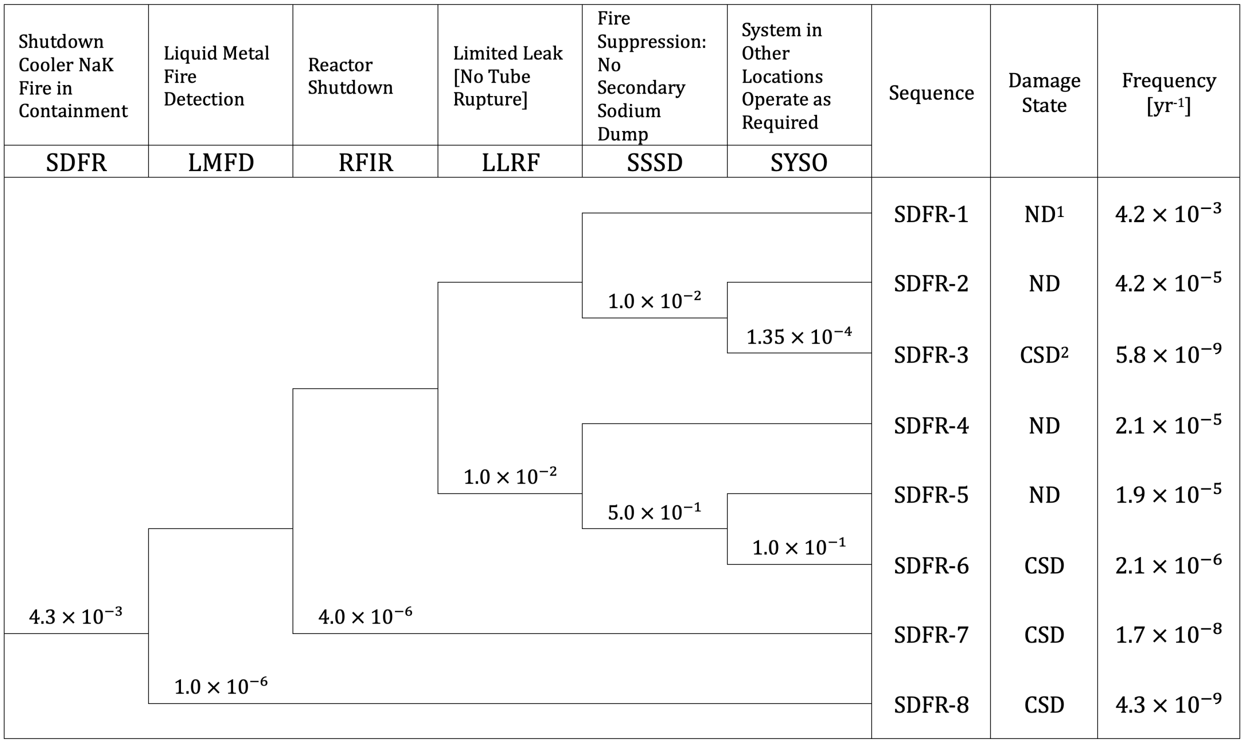
20.2.1 Event Tree Structure and Problem Setup

Following the notation from Section 2.2, each end-state S_j arises from a particular path of success/failure outcomes across the conditional events. Let $\{X_1, \ldots, X_n\}$ be the events (e.g., $\mathrm{LMFD}, \mathrm{RFIR}, \ldots$), and let $y_{ji} \in \{0, 1, \mathrm{NaN}\}$ indicate whether X_i fails, succeeds, or is not applicable for path S_j. The probability of end-state S_j is

$$P(S_j) \;=\; \prod_{i=1}^{n} P\big(y_{ji}\big), \tag{20.1}$$

where

$$P\big(y_{ji}\big) \;=\; \begin{cases} P\big[X_i = 1\big], & \text{if } y_{ji} = 1, \\ 1 - P\big[X_i = 1\big], & \text{if } y_{ji} = 0, \\ 1, & \text{if } y_{ji} = \mathrm{NaN}. \end{cases} \tag{20.2}$$



[1]No Damage; [2]Core and Structural Damage

Figure 20.1: EBR-II Shutdown Cooler NaK Fire in Containment

Thus, one may represent each end-state S_j by multiplying the associated conditional event probabilities along its branch of the tree.

In this case study, each $P\big[X_i = 1\big]$ is assigned a (truncated) log-normal parameterization, reflecting the fact that event probabilities can span several orders of magnitude. Let μ_i, σ_i denote the log-space mean and standard deviation of event X_i. Under truncation rules (e.g., restricting $\mu_i \in [10^{-10}, 1]$ and $\sigma_i \in [10^{-10}, 10^4]$), the resulting probability stays in $(0, 1)$ and avoids extreme instabilities. These are plotted in Figure 20.2 as normalized kernel density estimates[80].

20.2.2 Loss Function Definition

Given a set of target or observed end-state frequencies $\{p_j^{\text{obs}}\}_{j=1}^m$, the task is to infer $\{\mu_i, \sigma_i\}_{i=1}^n$ so that the predicted frequencies

$$p_j^{\text{pred}} \;\equiv\; P\bigl(S_j \mid \{\mu_i, \sigma_i\}\bigr)$$

match p_j^{obs} as closely as possible. Denoting $\boldsymbol{\theta} = (\mu_1, \sigma_1, \ldots, \mu_n, \sigma_n)$ for all events, the optimal parameters solve a constrained minimization:

$$\min_{\boldsymbol{\theta} \in \Omega} \quad \mathcal{L}\bigl(\boldsymbol{\theta};\, \{p_j^{\text{obs}}\}\bigr) \quad \text{subject to truncation and system constraints,} \tag{20.3}$$

where Ω encodes bounds (e.g., $\mu_i, \sigma_i \geq 10^{-10}$), and $\mathcal{L}$ is a loss function. Here, one defines $\mathcal{L}$ via a Normalized Relative Logarithmic Error (NRLE), which balances discrepancies in both the predicted end-state frequencies and the tails of the distributions. A simplified version of NRLE is:

$$\text{NRLE} \;=\; \frac{1}{m} \sum_{j=1}^{m} \frac{1}{2} \Bigl(\text{MAE}\bigl(\log[p_j^{\text{obs}} + \epsilon],\, \log[p_j^{\text{pred}} + \epsilon]\bigr) \;+\; \text{MAE}\bigl(\sigma_j^{\text{obs}},\, \sigma_j^{\text{pred}}\bigr) \Bigr), \tag{20.4}$$

where MAE denotes mean absolute error, and ϵ is a small positive constant to avoid $\log(\,0\,)$. The terms σ_j^{obs} and σ_j^{pred} refer to log-space standard deviations for the respective distributions of (or mapped from) end-states or functional events. By design, this objective penalizes deviations of both central tendencies and spread. A gradient-based algorithm (e.g., Adam [89]) then iteratively refines $\{\mu_i, \sigma_i\}$, using automatic differentiation with respect to $\mathcal{L}$.

20.2.3 Results & Discussion

The parameter estimation recovered target distributions and corresponding end-state frequencies with near-accurate fidelity, indicating that the method is capable of approximating underlying probabilities from limited inputs. The predicted end-state frequency estimates are plotted in Figure 20.5.

Specifically, end-state frequencies estimated under the constrained optimization process diverged from reported references by small margins: on average, the mean values were recovered with an error of about $(1.08 \pm 0.96)\%$, the 5th percentile with $(4.39 \pm 7.09)\%$, and the 95th percentile with $(3.82 \pm 5.91)\%$. Such deviations suggest that the overall approach captures the central tendencies of event probabilities reasonably well, while still exhibiting moderate scatter in both lower and upper distribution tails. Recurrence of larger discrepancies in selected events (e.g., certain fire detection or suppression paths) emphasizes the known difficulty of accurately modeling rare failure or success probabilities—particularly when the choice of distribution (e.g., log-normal) imposes strong structural assumptions on the shapes of these probability curves.

Despite these promising quantitative metrics, two issues warrant discussion. First, although end-state frequencies are reproduced within small mean errors, there is a real possibility of overfitting to the specified targets. The optimization-driven procedure can finely tune parameters to minimize a chosen loss function; however, doing so may lead to calibrated event probabilities that reflect artifacts of the objective rather than a physically robust representation. This risk is heightened when dealing with low-probability events (e.g., a rare liquid metal fire condition combined with other system failures)—situations that often exhibit limited empirical data.

Second, the truncation and bounds on the log-normal parameterization, while necessary for numerical stability, can restrict the feasible solution space in unintended ways. Large or extremely small event probabilities, particularly in tail regions, must fit within these truncated distributions. If the true system behavior lies outside the assumed bounds, the resulting estimates may systematically under- or overestimate important tail events. This possibility is underscored by the modest underestimation observed at the 95th percentile for certain functional events in the demonstration.

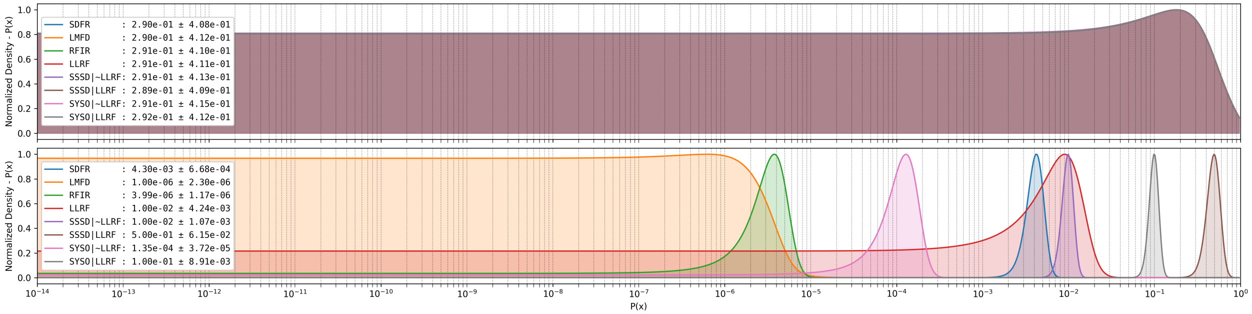


Figure 20.2: Initial vs Target Functional Event Probability Distributions

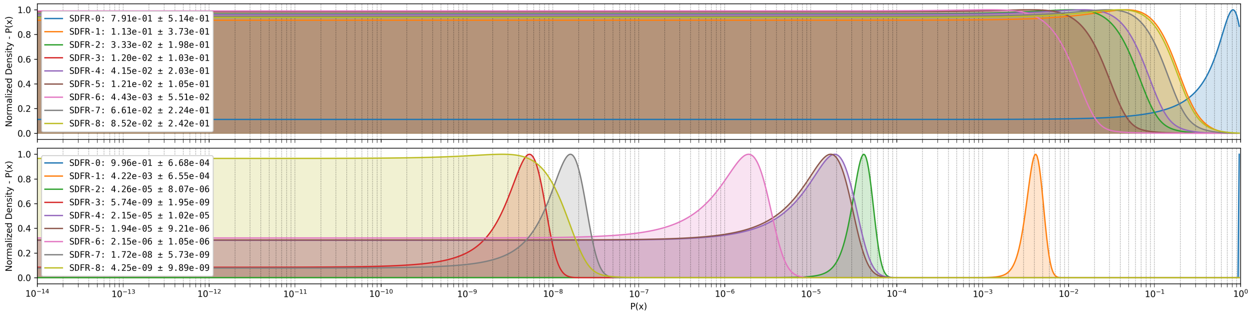


Figure 20.3: Initial vs Target End-State Frequency Distributions

Table 20.1: Estimated vs Target Functional Event Probabilities Summarized

Event	5th Percentile			Mean			95th Percentile		
	Estimated	Target	Error[1]	Estimated	Target	Error[1]	Estimated	Target	Error[1]
SDFR	3.29×10^{-3}	3.31×10^{-3}	-2.31×10^{-5}	4.25×10^{-3}	4.31×10^{-3}	-5.98×10^{-5}	5.38×10^{-3}	5.48×10^{-3}	-1.08×10^{-4}
LMFD	4.40×10^{-8}	4.30×10^{-8}	9.86×10^{-10}	1.01×10^{-6}	1.00×10^{-6}	7.14×10^{-10}	3.71×10^{-6}	3.71×10^{-6}	-5.10×10^{-9}
RFIR	2.50×10^{-6}	2.40×10^{-6}	9.29×10^{-8}	4.14×10^{-6}	4.01×10^{-6}	1.30×10^{-7}	6.33×10^{-6}	6.16×10^{-6}	1.72×10^{-7}
LLRF	4.74×10^{-3}	4.73×10^{-3}	1.95×10^{-5}	9.94×10^{-3}	1.00×10^{-2}	-6.37×10^{-5}	1.78×10^{-2}	1.80×10^{-2}	-2.26×10^{-4}
SSSD | $\overline{\text{LLRF}}$[2]	8.73×10^{-3}	8.36×10^{-3}	3.69×10^{-4}	1.02×10^{-2}	1.01×10^{-2}	5.38×10^{-5}	1.16×10^{-2}	1.20×10^{-2}	-3.32×10^{-4}
SSSD | LLRF	4.94×10^{-1}	4.07×10^{-1}	8.75×10^{-2}	4.95×10^{-1}	5.01×10^{-1}	-5.49×10^{-3}	4.97×10^{-1}	6.08×10^{-1}	-1.12×10^{-1}
SYSO | $\overline{\text{LLRF}}$	8.25×10^{-5}	8.34×10^{-5}	-9.55×10^{-7}	1.37×10^{-4}	1.36×10^{-4}	8.32×10^{-7}	2.08×10^{-4}	2.04×10^{-4}	3.86×10^{-6}
SYSO | LLRF	8.55×10^{-2}	8.61×10^{-2}	-6.50×10^{-4}	9.90×10^{-2}	1.01×10^{-1}	-1.12×10^{-3}	1.15×10^{-1}	1.16×10^{-1}	-1.67×10^{-3}

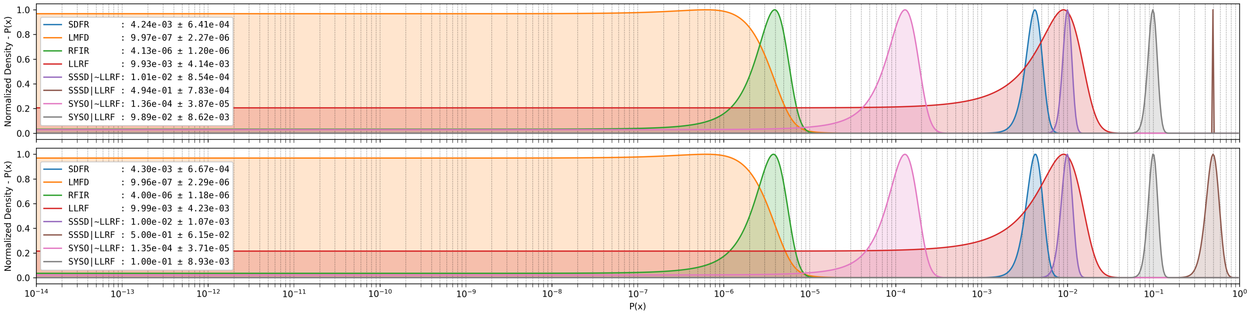


Figure 20.4: Estimated vs Target Functional Event Probability Distributions

[1] [Estimated - Target], negative values represent underestimates.

[2] $A \mid \overline{B}$: event A conditional on the non-occurrence of event B.

Table 20.2: Estimated vs Target End-State Frequencies Summarized

Event	5th Percentile			Mean			95th Percentile		
	Estimated	Target	Error[3]	Estimated	Target	Error[1]	Estimated	Target	Error[1]
SDFR-0[4]	9.96×10^{-1}	9.96×10^{-1}	1.09×10^{-4}	9.97×10^{-1}	9.97×10^{-1}	5.99×10^{-5}	9.98×10^{-1}	9.98×10^{-1}	2.31×10^{-5}
SDFR-1	3.22×10^{-3}	3.24×10^{-3}	-2.27×10^{-5}	4.17×10^{-3}	4.23×10^{-3}	-5.87×10^{-5}	5.28×10^{-3}	5.38×10^{-3}	-1.07×10^{-4}
SDFR-2	3.13×10^{-5}	3.08×10^{-5}	5.82×10^{-7}	4.23×10^{-5}	4.27×10^{-5}	-3.57×10^{-7}	5.53×10^{-5}	5.70×10^{-5}	-1.71×10^{-6}
SDFR-3	3.16×10^{-9}	3.16×10^{-9}	-7.37×10^{-12}	5.74×10^{-9}	5.75×10^{-9}	-1.32×10^{-11}	9.34×10^{-9}	9.36×10^{-9}	-2.40×10^{-11}
SDFR-4	9.63×10^{-6}	9.23×10^{-6}	4.08×10^{-7}	2.14×10^{-5}	2.16×10^{-5}	-1.91×10^{-7}	3.95×10^{-5}	4.08×10^{-5}	-1.34×10^{-6}
SDFR-5	8.48×10^{-6}	8.32×10^{-6}	1.67×10^{-7}	1.89×10^{-5}	1.95×10^{-5}	-5.88×10^{-7}	3.48×10^{-5}	3.68×10^{-5}	-2.05×10^{-6}
SDFR-6	9.11×10^{-7}	9.08×10^{-7}	3.36×10^{-9}	2.07×10^{-6}	2.16×10^{-6}	-9.11×10^{-8}	3.86×10^{-6}	4.14×10^{-6}	-2.87×10^{-7}
SDFR-7	9.85×10^{-9}	9.59×10^{-9}	2.65×10^{-10}	1.76×10^{-8}	1.73×10^{-8}	3.09×10^{-10}	2.83×10^{-8}	2.80×10^{-8}	3.18×10^{-10}
SDFR-8	1.82×10^{-10}	1.81×10^{-10}	1.86×10^{-12}	4.19×10^{-9}	4.25×10^{-9}	-5.51×10^{-11}	1.57×10^{-8}	1.59×10^{-8}	-2.54×10^{-10}

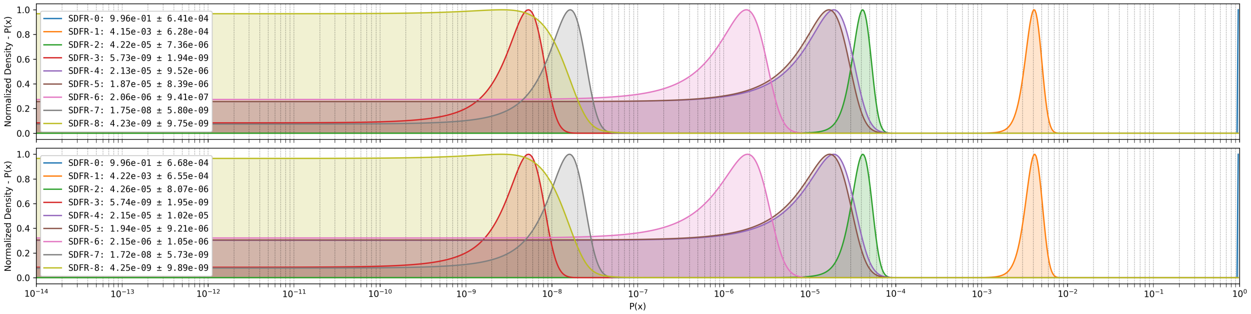


Figure 20.5: Estimated vs Target End-State Frequency Distributions

[3] [Estimated - Target], negative values represent underestimates.

[4] The likelihood of no SDFR. Computed by subtracting all end-state frequencies from the total probability.

Chapter 21

Conclusion and Future Work

The preceding chapters have advanced a hardware–accelerated Monte–Carlo framework that re-imagines quantitative risk assessment from first principles: from the unification of event trees and fault trees into a single *probabilistic directed acyclic graph* (PDAG), through a bit–parallel execution model that saturates commodity GPUs, to a suite of statistical diagnostics that certify estimator quality in real time. The purpose of this chapter is threefold:

1. Recapitulate the dissertation's principal contributions and empirical findings;
2. Assess the methodological limitations that remain; and
3. Chart a research agenda that extends the present work toward an end-to–end, industry–grade risk-analytics platform.

21.1 Summary of Contributions

1. Holistic Modeling via Unified PDAGs. We demonstrated that the full spectrum of PRA artifacts—hundreds of event trees and thousands of fault trees—can be embedded in a single computation graph (Chapter 4.1). The formulation preserves logical semantics while eliminating bookkeeping boundaries between top-down and forward-chaining analyses.

2. Hardware-Native Logic Gates. Threshold, cardinality, and related voting constructs were formalized as first-class citizens whose bit-packed kernels provably replicate AND/OR expansions (Chapter 18.1). The result is an exponential reduction in graph size—up to $\mathcal{O}(2^n/\sqrt{n})$ for worst–case fan-ins—without sacrificing estimator unbiasedness or variance.

3. Knowledge Compilation Re-examined. An eight-stage transformation pipeline was designed for Monte-Carlo rather than exact inference (Chapter 18.3). By relaxing certain normal-form constraints and introducing linear-time data structures, compile times fell from 1 080 s to 5.8 s (186×) on a representative ATLEAST(6/32) gate.

4. Bit-Parallel Execution Model. Layers of the PDAG are evaluated in lock-step by SYCL kernels that process 64 Bernoulli trials per machine word (Parts III and IV). Benchmarks on the 43-model Aralia dataset confirm *fully saturated* memory bandwidth and arithmetic pipelines, with sub-percent relative error attained in < 5 s for graphs containing $\sim 10^3$ unique events—even on entry-level consumer GPUs.

5. Rigorous Convergence Diagnostics. A composite stopping rule (Chapter 14) fuses frequentist, Bayesian, and information-theoretic criteria, ensuring that every estimate meets a user-specified margin of error while avoiding wasteful oversampling.

6. Domain-Specific Extensions. Common-cause failure modeling, first-order importance measures, and a practical importance-sampling scheme for rare events were integrated without altering kernel code paths (Part IV). Each extension carries formal proofs of unbiasedness and variance preservation.

7. Open-Source Reference Implementation. The entire framework is released under a permissive license, thereby lowering the barrier to third-party validation, audit, and reuse.

21.2 Empirical Findings

- **Graph Compression.** Knowledge-compilation levels C_2–C_4 already yield median gate-count reductions of 30 %; final compiled graphs shrink by a median factor of 1.3× while preserving logical fidelity.

- **Throughput and Scaling.** On an NVIDIA RTX 3060 Laptop GPU the solver sustains $\approx 5.8 \times 10^9$ Boolean operations *per second*, a 200–400× improvement over single-threaded CPU baselines.

- **Estimator Quality.** Across all Aralia models the empirical coverage of 99 % confidence intervals exceeds 98.6 %, confirming that the composite stopping rule is neither overly conservative nor prone to premature termination.

- **Rare-Event Performance.** Importance sampling lowers variance by up to two orders of magnitude for top-event probabilities below 10^{-6}, but probabilities in the 10^{-8} regime still require prohibitively many samples on commodity hardware.

21.3 Limitations

Despite substantial progress, several constraints delimit the present framework's applicability:

Ultra-Rare Events For probabilities $< 10^{-8}$ the sample size needed to achieve sub-percent precision remains computationally expensive even with importance sampling.

Hardware Dependence Peak performance assumes wide SIMD lanes and high memory bandwidth; CPU-only deployments incur 1–2 orders of magnitude slower runtimes.

Static Graphs The methodology targets *static* Boolean logic. Time-dependent degrading components, repair policies, or scenario trees with feedback loops are out of scope.

Correlation Modeling While common-cause failures are supported, broader classes of statistical dependence (e.g. epistemic-aleatory mixtures, copulas) have not yet been integrated into the bit-packed sampler.

Verification and Validation Industry-scale PRA models—such as the Generic Pressurized Water Reactor benchmark—have yet to undergo full-fidelity replication within mcSCRAM.

21.4 Future Work

The limitations above point to a fertile research agenda:

21.4.1 Variance-Reduction Beyond Importance Sampling

Importance sampling remains our primary defense against estimator blow-up in the tail, yet it is notoriously brittle: likelihood–ratio weights can explode, self-normalized estimators suffer when weight variance is high, and GPU reduction of double-precision weights stresses memory bandwidth. A richer palette of variance-reduction strategies promises more reliable convergence across the entire probability spectrum:

1. **Stratified and Latin–Hypercube Sampling (LHS).** By forcing each stratum of the basic-event probability simplex to be represented exactly once per iteration, LHS lowers variance for smoothly varying response surfaces by up to an order of magnitude and reuses the existing bit-packing kernels with only a deterministic permutation of PRNG counters.

2. **Antithetic Variates.** For coherent fault trees the mapping ($X \mapsto \neg X$) yields negatively correlated pairs. Launching antithetic trajectories doubles work per thread but can halve variance at negligible implementation cost.

3. **Control Variates and Rao–Blackwellization.** Gate-level surrogate models—e.g. logistic or polynomial fits updated on-the-fly—offer closed-form expectations that, when subtracted, act as low-cost control variates. The correction occurs during the tally reduction and does not perturb the sampling kernel.

4. **Multi-Level / Multi-Fidelity Monte Carlo.** Evaluate a hierarchy of approximations—coarse cut-set truncations, reduced bit-width kernels, or early-exit gate evaluations—and combine them with optimally chosen coefficients to achieve the canonical $\mathcal{O}(\epsilon^{-2})$ complexity with markedly smaller constants.

5. **Subset Simulation and Splitting.** Particularly effective for top events with probabilities below 10^{-9}, these algorithms recurse on nested conditional events rather than rely on exponentially small likelihood ratios, thereby avoiding the weight-degeneracy problem of extreme importance sampling.

All five techniques map cleanly onto modern accelerators: strata and antithetic pairs correspond to extra kernel dimensions; control-variate adjustment is a post-kernel reduction; multi-level sampling reuses the compiled graph with altered fan-in; and subset simulation requires only a lightweight, GPU-resident event queue. Integrating them would extend mcSCRAM's reach to risk scenarios where the current importance-sampling prototype stalls.

21.4.2 Discrete-Event Simulation Engine

The static Boolean abstraction is well suited to snapshot risk metrics, yet many industries require explicit reasoning about *when* failures occur, how long repairs take, and how operator interventions alter the hazard landscape. A discrete–event simulation (DES) back-end would elevate mcSCRAM from a purely probabilistic calculator to a full life-cycle risk engine.

Event Calendar. A lock-free, GPU–resident event calendar—realized as a ring buffer or bucketed heap—would schedule state transitions while maintaining sub-microsecond synchronization overhead.

Hybrid Evaluation. Boolean layers continue to leverage the bit-parallel kernels; the DES layer merely dictates *which* subset of nodes fire at each time stamp. This preserves the compiled-graph investment while adding a temporal dimension.

Variable-Time Stepping. Techniques such as *subset simulation* dovetail naturally with DES: busy early phases are simulated densely, whereas long quiescent intervals can be collapsed into single jumps, reducing the event calendar's size without biasing results.

Validation Path. An incremental roadmap begins with mission-time availability models—exponential failure and repair—where analytical benchmarks exist, then extends to phased-mission and standby redundancy logic.

Scalability Considerations. Typical DES workloads involve 10^6–10^7 events, orders of magnitude fewer than Monte-Carlo samples, making them ideal for CPU–GPU pipelining: the CPU advances the calendar while the GPU processes batched Boolean evaluations in the background.

A DES extension therefore promises accurate time-dependent risk metrics at interactive latencies, closing the gap between snapshot PRA and real-time condition monitoring.

21.4.3 Correlated Uncertainty Models

Incorporating copula-based or Bayesian network representations of component dependence will broaden the framework's realism. A first step is to embed vectorized random-field generators whose correlation structure aligns with bit-packed memory layouts.

21.4.4 Gradient-Based Parameter Learning

Chapter 16 hinted at automatic differentiation of Boolean circuits. A full autodiff backend would unlock gradient descent over leaf probabilities or gate parameters, enabling data-driven calibration and model fitting (Part V).

21.4.5 Real-Time and Streaming Probabilistic Risk Assessment

By streaming input data and maintaining incremental tallies, the solver could support rolling updates with statistical guarantees—a capability relevant to plant monitoring and digital twin applications.

21.4.6 Cross-Industry Validation

Applicability to aerospace, autonomous driving, and chemical processing systems should be demonstrated on open benchmark suites, thereby establishing mcSCRAM's generality.

Closing Remarks

This dissertation has shown that a principled blend of probabilistic circuits, high-throughput hardware, and modern statistical diagnostics redefines the computational frontier of quantitative risk assessment. By evaluating *entire* PRA models as unified computation graphs, mcSCRAM closes the gap between theoretical rigour and practical turnaround time—bringing risk-informed decision making within reach of domains where exhaustive symbolic methods have long since faltered. The roadmap laid out above offers a vision for completing the transition from point solution to pervasive, real-time risk analytics. The author invites the community to build upon this foundation and advance the state of the art in stochastic dependability modeling.

References

[1] Nuclear Energy Agency. *International Common-cause Failure Data Exchange (ICDE) Project*. URL: https://www.oecd-nea.org/jcms/pl_25090/international-common-cause-failure-data-exchange-icde-project (visited on 04/02/2025).

[2] Alan Agresti and Brent A. Coull. "Approximate Is Better Than "Exact" for Interval Estimation of Binomial Proportions." In: *The American Statistician* 52.2 (1998), pp. 119–126. DOI: 10.1080/00031305.1998.10480550.

[3] Giovanni Luca Amicucci, Fabio Pera, and Andrea Tonti. "Reliability analysis of nuclear instrumentation and control systems." en. In: *Instrumentation and Control Systems for Nuclear Power Plants*. Elsevier, 2023, pp. 887–956. ISBN: 978-0-08-102836-0. DOI: 10.1016/B978-0-08-102836-0.00010-2. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780081028360000102 (visited on 05/16/2025).

[4] Egemen Aras, Arjun Earthperson, and Mihai Diaconeasa. *Generic Pressurized Water Reactor (PWR) Open-PSA Model*. Nov. 2024. DOI: 10.5281/ZENODO.14070453. URL: https://zenodo.org/doi/10.5281/zenodo.14070453 (visited on 04/28/2025).

[5] Egemen Aras, Arjun Earthperson, and Mihai Diaconeasa. *Generic Pressurized Water Reactor (PWR) SAPHSOLVE Model*. Language: en. Oct. 2024. DOI: 10.5281/ZENODO.13996959. URL: https://zenodo.org/doi/10.5281/zenodo.13996959 (visited on 04/28/2025).

[6] Egemen Aras, Arjun Earthperson, and Mihai Diaconeasa. *Synthetic Open-PSA Models*. Language: en. Oct. 2024. DOI: 10.5281/ZENODO.13996370. URL: https://zenodo.org/doi/10.5281/zenodo.13996370 (visited on 04/28/2025).

[7] Egemen Aras, Arjun Earthperson, and Mihai Diaconeasa. *Synthetic SAPHSOLVE Models*. Language: en. Oct. 2024. DOI: 10.5281/ZENODO.13996735. URL: https://zenodo.org/doi/10.5281/zenodo.13996735 (visited on 04/28/2025).

[8] Egemen Aras et al. "Enhancing the SAPHIRE Solve Engine: Initial Progress and Efforts." en. In: *Advanced Reactor Safety (ARS)*. Las Vegas, NV: American Nuclear Society, 2024, pp. 542–551. ISBN: 978-0-89448-215-1. DOI: 10.13182/T130-43361. URL: https://www.ans.org/pubs/proceedings/article-56464/ (visited on 04/28/2025).

[9] Egemen Aras et al. "Introducing OpenPRA's Quantification Engine: Exploring Capabilities, Recognizing Limitations, and Charting the Path to Enhancement." en. In: *Advanced Reactor Safety (ARS)*. Las Vegas, NV: American Nuclear Society, 2024, pp. 552–561. ISBN: 978-0-89448-215-1. DOI: 10.13182/T130-43362. URL: https://www.ans.org/pubs/proceedings/article-56465/ (visited on 04/28/2025).

[10] Egemen Aras et al. "Method of Developing a SCRAM Parallel Engine for Efficient Quantification of Probabilistic Risk Assessment Models." en. In: *18th International Probabilistic Safety Assessment and Analysis (PSA 2023)*. Knoxville, TN: American Nuclear Society, 2023, pp. 134–140. ISBN: 978-0-89448-792-7. DOI: 10.13182/PSA23-41054. URL: https://www.ans.org/pubs/proceedings/article-53973/ (visited on 04/28/2025).

[11] Egemen Aras et al. "Methodology and Demonstration for Performance Analysis of a Probabilistic Risk Assessment Quantification Engine: SCRAM." en. In: *18th International Probabilistic Safety Assessment and Analysis (PSA 2023)*. Knoxville, TN: American Nuclear Society, 2023, pp. 452–459. ISBN: 978-0-89448-792-7. DOI: 10.13182/PSA23-41053. URL: https://www.ans.org/pubs/proceedings/article-54005/ (visited on 04/28/2025).

[12] Egemen Aras et al. *openpra-org/synthetic-models: Synthetically generated PRA models*. Version v0.1.0. May 2025. DOI: 10.5281/zenodo.15320670. URL: https://doi.org/10.5281/zenodo.15320670 (visited on 05/13/2025).

[13] Egemen M Aras et al. "Facilitating PRA Model Accessibility: Model Converter Utility from SAPHIRE to Open-PSA." en. In: *19th International Probabilistic Safety Assessment and Analysis (PSA 2025)*. Chicago, IL: American Nuclear Society, 2025.

[14] Egemen M Aras et al. "Synthetical Model Generator for Probabilistic Risk Assessment Tools: Enhancing Testing, Verifying and Learning." en. In: *19th International Probabilistic Safety Assessment and Analysis (PSA 2025)*. Chicago, IL: American Nuclear Society, 2025.

[15] Egemen M. Aras, Arjun Earthperson, and Mihai A. Diaconeasa. "A Systematic Diagnostics and Enhancement Framework for Advancing Probabilistic Risk Assessment Tools." en. In: *Nuclear Technology* (Aug. 2025), pp. 1–18. ISSN: 0029-5450, 1943-7471. DOI: `10.1080/00295450.2025.2511510`. URL: `https://www.tandfonline.com/doi/full/10.1080/00295450.2025.2511510` (visited on 11/04/2025).

[16] Egemen M. Aras et al. "Benchmark Study of XFTA and SCRAM Fault Tree Solvers Using Synthetically Generated Fault Trees Models." In: *Volume 9: Mechanics of Solids, Structures, and Fluids; Micro- and Nano-Systems Engineering and Packaging; Safety Engineering, Risk, and Reliability Analysis; Research Posters*. Columbus, Ohio, USA: American Society of Mechanical Engineers, Oct. 2022, V009T14A016. ISBN: 978-0-7918-8671-7. DOI: `10.1115/IMECE2022-95783`. URL: `https://asmedigitalcollection.asme.org/IMECE/proceedings/IMECE2022/86717/V009T14A016/1157443` (visited on 02/15/2023).

[17] Egemen Mutlu Aras. "Enhancement Methodology for Probabilistic Risk Assessment Tools through Diagnostics, Optimization, and Parallel Computing." English. PhD. Raleigh, North Carolina: North Carolina State University, 2024. URL: `https://catalog.lib.ncsu.edu/catalog/NCSU5937498`.

[18] Anis Baklouti et al. "Free and open source fault tree analysis tools survey." In: *2017 Annual IEEE International Systems Conference (SysCon)*. Montreal, QC, Canada: IEEE,

Apr. 2017, pp. 1–8. ISBN: 978-1-5090-4623-2. DOI: 10.1109/SYSCON.2017.7934794. URL: http://ieeexplore.ieee.org/document/7934794/ (visited on 04/29/2025).

[19] Akram Batikh, Mihai Diaconeasa, and OpenPRA. *Multi Hazard PRA Model Generator*. Oct. 2023. DOI: 10.5281/ZENODO.15298045. URL: https://zenodo.org/doi/10.5281/zenodo.15298045 (visited on 04/28/2025).

[20] Patrick Billingsley. *Probability and Measure*. 3rd. Wiley Series in Probability and Mathematical Statistics. New York: John Wiley & Sons, 1995, p. 593. ISBN: 0471007102. DOI: 10.1002/9781118341919.

[21] Craig Boutilier et al. *Context-Specific Independence in Bayesian Networks*. Version Number: 1. 2013. DOI: 10.48550/ARXIV.1302.3562. URL: https://arxiv.org/abs/1302.3562 (visited on 05/14/2025).

[22] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. “Interval Estimation for a Binomial Proportion.” In: *Statistical Science* 16.2 (2001), pp. 101–133. DOI: 10.1214/ss/1009213286.

[23] *CAFTA Software Technology PackageV 10 - Phoenix Architect*. https://polestartechnicalservices.com/cafta-software/. [Online; accessed 2023-03-20]. URL: https://polestartechnicalser
com/cafta-software/.

[24] Yoon Chang. *Experimental Breeder Reactor II (EBR-II) Level 1 Probabilistic Risk Assessment*. English. Tech. rep. ANL-NSE-2. Argonne National Lab. (ANL), Argonne, IL (United States), Oct. 2018. DOI: 10.2172/1483951. URL: https://www.osti.gov/biblio/1483951-experimental-breeder-reactor-ii-ebr-ii-level-probabilistic-risk-assessment (visited on 09/09/2021).

[25] *Computerized Fault Tree Analysis: TREEL and MICSUP*. en. Tech. rep. Section: Technical Reports. URL: https://apps.dtic.mil/sti/citations/ADA010146 (visited on 11/20/2022).

[26] A. Darwiche and P. Marquis. "A Knowledge Compilation Map." In: *Journal of Artificial Intelligence Research* 17 (Sept. 2002), pp. 229–264. ISSN: 1076-9757. DOI: 10.1613/jair.989. URL: https://jair.org/index.php/jair/article/view/10311 (visited on 05/07/2025).

[27] Mihai A Diaconeasa and Ali Mosleh. "Branching rules and quantification based on human behavior in the ADS-IDAC dynamic PRA platform." en. In: *Proceedings of the European Society for Reliability Annual Meeting*. Trondheim, Norway, 2018, p. 7.

[28] Mihai A Diaconeasa and Ali Mosleh. "The ADS-IDAC Dynamic PSA Platform with Dynamically Linked System Fault Trees." en. In: *American Nuclear Society Probabilistic Safety Assessment*. Pittsburgh, PA, 2017, p. 7.

[29] Arjun Earthperson. *Inverse Canopy*. Oct. 2024. URL: https://github.com/a-earthperson/inverse-canopy (visited on 10/01/2024).

[30] Arjun Earthperson. *Monte Carlo SCRAM*. Apr. 2025. URL: https://github.com/a-earthperson/mcSCRAM (visited on 07/01/2025).

[31] Arjun Earthperson, Mihai Diaconeasa, and OpenPRA. *Canopy Benchmarks*. Mar. 2025. DOI: 10.5281/ZENODO.15298728. URL: https://zenodo.org/doi/10.5281/zenodo.15298728 (visited on 04/28/2025).

[32] Arjun Earthperson, Mihai Diaconeasa, and OpenPRA. *pracciolini - PRA Model Translator*. July 2024. DOI: 10.5281/ZENODO.15298515. URL: https://zenodo.org/doi/10.5281/zenodo.15298515 (visited on 04/28/2025).

[33] Arjun Earthperson et al. *Benchmarks Post-processing Analysis & Supplementary Notes - An Open Source, Parallel, and Distributed Web-Based Probabilistic Risk Assessment Platform to Support Real Time Nuclear Power Plant Risk-Informed Operational Decisions*. May 2025. DOI: 10.5281/ZENODO.15320401. URL: https://zenodo.org/doi/10.5281/zenodo.15320401 (visited on 05/19/2025).

[34] Arjun Earthperson et al. *Dataset: Benchmarking SAPHIRE, SCRAM and XFTA using Synthetically Generated Fault Trees with Common Cause Failures*. en. Mar. 2023. DOI: 10.5281/ZENODO.7706615. URL: https://zenodo.org/doi/10.5281/zenodo.7706615 (visited on 04/28/2025).

[35] Arjun Earthperson et al. *Generic OpenPSA Models: The Aralia Fault Tree Dataset*. 2021. DOI: 10.5281/ZENODO.15293416. URL: https://zenodo.org/doi/10.5281/zenodo.15293416 (visited on 04/28/2025).

[36] Arjun Earthperson et al. "Introducing OpenPRA: A Web-Based Framework for Collaborative Probabilistic Risk Assessment." In: *Volume 13: Research Posters; Safety Engineering, Risk and Reliability Analysis*. New Orleans, Louisiana, USA: American Society of Mechanical Engineers, Oct. 2023, V013T15A014. ISBN: 978-0-7918-8770-7. DOI: 10.1115/IMECE2023-111708. URL: https://asmedigitalcollection.asme.org/IMECE/proceedings/IMECE2023/87707/V013T15A014/1196307 (visited on 04/02/2024).

[37] Arjun Earthperson et al. *PRA Model Benchmark Tools*. Apr. 2025. DOI: 10.5281/ZENODO.15297777. URL: https://zenodo.org/doi/10.5281/zenodo.15297777 (visited on 04/28/2025).

[38] Arjun Earthperson et al. *PRA Model Generator*. June 2022. DOI: 10.5281/ZENODO.15297976. URL: https://zenodo.org/doi/10.5281/zenodo.15297976 (visited on 04/28/2025).

[39] Arjun Earthperson et al. *PRA Model Schema Definitions*. Oct. 2024. DOI: 10.5281/ZENODO.15298269. URL: https://zenodo.org/doi/10.5281/zenodo.15298269 (visited on 04/28/2025).

[40] Arjun Earthperson et al. "Probability Estimation using Monte Carlo Simulation of Boolean Logic on Hardware-Accelerated Platforms." en. In: *19th International Proba-*

bilistic Safety Assessment and Analysis (PSA 2025). Chicago, IL: American Nuclear Society, 2025.

[41] Arjun Earthperson et al. "Towards a Deep-Learning based Heuristic for Optimal Variable Ordering in Binary Decision Diagrams to Support Fault Tree Analysis." en. In: *Advanced Reactor Safety (ARS)*. Las Vegas, NV: American Nuclear Society, 2024, pp. 400–409. ISBN: 978-0-89448-215-1. DOI: `10.13182/T130-43397`. URL: `https://www.ans.org/pubs/proceedings/article-56450/` (visited on 04/28/2025).

[42] Asmaa Farag et al. "Evaluating PRA Tools for Accurate and Efficient Quantifications: A Follow-Up Benchmarking Study Including FTREX." en. In: *Advanced Reactor Safety (ARS)*. Las Vegas, NV: American Nuclear Society, 2024, pp. 573–582. ISBN: 978-0-89448-215-1. DOI: `10.13182/T130-43377`. URL: `https://www.ans.org/pubs/proceedings/article-56467/` (visited on 04/28/2025).

[43] Asmaa Farag et al. "Preliminary Benchmarking of SAPHSOLVE, XFTA, and SCRAM Using Synthetically Generated Fault Trees with Common Cause Failures." en. In: *18th International Probabilistic Safety Assessment and Analysis (PSA 2023)*. Knoxville, TN: American Nuclear Society, 2023, pp. 40–49. ISBN: 978-0-89448-792-7. DOI: `10.13182/PSA23-41031`. URL: `https://www.ans.org/pubs/proceedings/article-53964/` (visited on 04/28/2025).

[44] Asmaa Salem Farag. *Benchmarking Study of Probabilistic Risk Assessment Tools Using Synthetically Generated Fault Tree Models: SAPHSOLVE, XFTA, and SCRAM*. Raleigh, North Carolina, 2023.

[45] Asmaa Salem Amin Aly Farag. "Benchmarking Study of Probabilistic Risk Assessment Tools Using Synthetically Generated Fault Tree Models : SAPHSOLVE, XFTA, and SCRAM." English. MA thesis. Raleigh, North Carolina: North Carolina State University, 2023. URL: `https://catalog.lib.ncsu.edu/catalog/NCSU5630439`.

[46] *FTREX 2.0 User Manual*. https://www.epri.com/research/programs/061177/results/3002018234. [Online; accessed 2023-07-31]. URL: https://www.epri.com/research/programs/061177/results/3002018234.

[47] J.B. Fussell, E.B. Henry, and N.H. Marshall. *MOCUS: a computer program to obtain minimal sets from fault trees*. Tech. rep. ANCR–1156, 4267950. DOI: 10.2172/4267950. Aug. 1974, ANCR–1156, 4267950. URL: http://www.osti.gov/servlets/purl/4267950/.

[48] B. John Garrick. *Quantifying and Controlling Catastrophic Risks*. en. Academic Press, Oct. 2008. ISBN: 978-0-08-092345-1.

[49] Frank J. Groen, Carol Smidts, and Ali Mosleh. "QRAS-the quantitative risk assessment system." en. In: *Reliability Engineering & System Safety* 91.3 (Mar. 2006), pp. 292–304. ISSN: 09518320. DOI: 10.1016/j.ress.2005.01.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0951832005000463 (visited on 10/04/2022).

[50] Mostafa Hamza et al. "Model Exchange Methodology Between Probabilistic Risk Assessment Tools: SAPHIRE and CAFTA Case Study." en. In: *18th International Probabilistic Safety Assessment and Analysis (PSA 2023)*. Knoxville, TN: American Nuclear Society, 2023, pp. 150–158. ISBN: 978-0-89448-792-7. DOI: 10.13182/PSA23-41025. URL: https://www.ans.org/pubs/proceedings/article-53975/ (visited on 04/28/2025).

[51] Mostafa Hamza et al. *openpra-org/generic-mhtgr-model*. May 2025. DOI: 10.5281/ZENODO.15397904. URL: https://zenodo.org/doi/10.5281/zenodo.15397904 (visited on 05/13/2025).

[52] S H Han, T W Kim, and K S Jeong. "PC Workstation-Based Level 1 PRA Code Package KIRAP." In: (), p. 10.

[53] Sang Hoon Han, Tae Woon Kim, and Kun Joong Yoo. "Development of an integrated fault tree analysis computer code MODULE by modularization technique." In: *Reliability*

Engineering & System Safety 21.2 (Jan. 1988), pp. 145–154. ISSN: 09518320. DOI: 10.1016/0951-8320(88)90052-X.

[54] Stanley Kaplan and B. John Garrick. "On The Quantitative Definition of Risk." en. In: *Risk Analysis* 1.1 (Mar. 1981), pp. 11–27. ISSN: 0272-4332, 1539-6924. DOI: 10.1111/j.1539-6924.1981.tb01350.x. URL: https://onlinelibrary.wiley.com/doi/10.1111/j.1539-6924.1981.tb01350.x (visited on 10/16/2022).

[55] J.M. Koren and J. Gaertner. *CAFTA: A fault tree analysis tool designed for PSA*. Germany: Verl TUEV Rheinland, 1987. ISBN: 3-88585-417-1. URL: http://inis.iaea.org/search/search.aspx?orig_q=RN:20061750.

[56] Lawrence Livermore National Laboratory. "SIGPI:A USER'S MANUAL FOR FAST COMPUTATION OF THE PROBABILISTIC PERFORMANCE OF COMPLEX SYSTEMS." In: (1988), p. 73.

[57] David A. Levin and Yuval Peres. *Markov Chains and Mixing Times*. 2nd ed. Vol. 107. AMS Non-Series Monographs. Providence, Rhode Island: American Mathematical Society, 2017, p. 447. ISBN: 978-1-4704-2962-1. DOI: 10.1090/mbk/107. URL: https://doi.org/10.1090/mbk/107.

[58] Zhegang Ma and Kellie J Kvarfordt. *CCF Parameter Estimations, 2020 Update*. en. Tech. rep. INL/EXT-21-62940. U.S. NRC, Aug. 2022.

[59] Manorma. "RiskSpectrum: Emerging software for Nuclear Power Industry." In: *2010 1st International Nuclear & Renewable Energy Conference (INREC)*. [Online; accessed 2022-05-04]. IEEE. Amman, Jordan, Mar. 2010, pp. 1–6. ISBN: 978-1-4244-5213-2. DOI: 10.1109/INREC.2010.5462562. URL: http://ieeexplore.ieee.org/document/5462562/.

[60] OpenPRA Initiative. *OpenPRA Monorepo*. Mar. 2024. DOI: 10.5281/ZENODO.10891407. URL: https://zenodo.org/doi/10.5281/zenodo.10891407 (visited on 04/28/2025).

[61] Palisade/Lumivero. *@RISK | Best Risk Software in Excel*. en-US. URL: https://lumivero.com/products/at-risk/ (visited on 04/17/2025).

[62] Palisade/Lumivero. *Techniques and Tips - Palisade Knowledge Base*. URL: https://kb.palisade.com/index.php (visited on 04/17/2025).

[63] Priyanka Pandit, Arjun Earthperson, and Mihai A. Diaconeasa. "The Development, Implementation, and Application of a Probabilistic Risk Assessment Framework to Evaluate Supply Chain Shortages." en. In: *Logistics* 9.4 (Oct. 2025), p. 141. ISSN: 2305-6290. DOI: 10.3390/logistics9040141. URL: https://www.mdpi.com/2305-6290/9/4/141 (visited on 11/04/2025).

[64] Olzhas Rakhimov. *SCRAM*. 10.5281/zenodo.1146337. C++. original-date: 2014-03-21T01:19:49Z. Apr. 2022. URL: 10.5281/zenodo.1146337.

[65] Hasibul H Rasheeq. "[Working Title] Design and Implementation of a Distributed Queueing System for PRA Quantification." English. MA thesis. Raleigh, North Carolina: North Carolina State University, 2025.

[66] Hasibul H Rasheeq et al. "Automated OpenPSA Model Generation from Reliability Diagrams Using Agentic Retrieval-Augmented Generation: A Case Study on MHTGR." en. In: *19th International Probabilistic Safety Assessment and Analysis (PSA 2025)*. Chicago, IL: American Nuclear Society, 2025.

[67] Hasibul H Rasheeq et al. "Design and Implementation of a Distributed Queueing System for OpenPRA." en. In: *19th International Probabilistic Safety Assessment and Analysis (PSA 2025)*. Chicago, IL: American Nuclear Society, 2025.

[68] Rasmussen. *Reactor safety study. An assessment of accident risks in U. S. commercial nuclear power plants. Executive summary: main report. [PWR and BWR]*. English. Tech. rep. WASH-1400-MR; NUREG-75/014-MR. Nuclear Regulatory Commission, Washington, D.C. (USA), Oct. 1975. DOI: 10.2172/7134131. URL: https://www.osti.gov/biblio/7134131 (visited on 08/25/2021).

[69] Antoine B. Rauzy. *Probabilistic Safety Analysis with XFTA*. ALTARICA ASSOCIATION, 2020. ISBN: 978-82-692273-0-7.

[70] The B. John Garrick Institute for the Risk Sciences. *HCLA Hybrid Causal Logic Analyzer - Command Line*. en-US. URL: https://www.risksciences.ucla.edu/hcla.

[71] The B. John Garrick Institute for the Risk Sciences. *HCLA Hybrid Causal Logic Analyzer - Web*. en-US. URL: https://www.apps.risksciences.ucla.edu/hcla.

[72] Enno Ruijters and Mariëlle Stoelinga. "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools." In: *Computer Science Review* 15-16 (Feb. 2015), pp. 29–62. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2015.03.001. URL: https://www.sciencedirect.com/science/article/pii/S1574013715000027.

[73] K.D. Russell, M.B. Sattison, and D.M. Rasmuson. "Living PRAs made easier with IRRAS." In: *Reliability Engineering & System Safety* 30.1-3 (Jan. 1990), pp. 239–269. ISSN: 09518320. DOI: 10.1016/0951-8320(90)90097-7.

[74] Kenneth D. Russell and Dale M. Rasmuson. "Fault tree reduction and quantification—an overview of IRRAS algorithms." In: *Reliability Engineering & System Safety* 40.2 (Jan. 1993), pp. 149–164. ISSN: 09518320. DOI: 10.1016/0951-8320(93)90105-8.

[75] John K. Salmon et al. "Parallel Random Numbers: As Easy as 1, 2, 3." In: *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. Best Paper Award winner. ACM. Seattle, Washington, USA, 2011, pp. 1–12. DOI: 10.1145/2063384.2063405. URL: https://doi.org/10.1145/2063384.2063405.

[76] *SAPHIRE | Training Info*. https://saphire.inl.gov/#/. [Online; accessed 2022-08-09]. URL: https://saphire.inl.gov/#/.

[77] *SAPHIRE 8.0.9: Systems Analysis Programs for Hands-On Integrated Reliability Evaluations*. https://rsicc.ornl.gov/codes/psr/psr6/psr-608.html. [Online; accessed 2023-02-09]. July 2014. URL: https://rsicc.ornl.gov/codes/psr/psr6/psr-608.html.

[78] Curtis Lee Smith. *Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners: NASA/SP-2011-3421 Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners*. English. CreateSpace Independent Publishing Platform, Feb. 2012. ISBN: 978-1-4700-8054-9. URL: `https://ntrs.nasa.gov/api/citations/20120001369/downloads/20120001369.pdf`.

[79] *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8, Volume 1: Overview and Summary.* Tech. rep. NUREG/CR-7039. DOI: 10.2172/130641. Idaho Falls, ID: Idaho National Lab. (INL), 2011.

[80] George R. Terrell and David W. Scott. "Variable Kernel Density Estimation." In: *The Annals of Statistics* 20.3 (1992). Publisher: Institute of Mathematical Statistics, pp. 1236–1265. ISSN: 00905364. URL: `http://www.jstor.org/stable/2242011` (visited on 02/09/2024).

[81] W E Vesely and R E Narum. *PREP KITT, System Reliability by Fault Tree Analysis. PREP, Min Path Set and Min Cut Set for Fault Tree Analysis, Monte-Carlo Method. KITT, Component and System Reliability Information from Kinetic Fault Tree Theory.* Mar. 1997.

[82] W. E. Vesely and R. E. Narum. *Prep and Kitt: Computer Codes for the Automatic Evaluation of a Fault Tree*. en. U.S. Atomic Energy Commission, 1970.

[83] Donald Wakefield. *Example Application of RISKMAN®*. Tech. rep. Operational Risk and Performance Consulting Division, 300 Commerce Drive 200, Irvine, CA, 92602: ABS Consulting, Inc., Sept. 2004, p. 10. URL: `https://flightsafety.org/wp-content/uploads/2016/09/RISKMAN_application.pdf`.

[84] Mr Donald Wakefield et al. "RISKMAN®, Celebrating 20+ Years of Excellence!" In: (), p. 10.

[85] Chengdong Wang. “Hybrid Causal Logic Methodology for Risk Assessment.” en. PhD. University of Maryland, Nov. 2007. URL: `http://drum.lib.umd.edu/handle/1903/7729` (visited on 01/20/2019).

[86] Samuel Williams, Andrew Waterman, and David Patterson. “Roofline: An Insightful Visual Performance Model for Multicore Architectures.” In: *Communications of the ACM* 52.4 (2009), pp. 65–76. DOI: `10.1145/1498765.1498785`.

[87] Stephen Wood et al. “Advancing SAPHIRE: Transitioning from Legacy to State-of-Art Excellence.” en. In: *Advanced Reactor Safety (ARS)*. Las Vegas, NV: American Nuclear Society, 2024, pp. 532–541. ISBN: 978-0-89448-215-1. DOI: `10.13182/T130-43357`. URL: `https://www.ans.org/pubs/proceedings/article-56463/` (visited on 04/28/2025).

[88] Yican Wu. “Development of reliability and probabilistic safety assessment program RiskA.” en. In: *Annals of Nuclear Energy* 83 (Sept. 2015), pp. 316–321. ISSN: 03064549. DOI: `10.1016/j.anucene.2015.03.020`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S030645491500153X` (visited on 04/16/2025).

[89] Zijun Zhang. “Improved Adam Optimizer for Deep Neural Networks.” In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. Banff, AB, Canada: IEEE, June 2018, pp. 1–2. ISBN: 978-1-5386-2542-2. DOI: `10.1109/IWQoS.2018.8624183`. URL: `https://ieeexplore.ieee.org/document/8624183/` (visited on 02/09/2024).

[90] Zhenxu Zhou, Hao Nie, and Qin Zhang. “Design and Development of DeRisk: A Fault Tree Analysis Program Package.” en. In: *Volume 2: Plant Systems, Structures, Components, and Materials; Risk Assessments and Management*. London, England: American Society of Mechanical Engineers, July 2018, V002T14A008. ISBN: 978-0-7918-5144-9. DOI: `10.1115/ICONE26-81291`. URL: `https://asmedigitalcollection.asme.org/ICONE/proceedings/ICONE26/51449/London,%20England/273329` (visited on 02/11/2022).

APPENDIX

Appendix A

Revised Aralia Convergence Traces

In this chapter, we plot the the Monte–Carlo convergence experiments on the *Aralia* fault–tree data set (Section 6.2.1). The revised study targeted a relative margin of error of 0.1%, that is, $\varepsilon = 10^{-3}\,\hat{p}$, at a 99 % confidence level with a wall–clock time limit of 60 seconds per model. The following figures collate the updated convergence traces for all 43 fault trees.

- the sample mean estimate (solid colored line),
- the empirical 90 % and 99 % confidence bands (shaded regions),
- where available, the reference "true" probability (black dashed), computed using SCRAM's Binary Decision Diagram (BDD) implementation,
- when plotting is feasible, the reference Rare-Event Approximation (REA) and Min Cut Upper Bound (MCUB) probabilities.

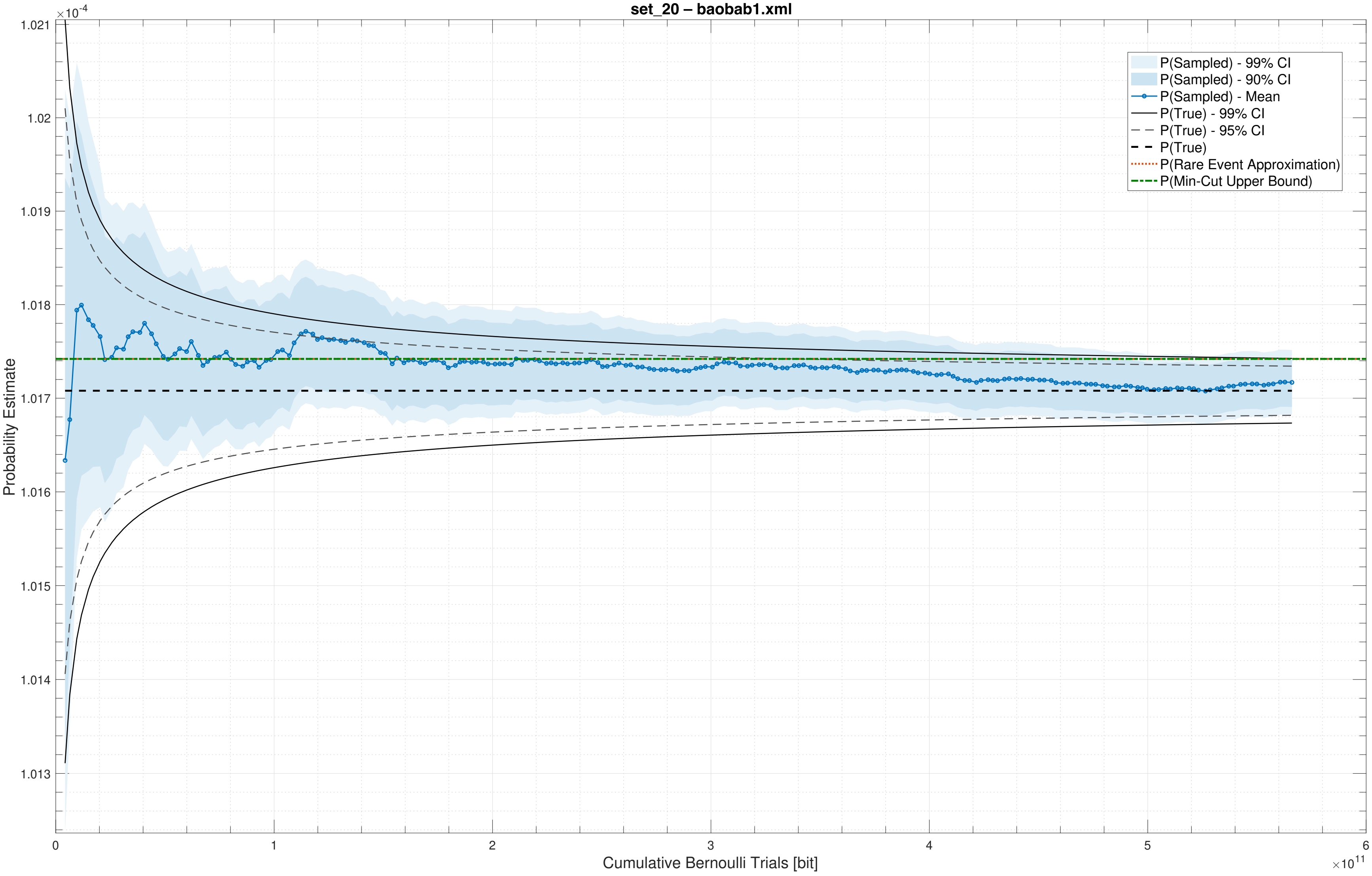


Figure A.1: Convergence trace for Aralia fault tree 1, 60 second timeout

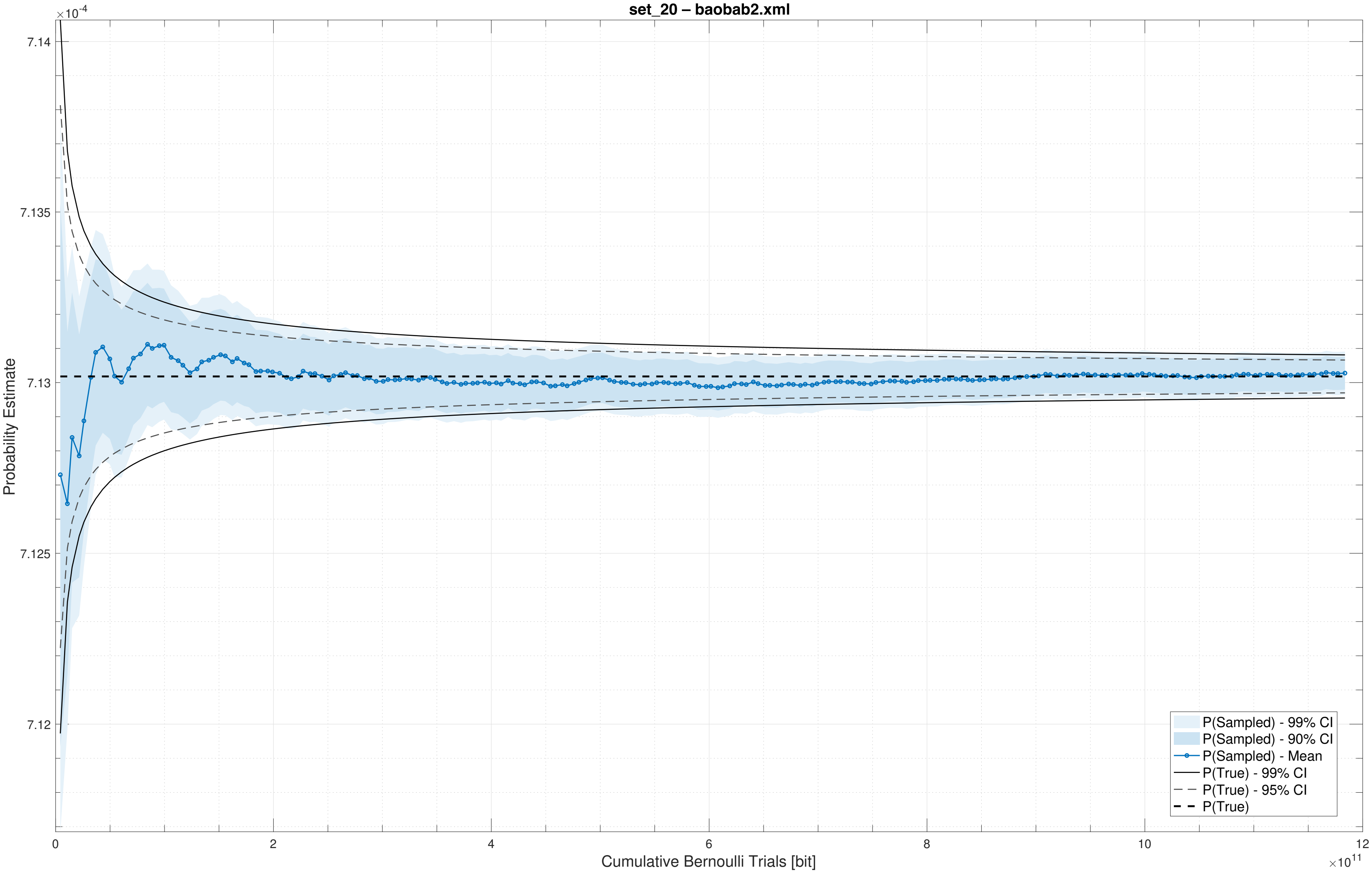


Figure A.2: Convergence trace for Aralia fault tree 2, 60 second timeout

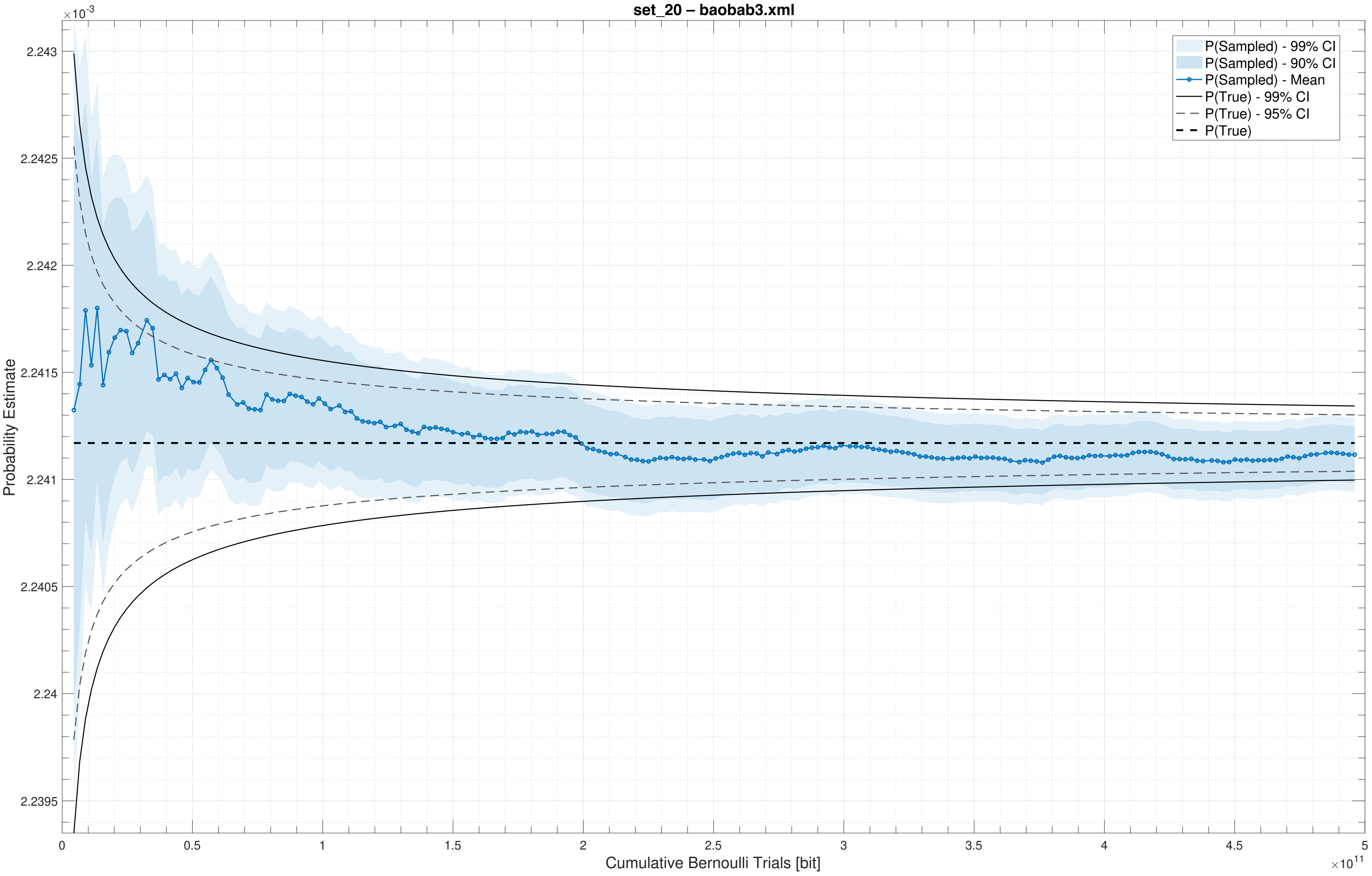


Figure A.3: Convergence trace for Aralia fault tree 3, 60 second timeout

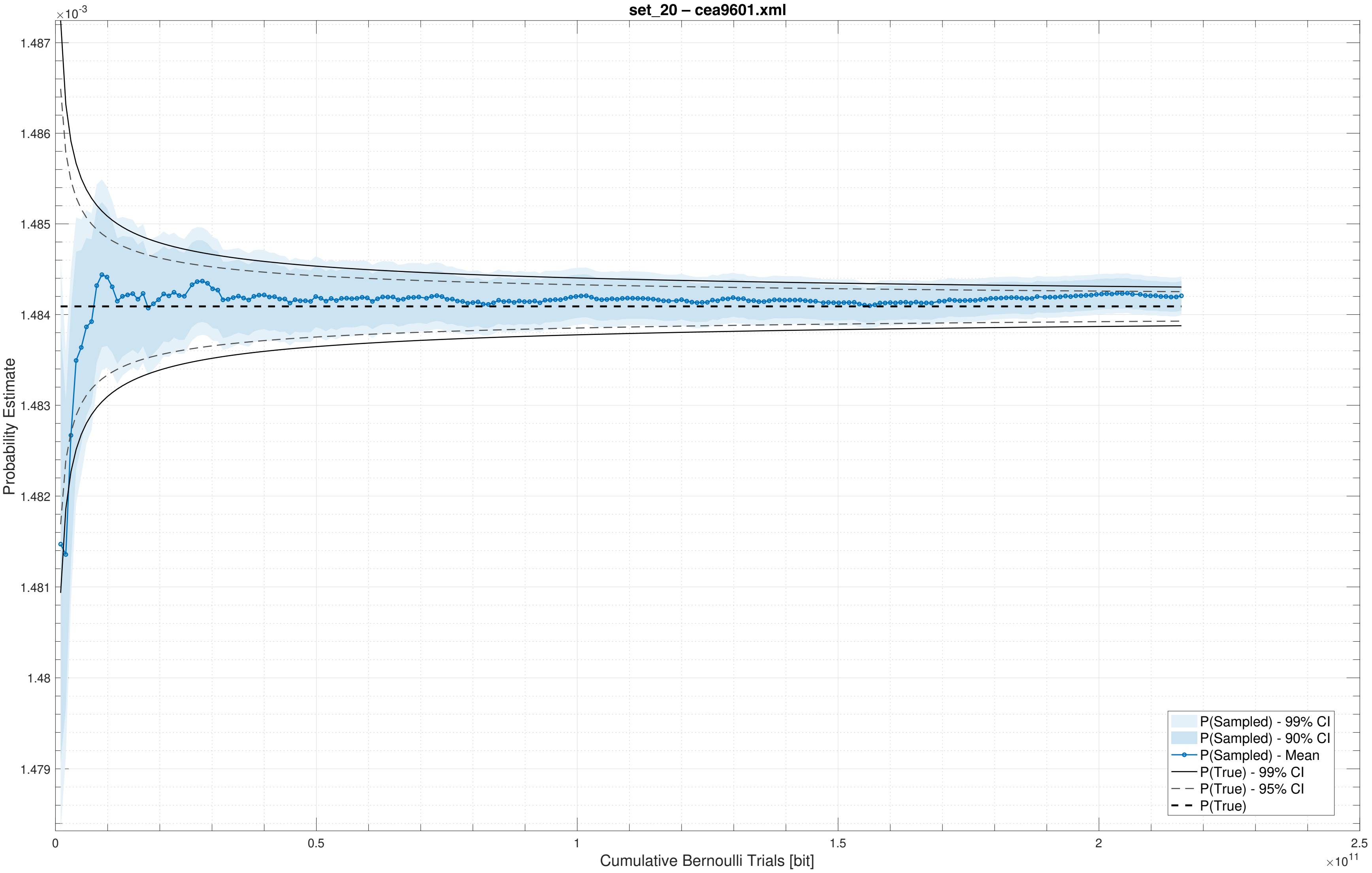


Figure A.4: Convergence trace for Aralia fault tree 4, 60 second timeout

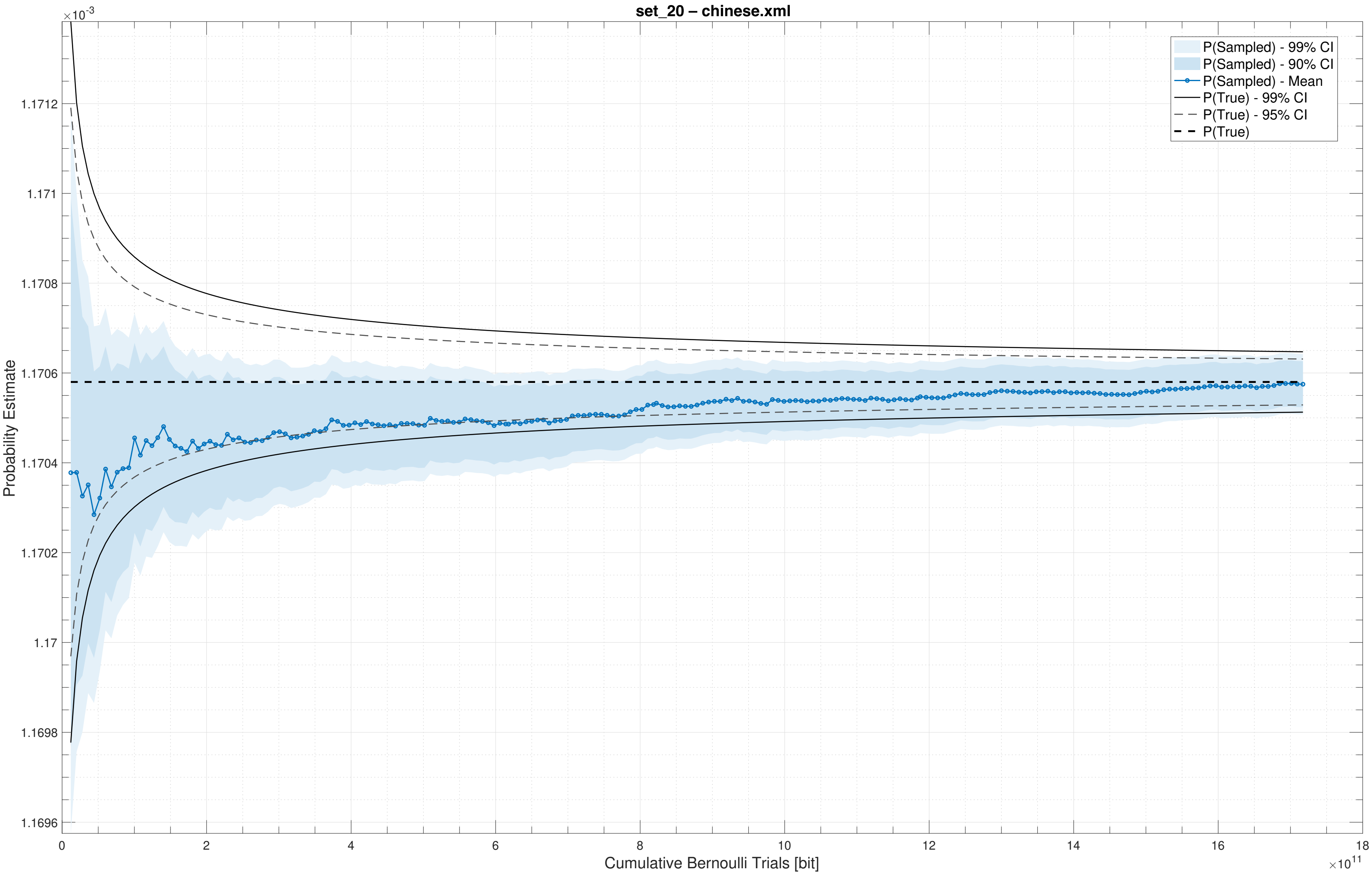


Figure A.5: Convergence trace for Aralia fault tree 5, 60 second timeout

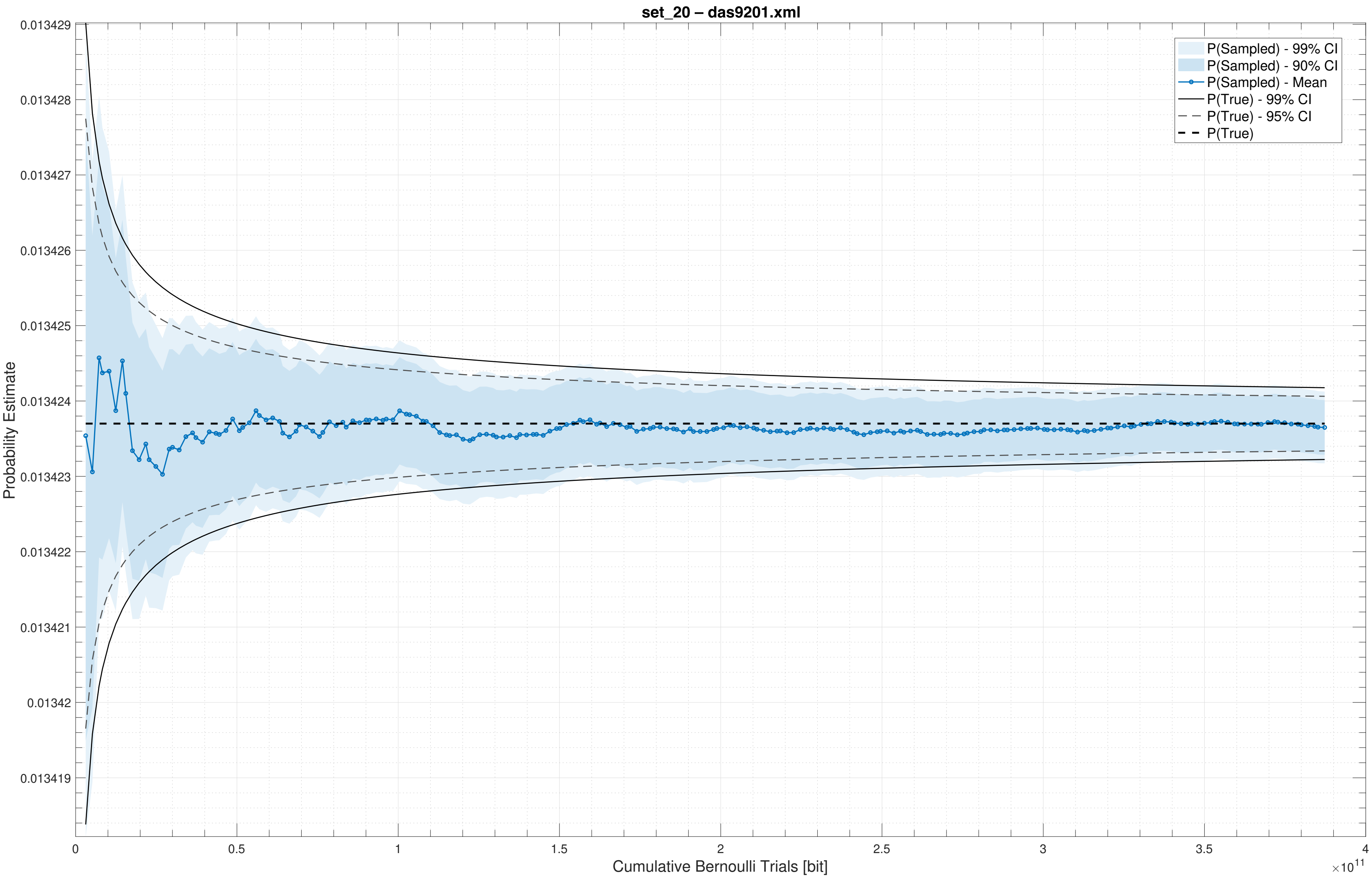


Figure A.6: Convergence trace for Aralia fault tree 6, 60 second timeout

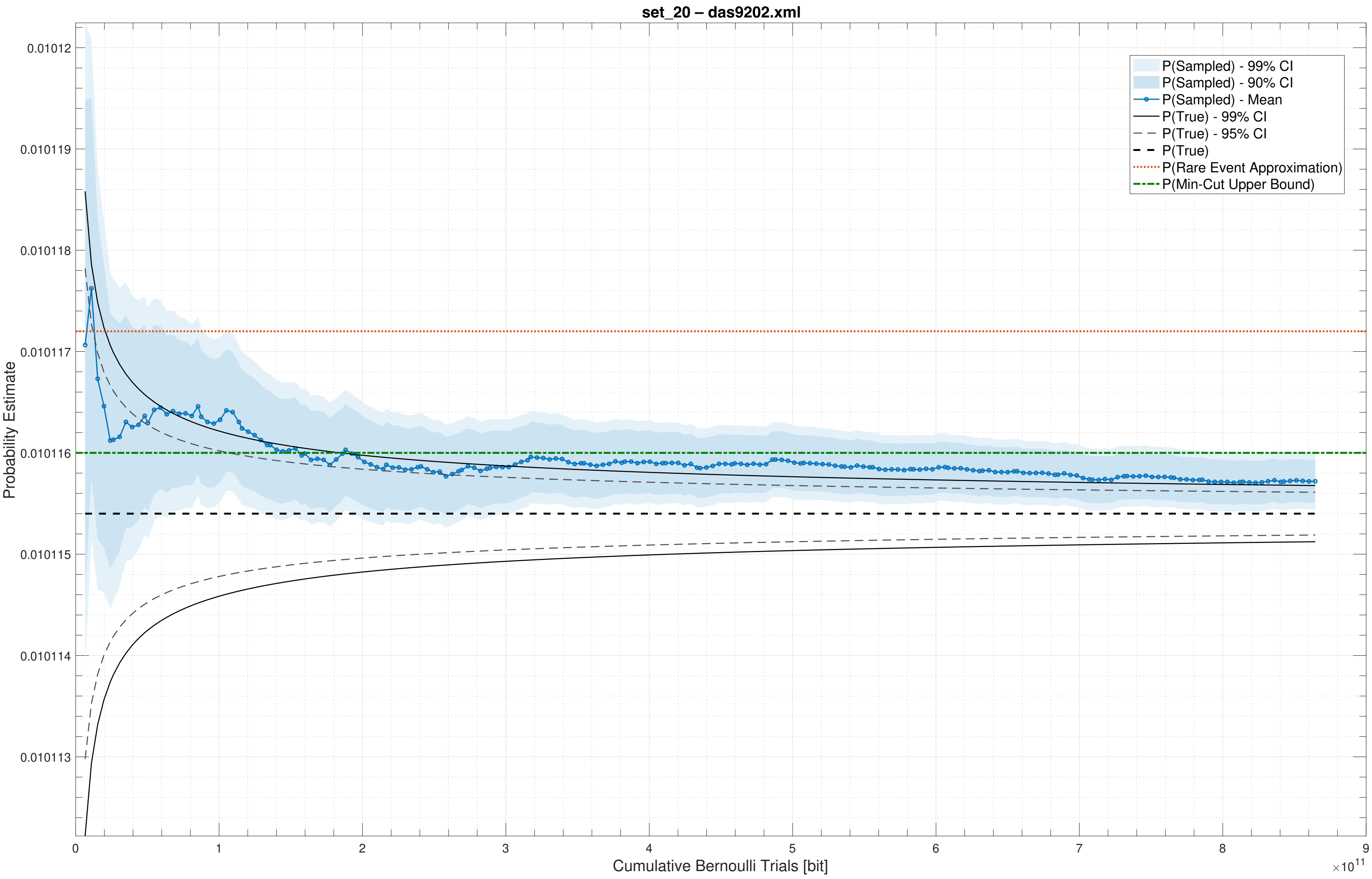


Figure A.7: Convergence trace for Aralia fault tree 7, 60 second timeout

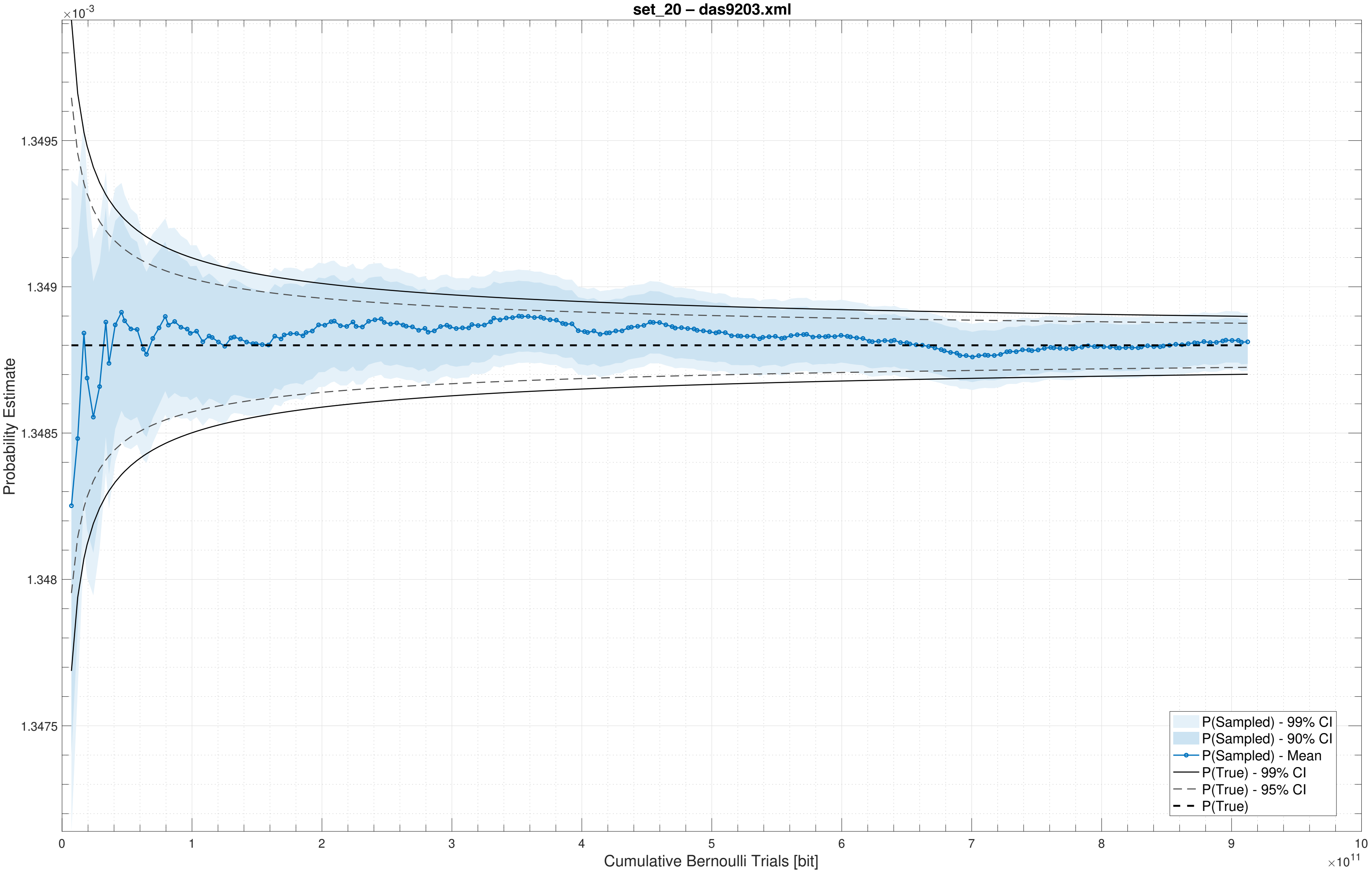


Figure A.8: Convergence trace for Aralia fault tree 8, 60 second timeout

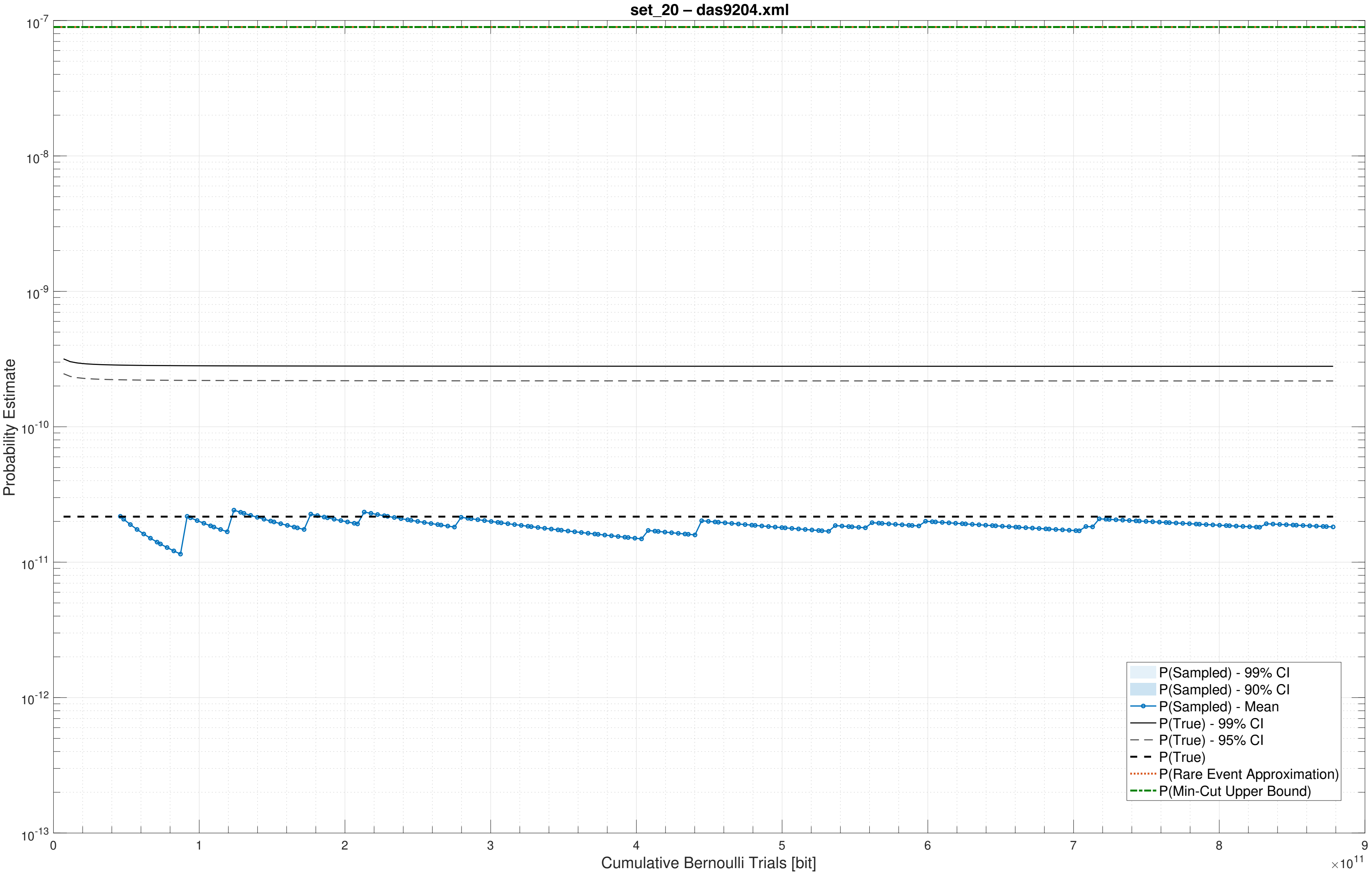


Figure A.9: Convergence trace for Aralia fault tree 9, 60 second timeout

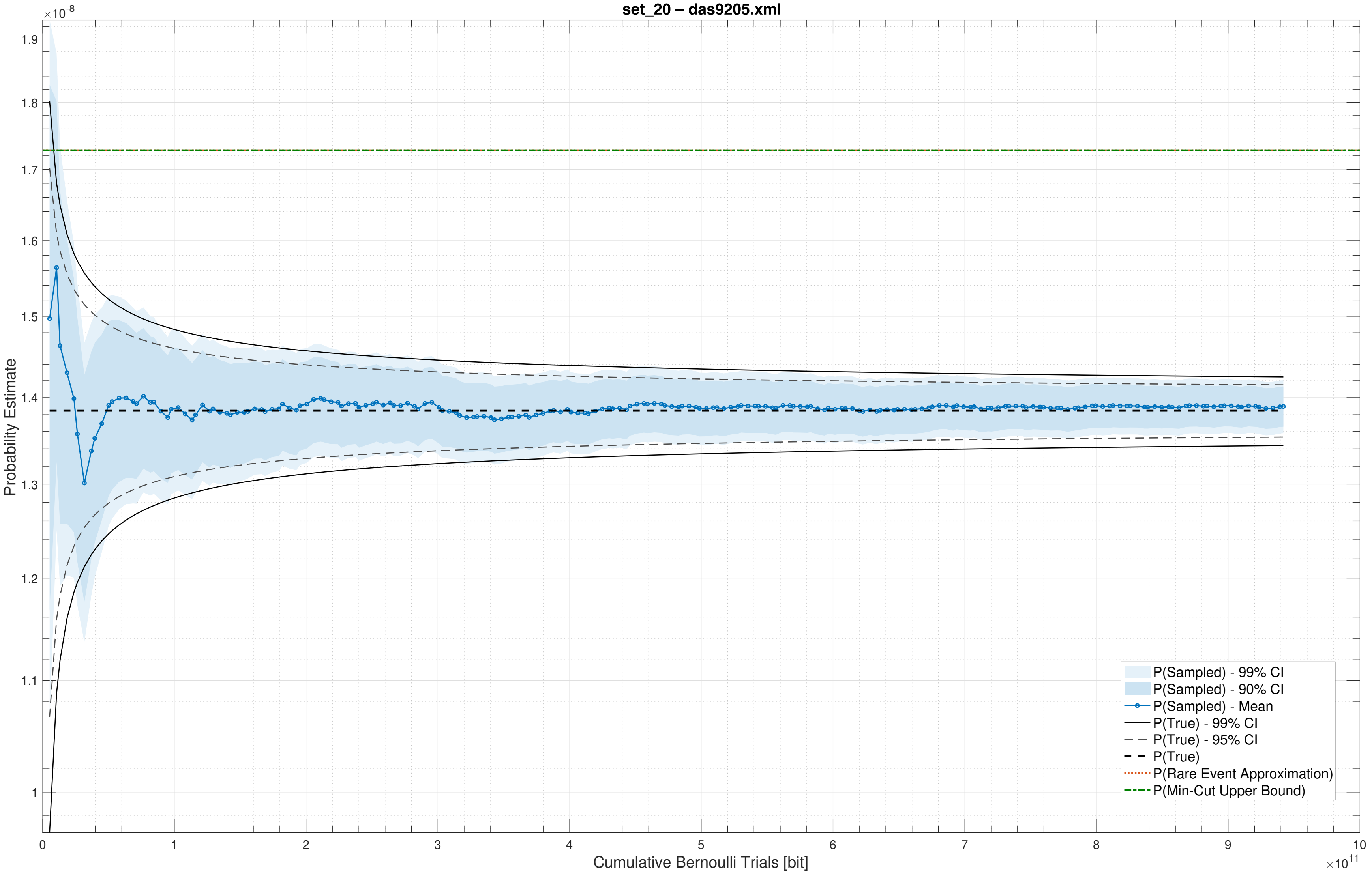


Figure A.10: Convergence trace for Aralia fault tree 10, 60 second timeout

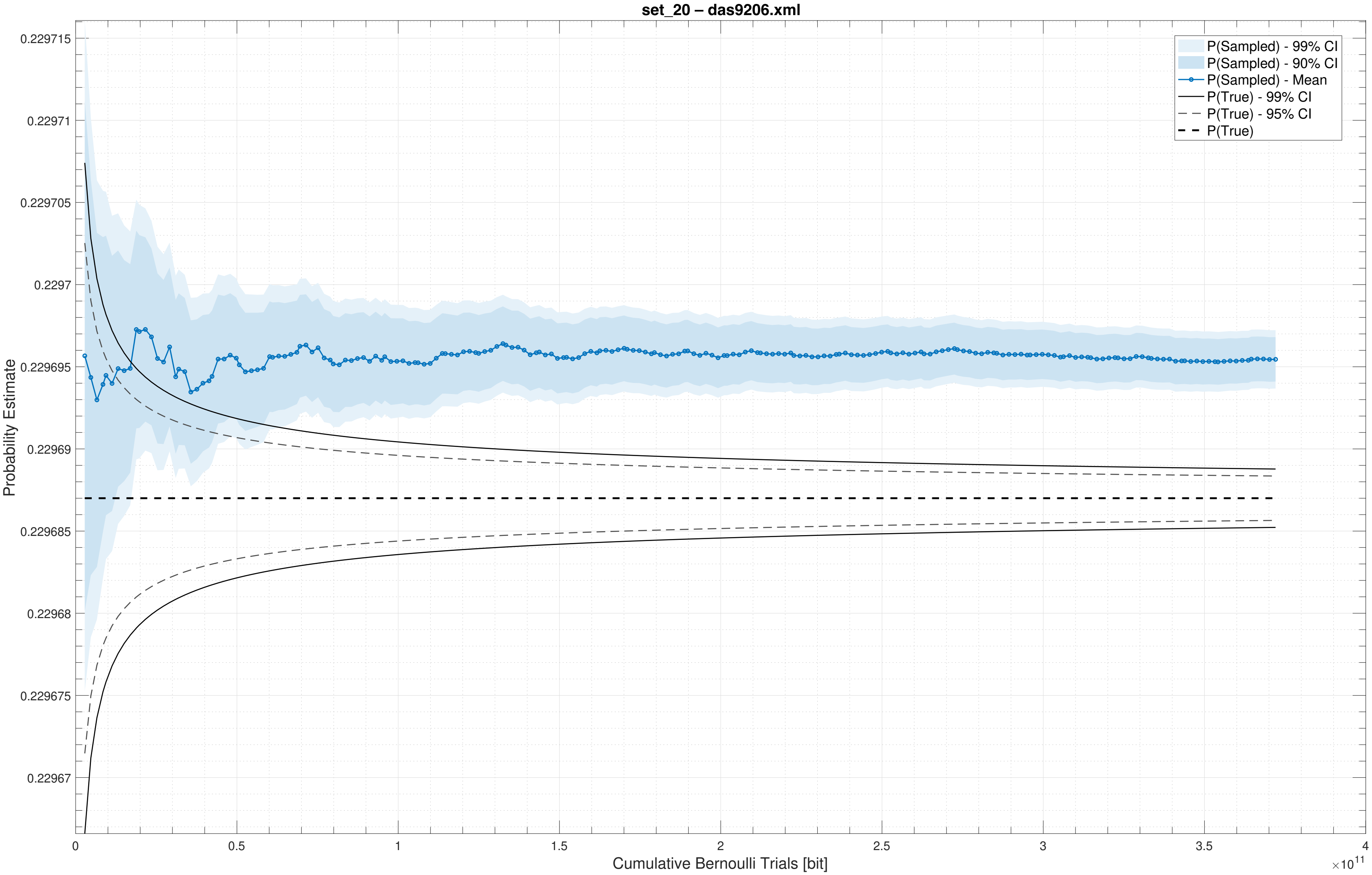


Figure A.11: Convergence trace for Aralia fault tree 11, 60 second timeout

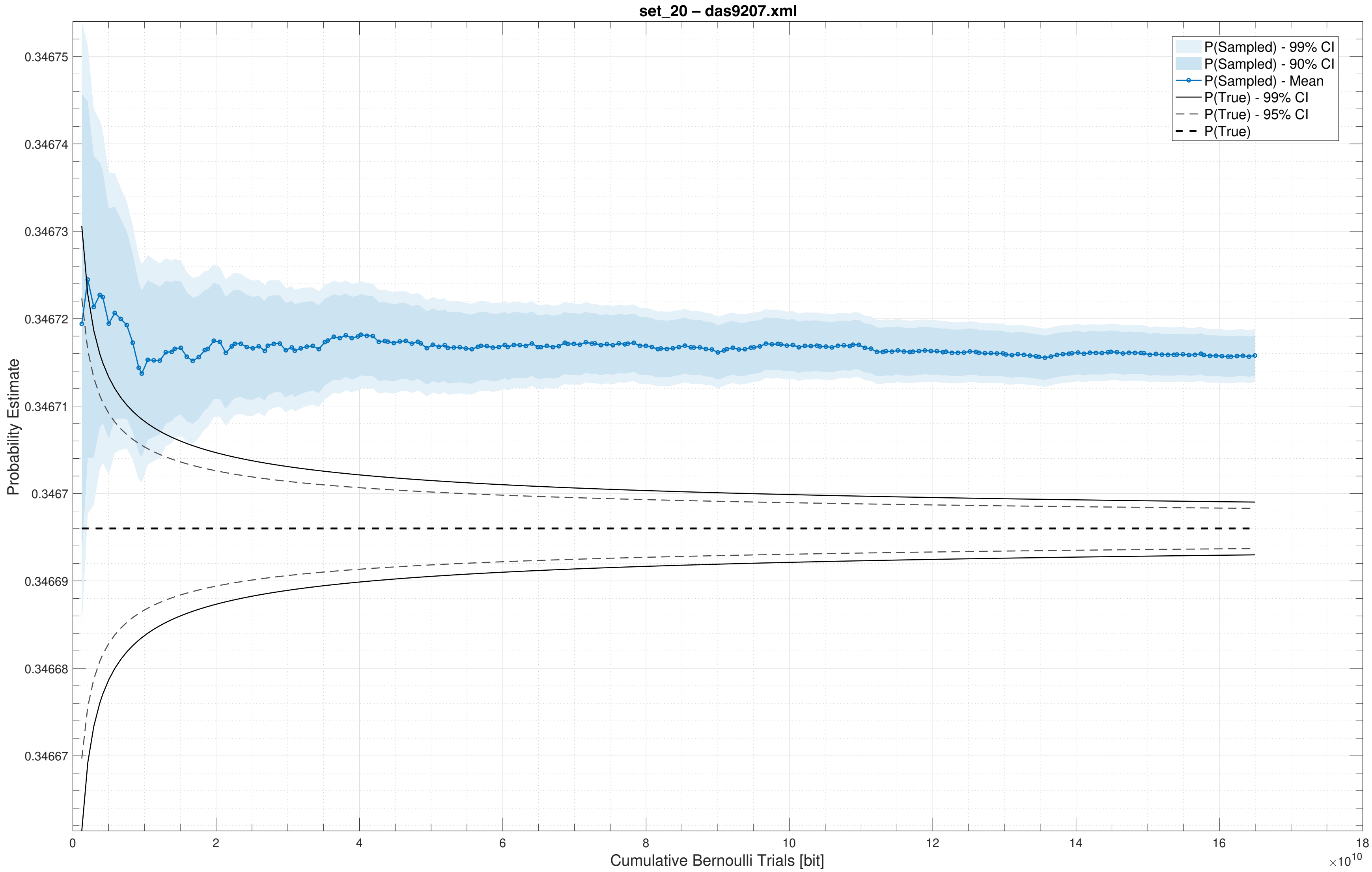


Figure A.12: Convergence trace for Aralia fault tree 12, 60 second timeout

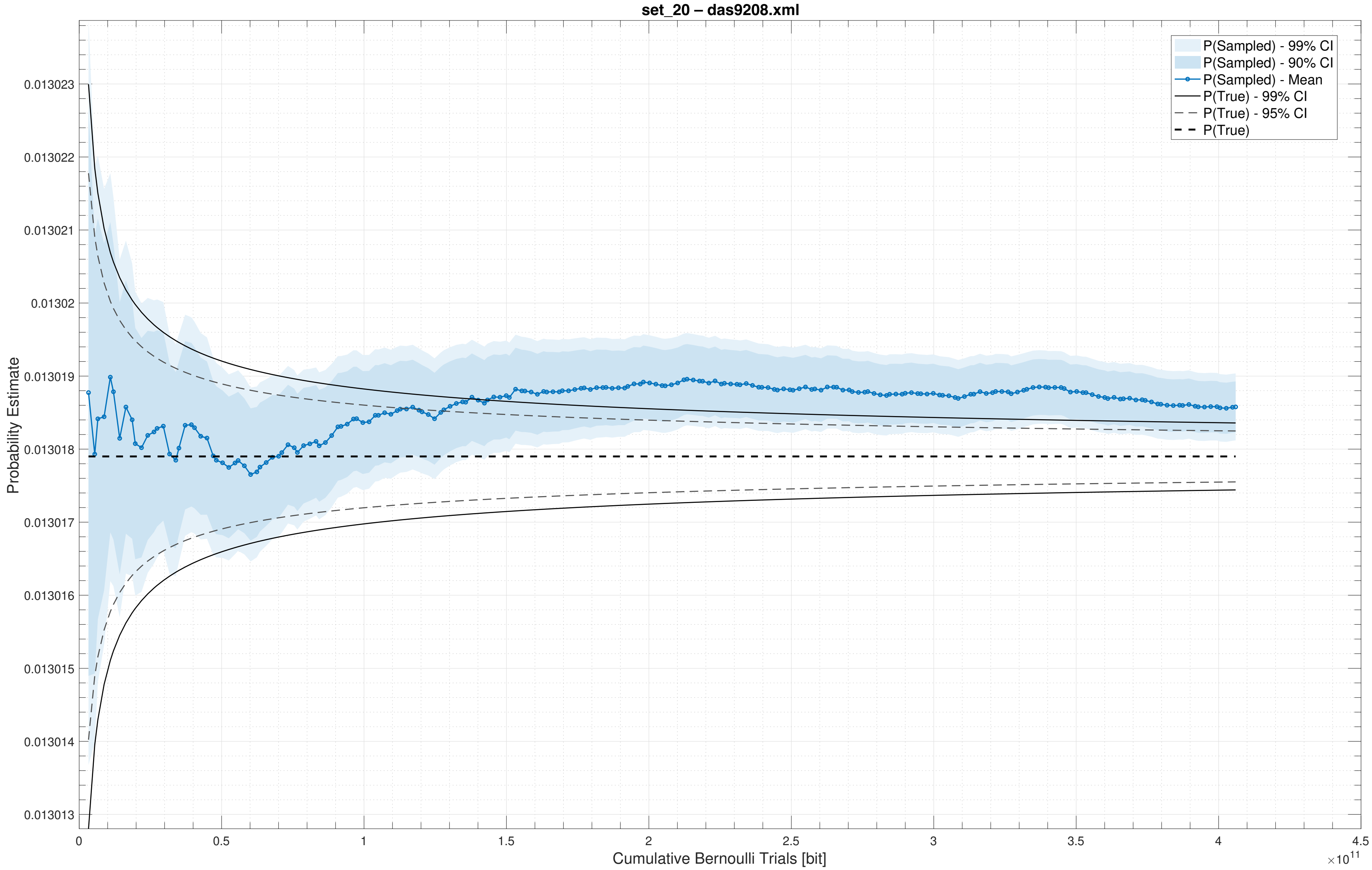


Figure A.13: Convergence trace for Aralia fault tree 13, 60 second timeout

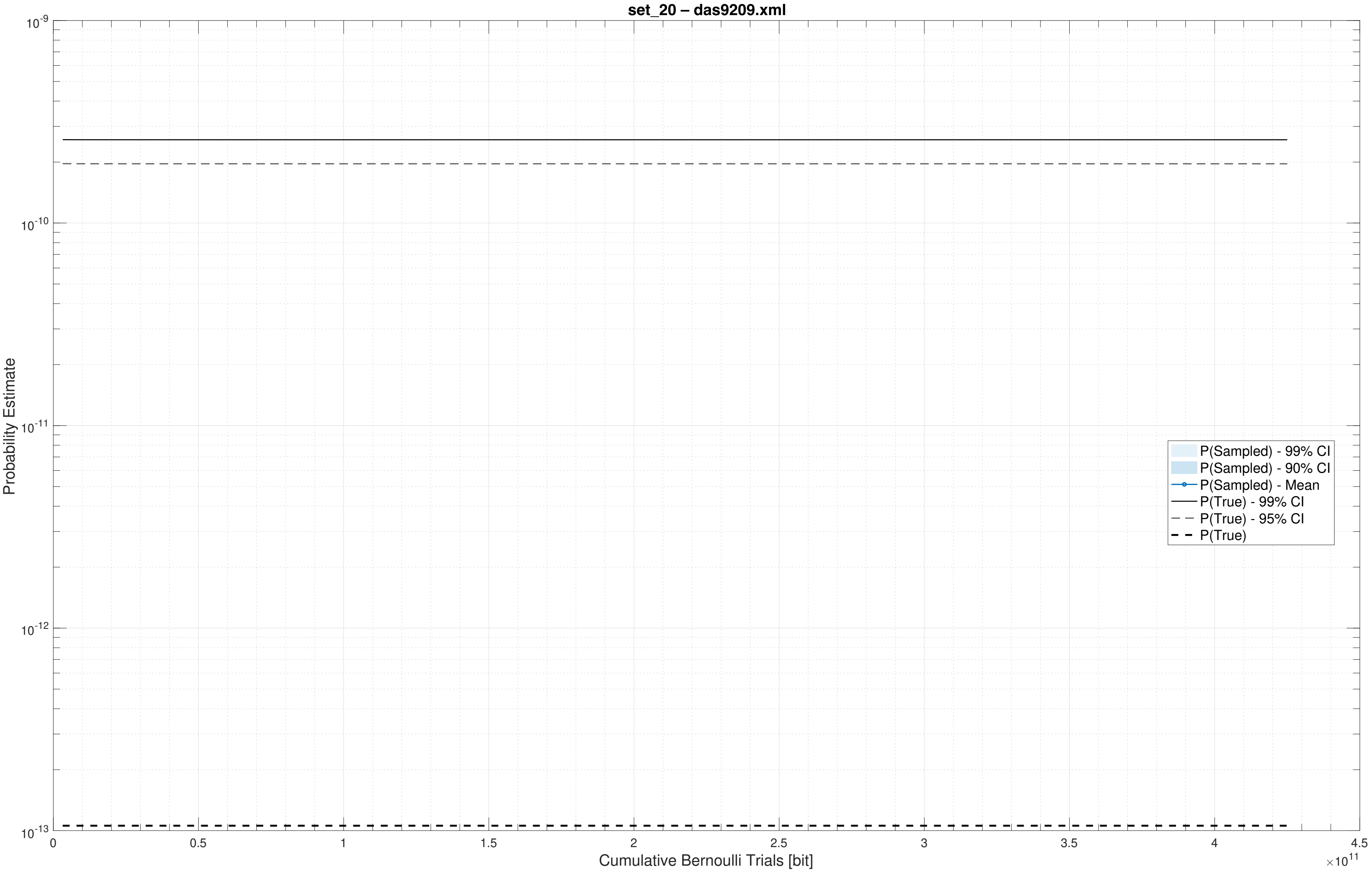


Figure A.14: Convergence trace for Aralia fault tree 14, 60 second timeout

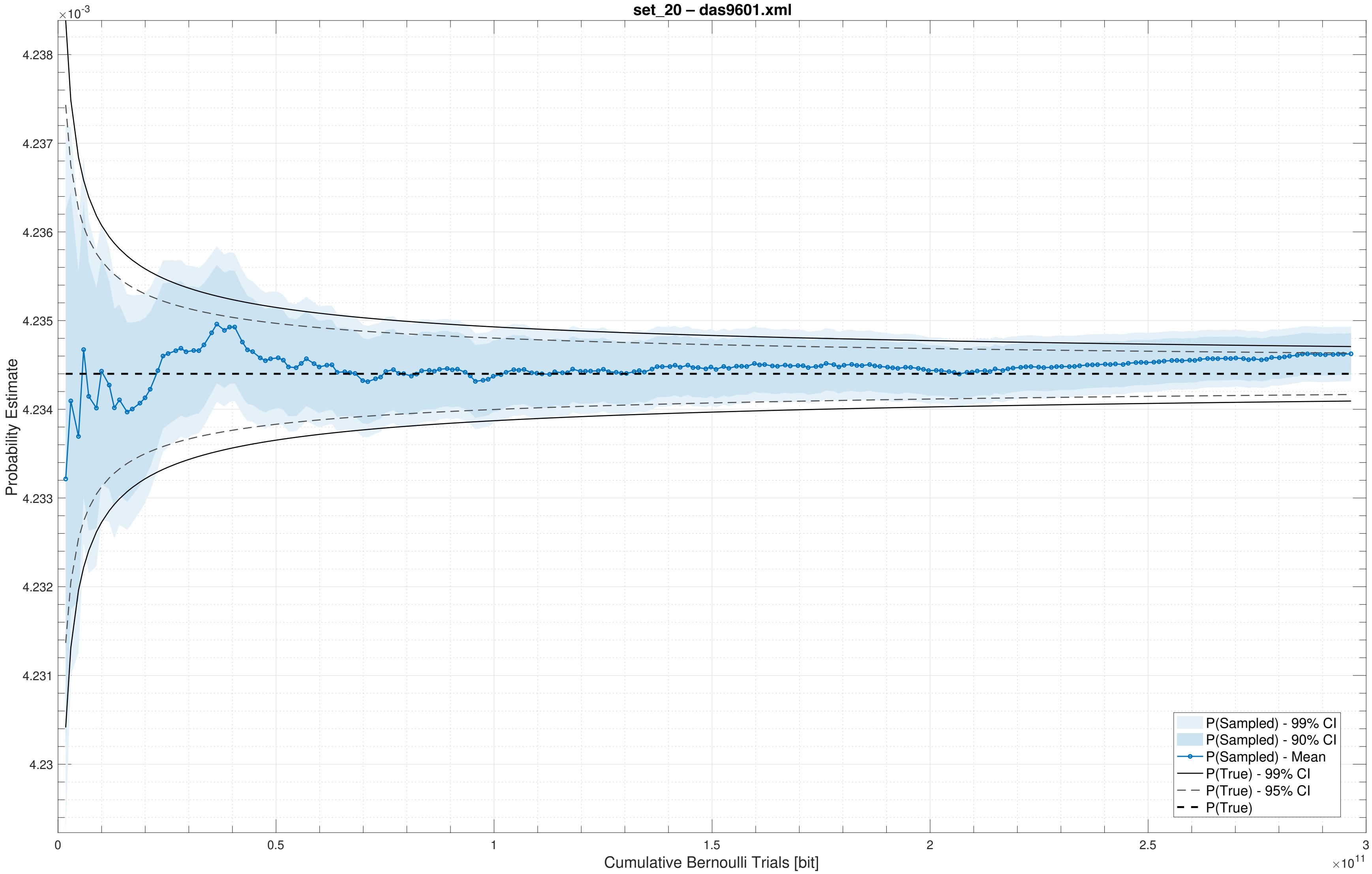


Figure A.15: Convergence trace for Aralia fault tree 15, 60 second timeout

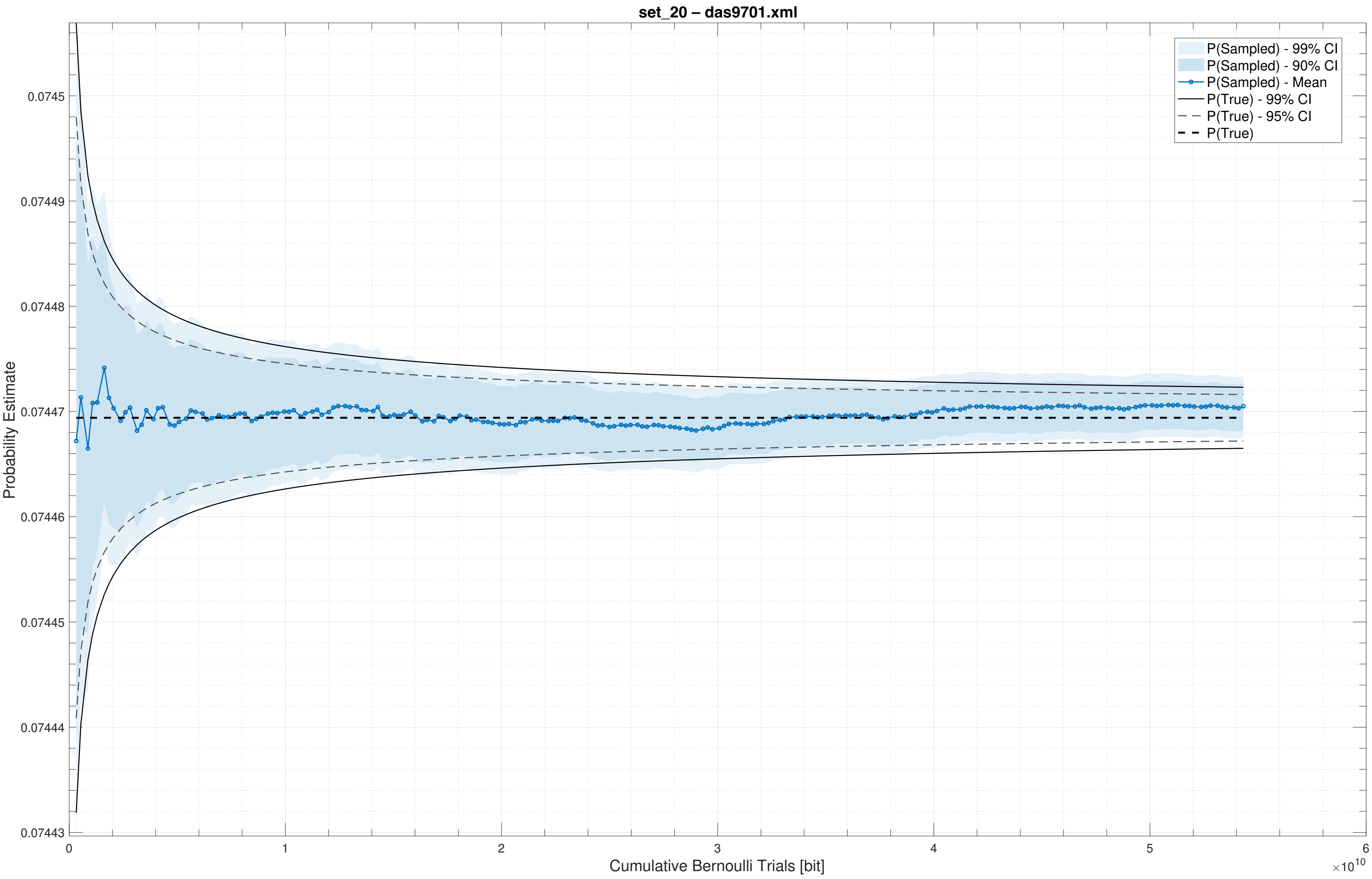


Figure A.16: Convergence trace for Aralia fault tree 16, 60 second timeout

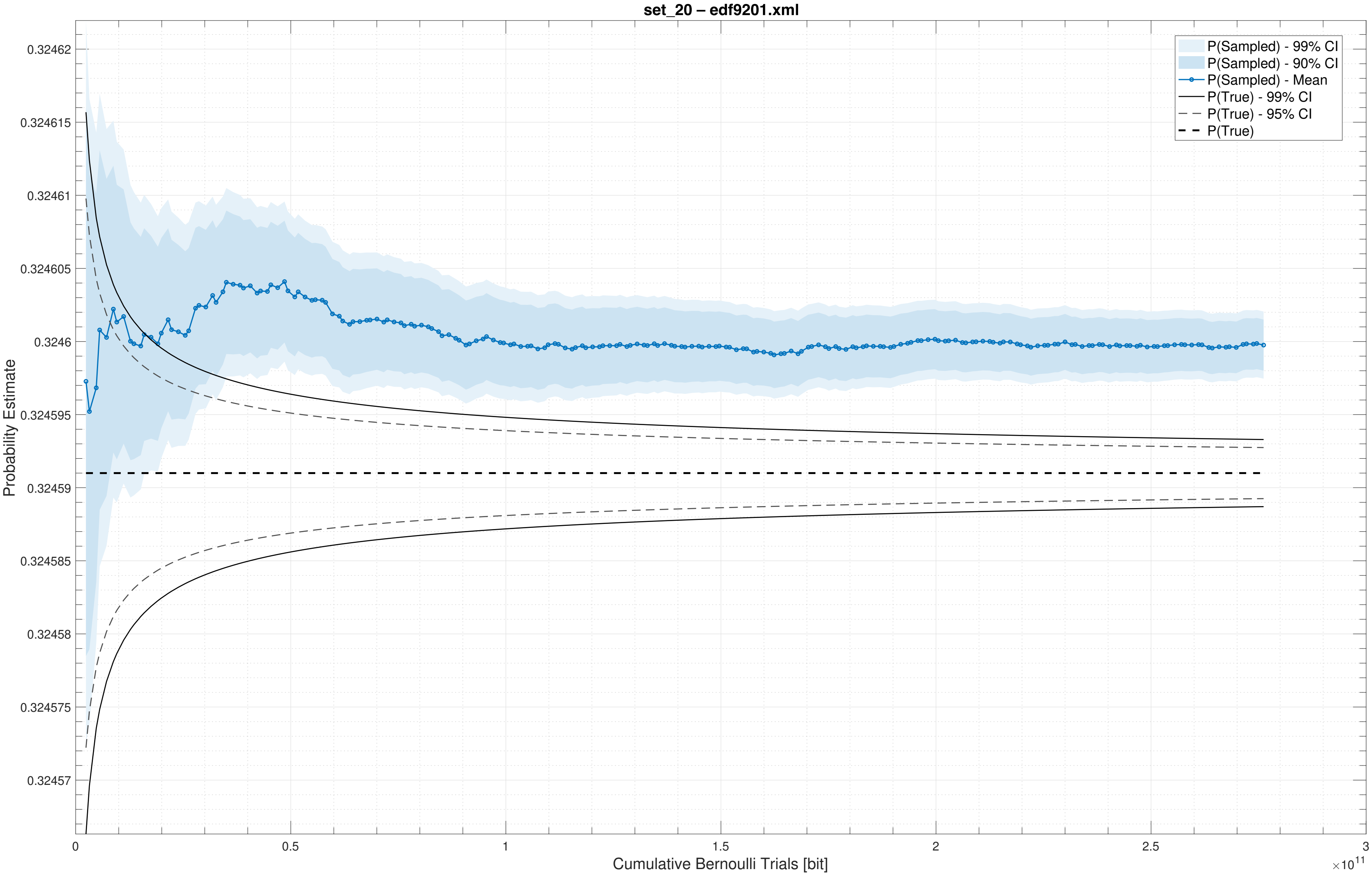


Figure A.17: Convergence trace for Aralia fault tree 17, 60 second timeout

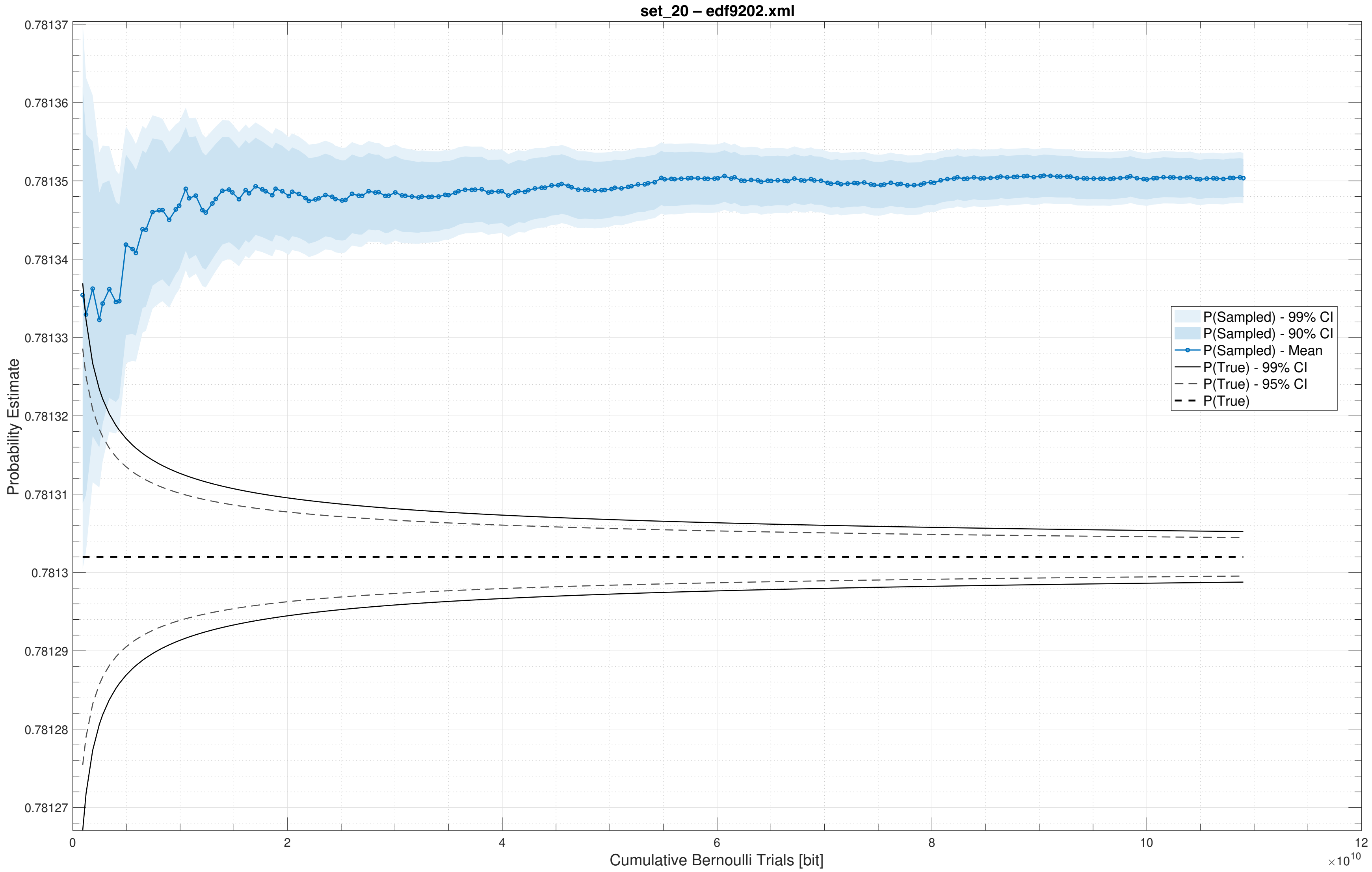


Figure A.18: Convergence trace for Aralia fault tree 18, 60 second timeout

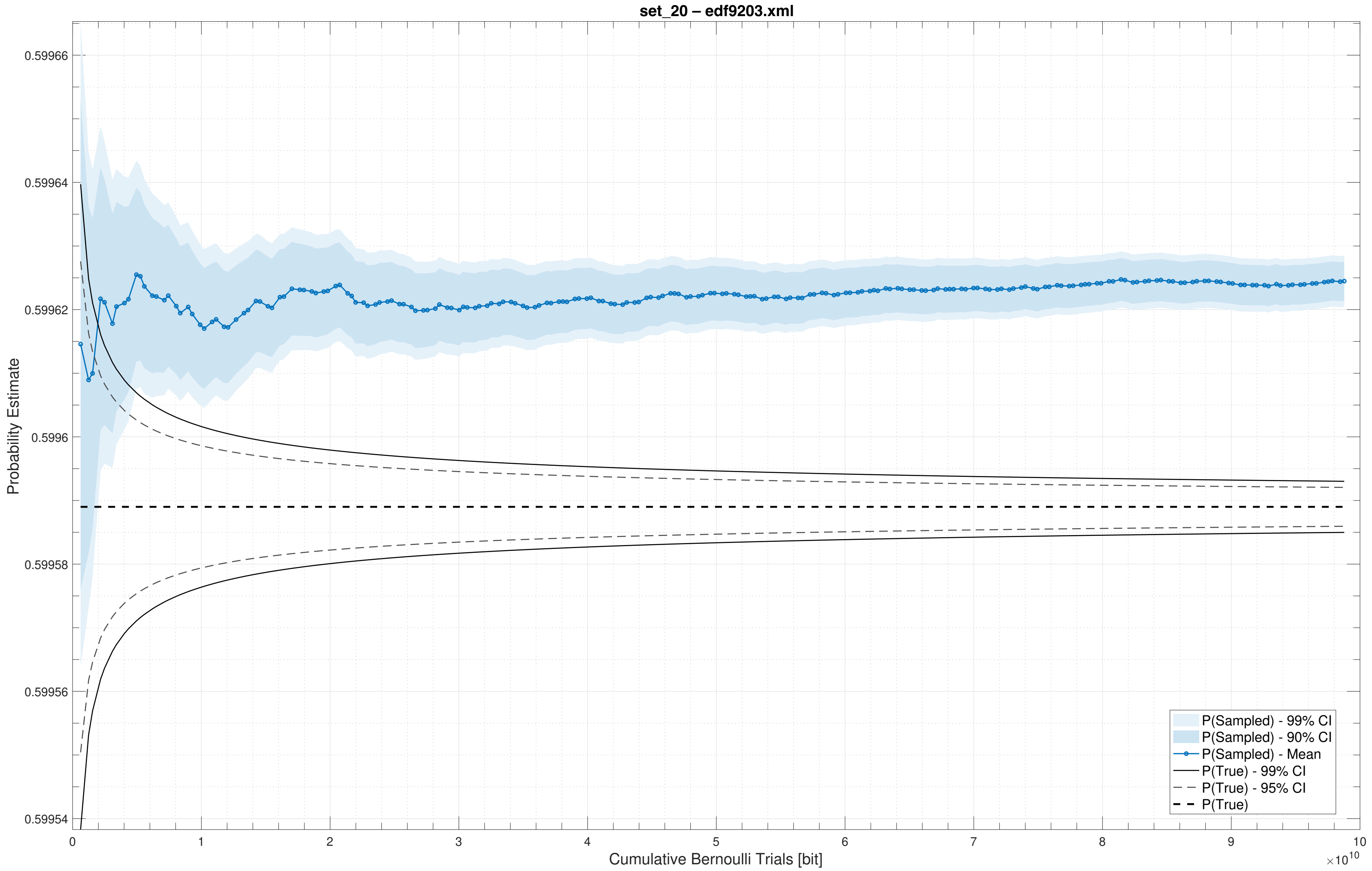


Figure A.19: Convergence trace for Aralia fault tree 19, 60 second timeout

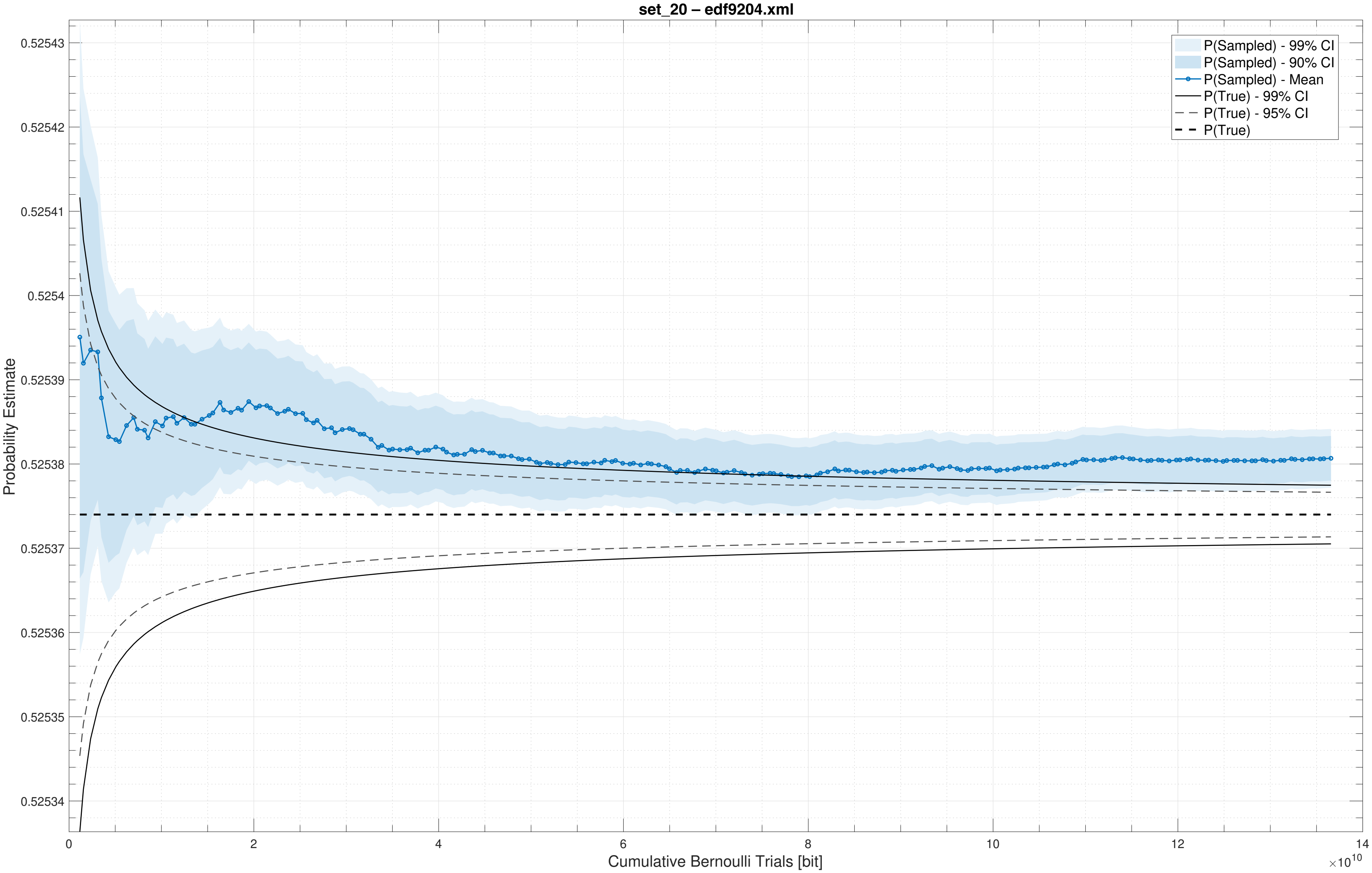


Figure A.20: Convergence trace for Aralia fault tree 20, 60 second timeout

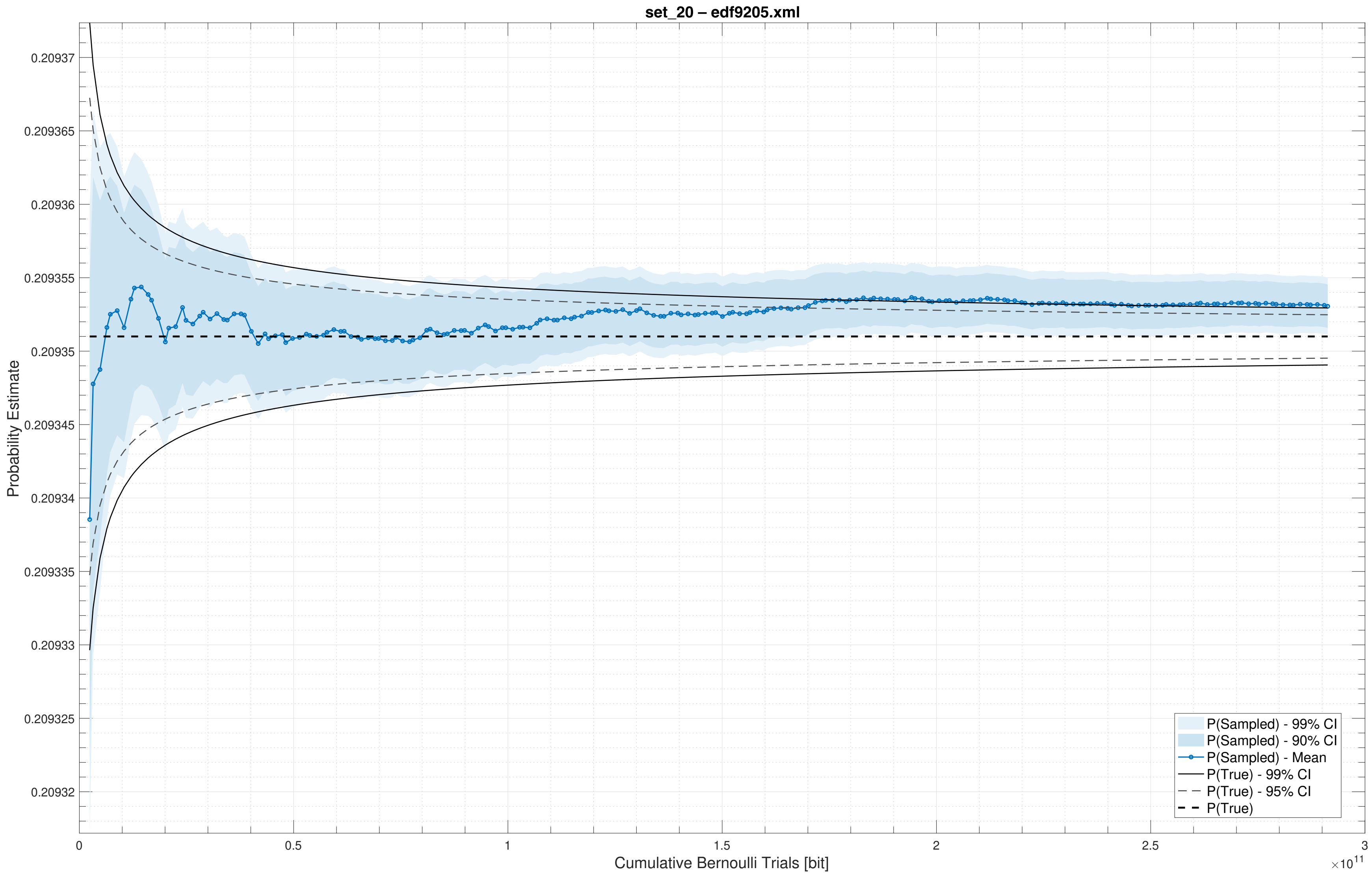


Figure A.21: Convergence trace for Aralia fault tree 21, 60 second timeout

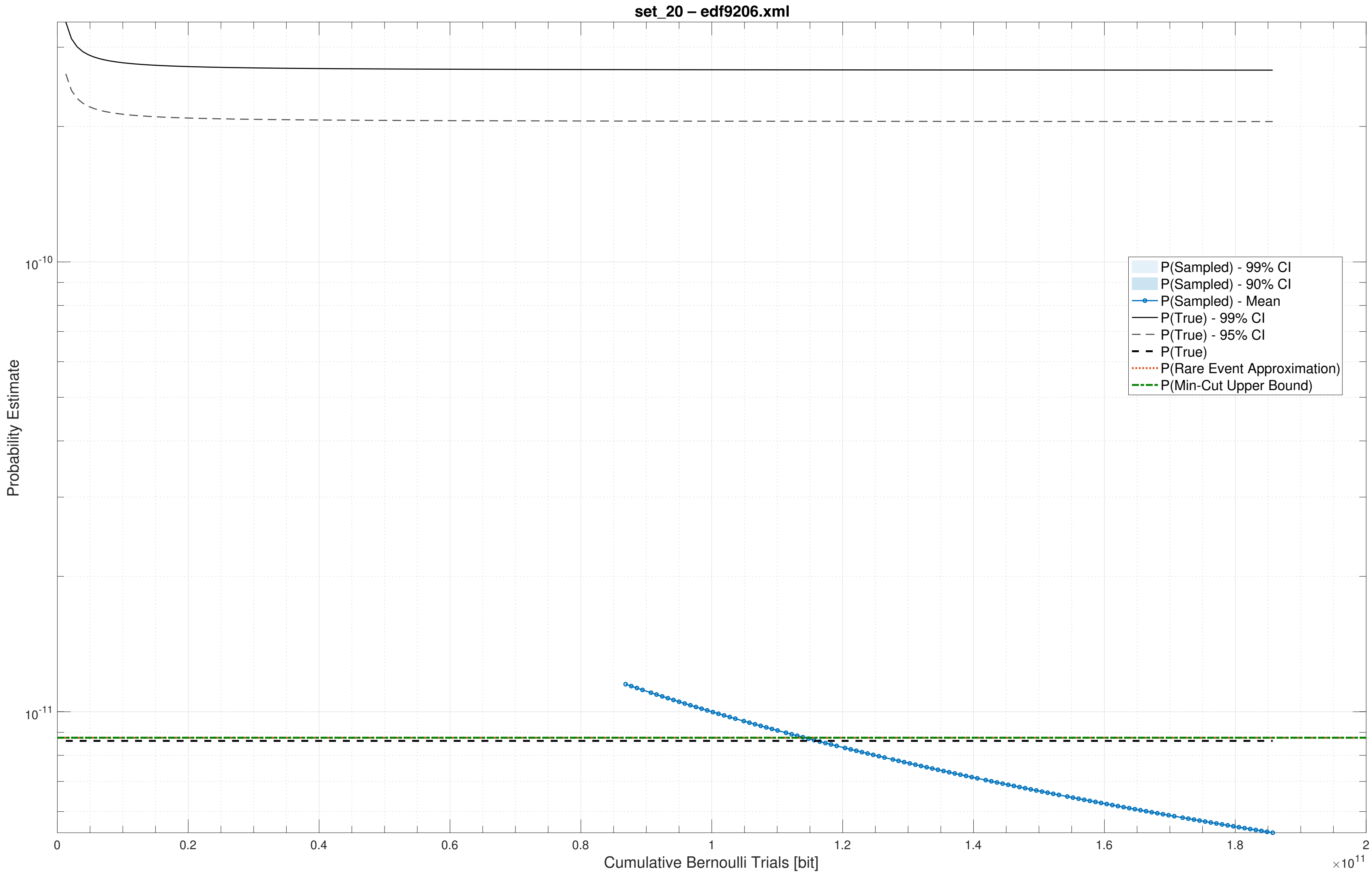


Figure A.22: Convergence trace for Aralia fault tree 22, 60 second timeout

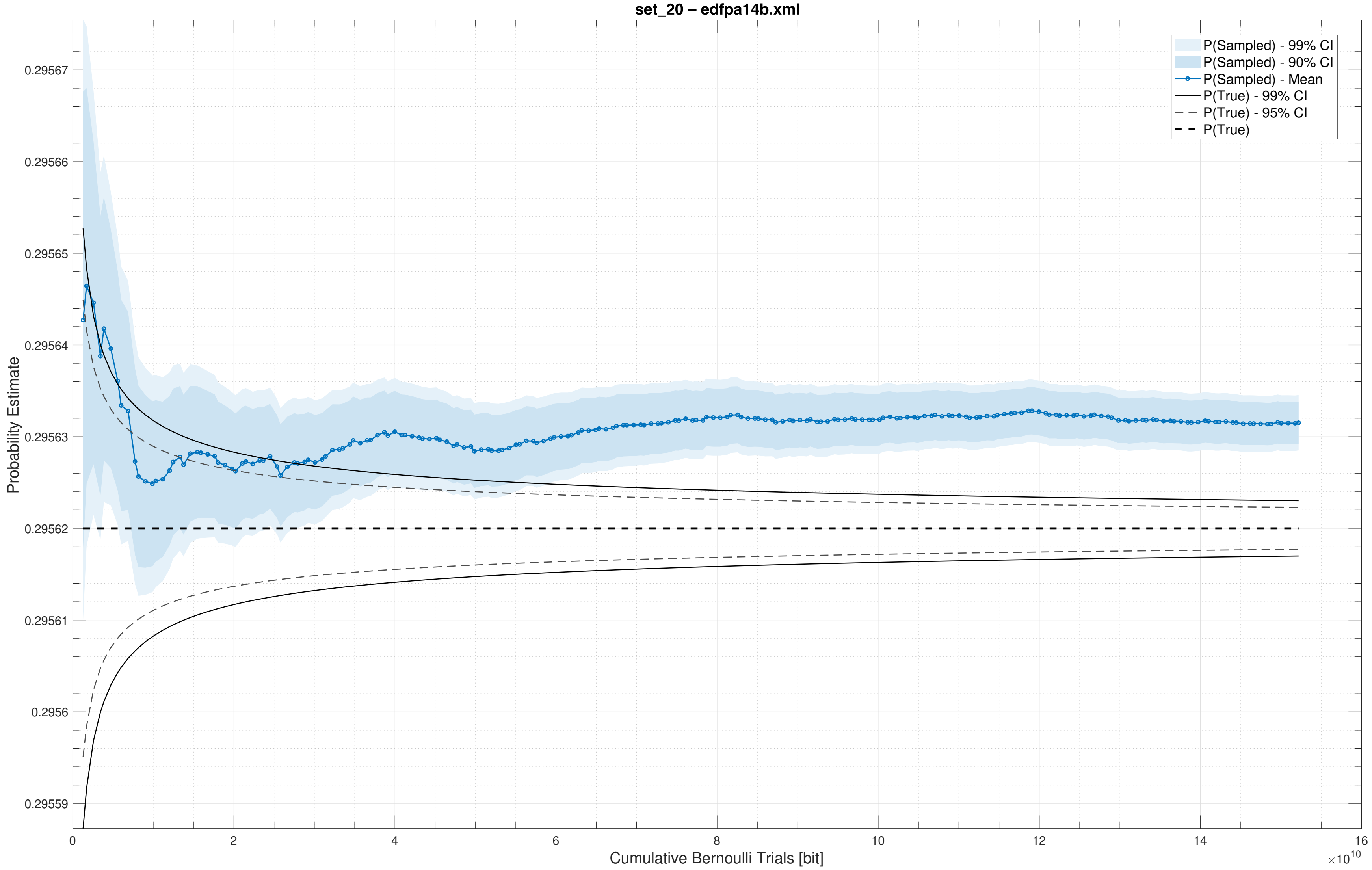


Figure A.23: Convergence trace for Aralia fault tree 23, 60 second timeout

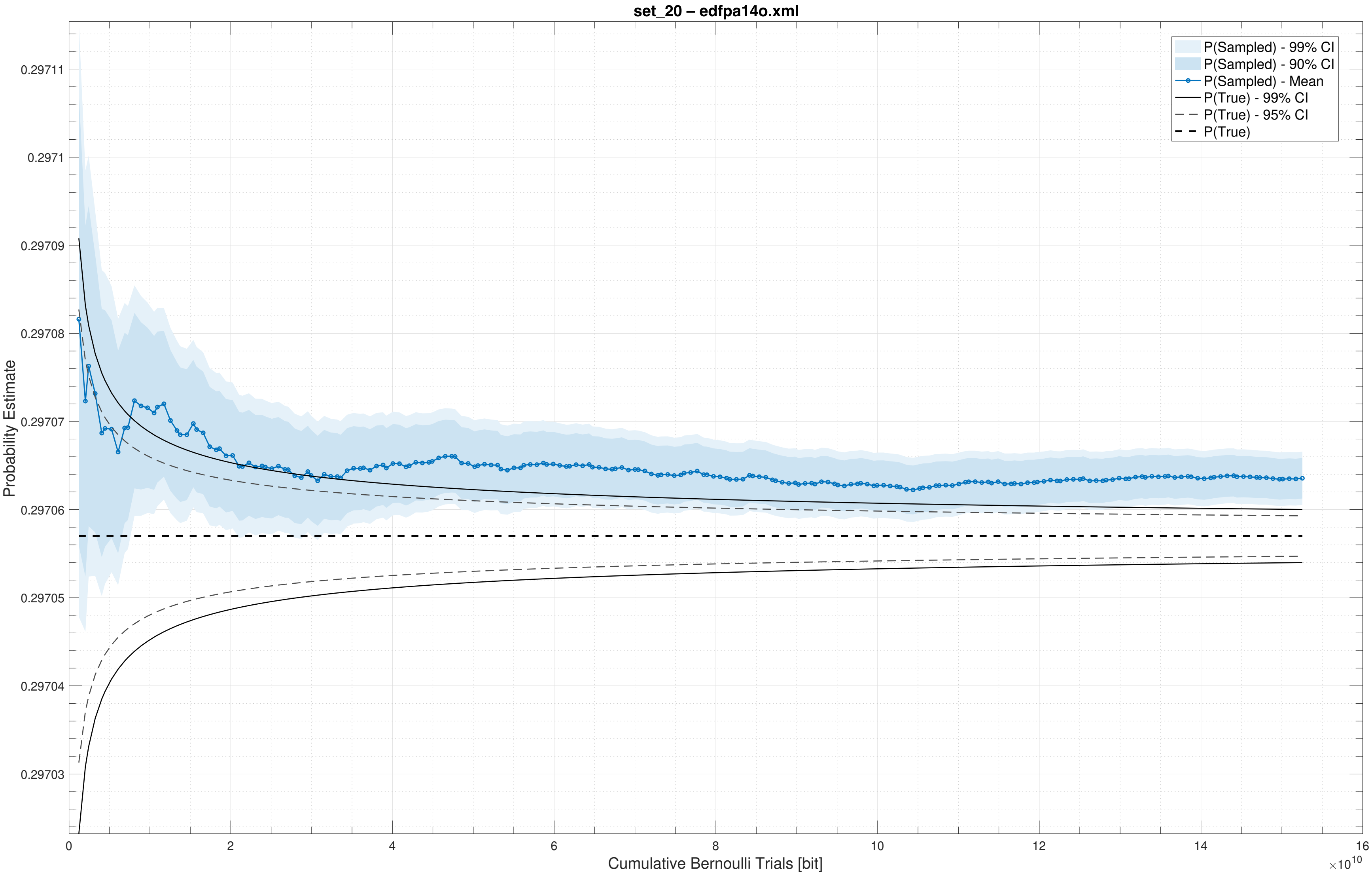


Figure A.24: Convergence trace for Aralia fault tree 24, 60 second timeout

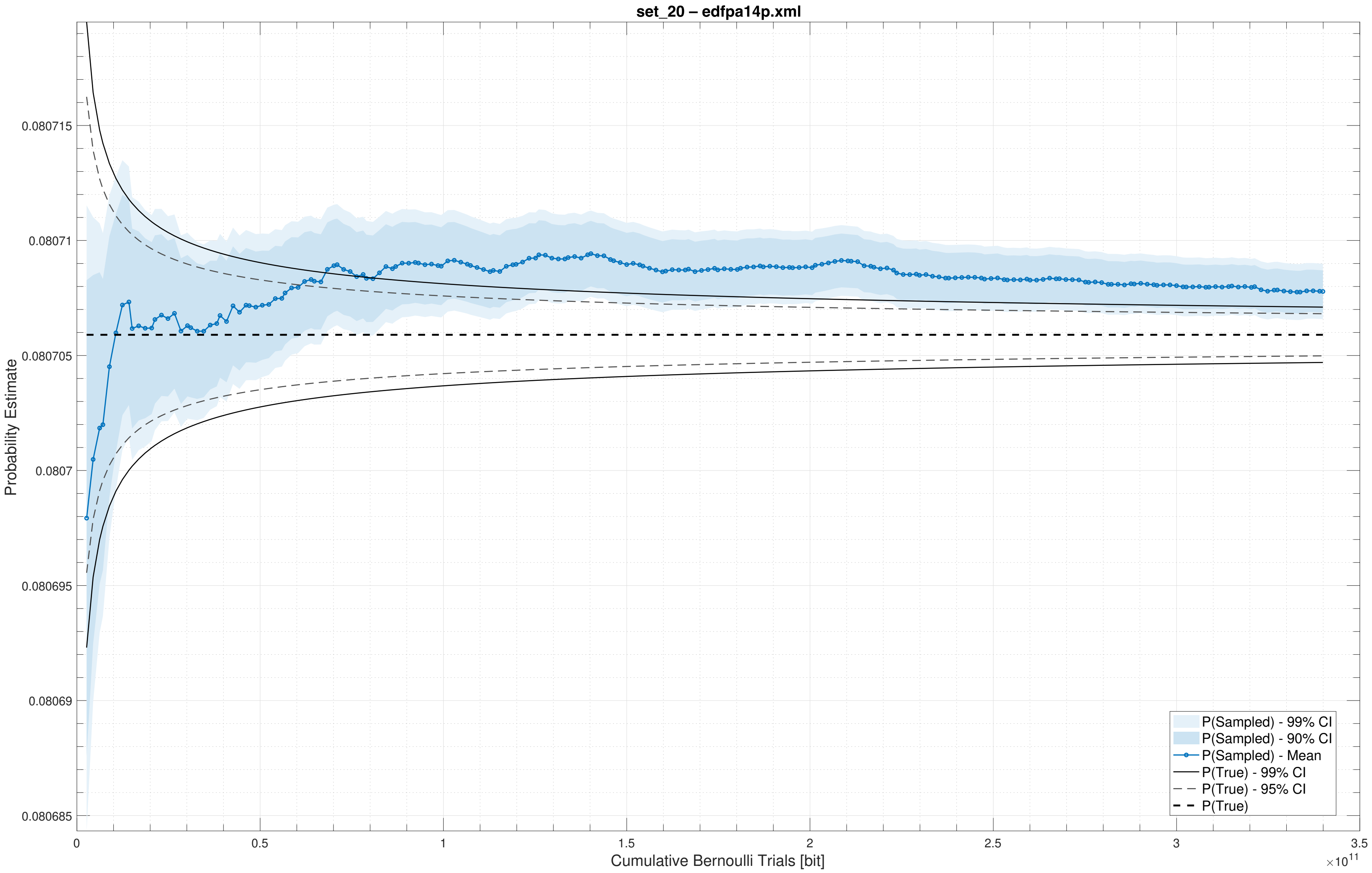


Figure A.25: Convergence trace for Aralia fault tree 25, 60 second timeout

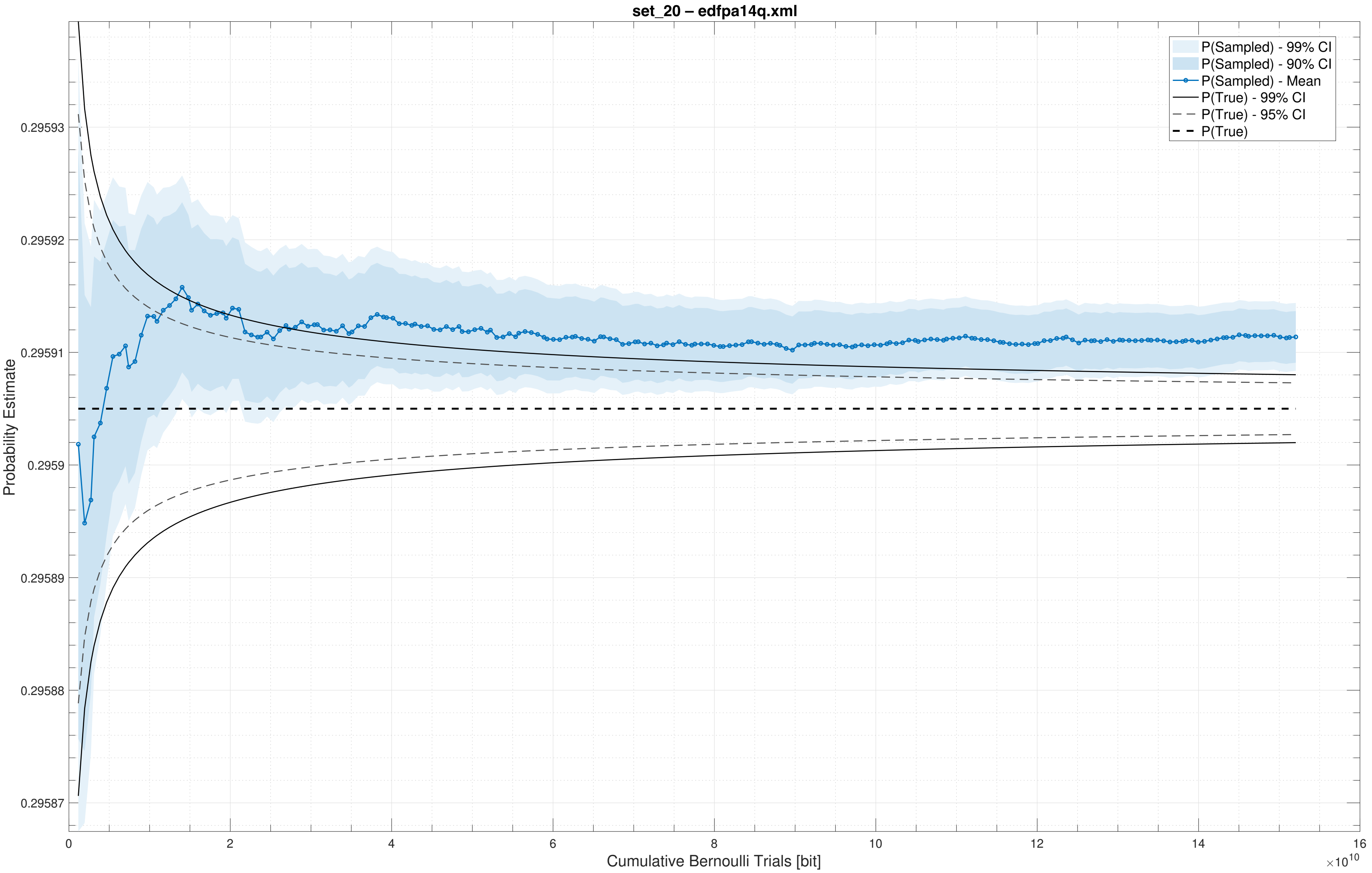


Figure A.26: Convergence trace for Aralia fault tree 26, 60 second timeout

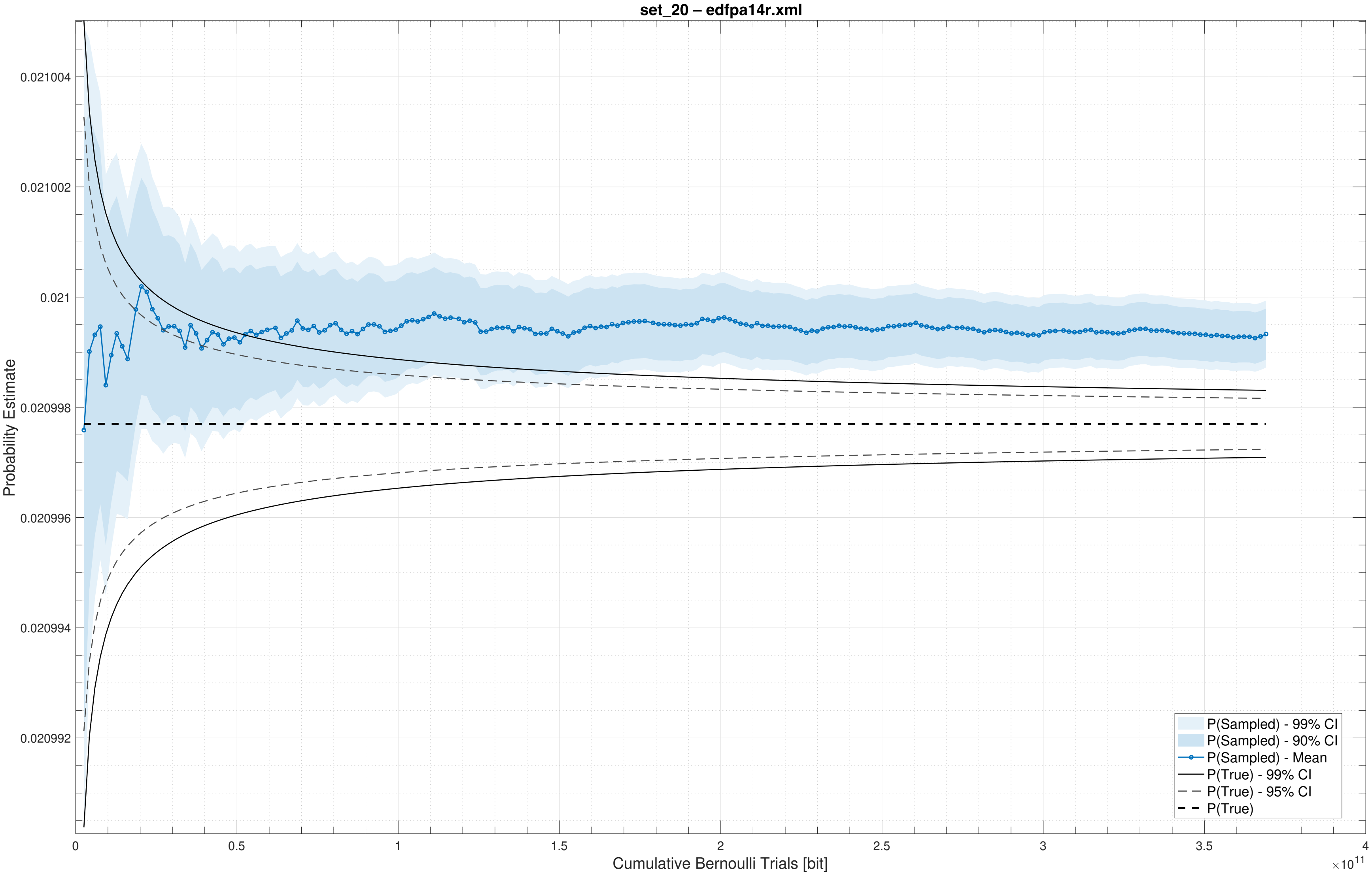


Figure A.27: Convergence trace for Aralia fault tree 27, 60 second timeout

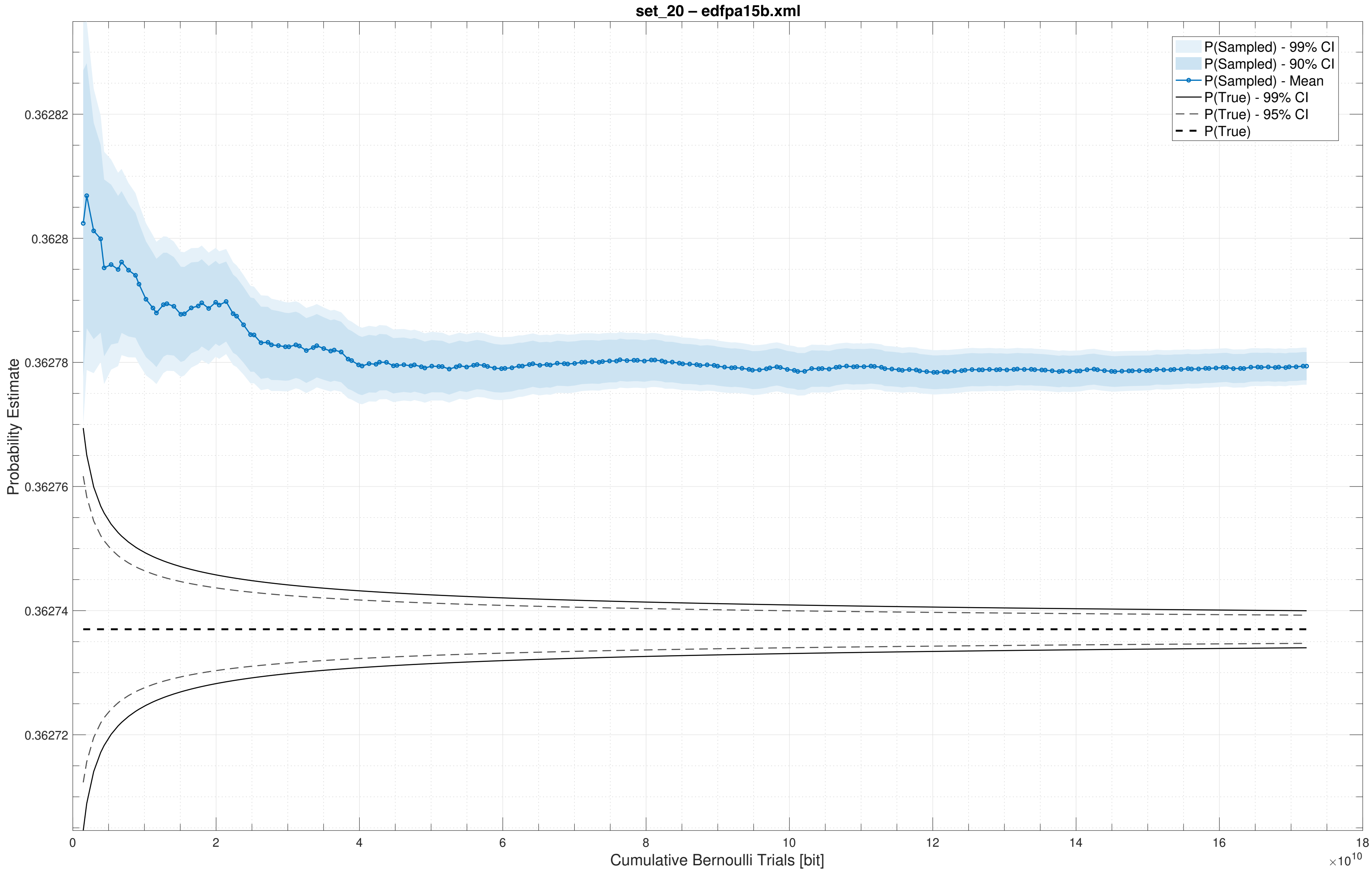


Figure A.28: Convergence trace for Aralia fault tree 28, 60 second timeout

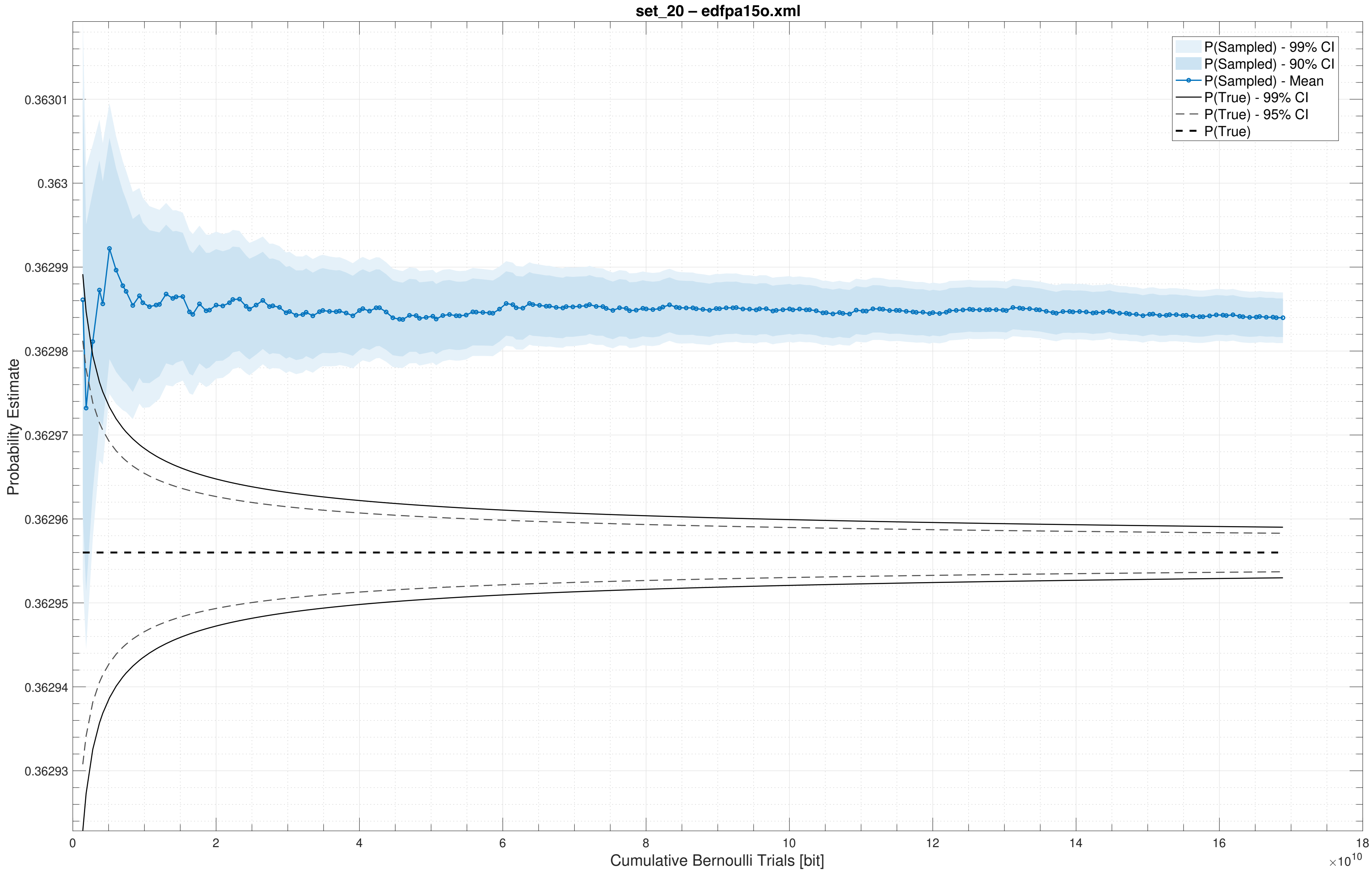


Figure A.29: Convergence trace for Aralia fault tree 29, 60 second timeout

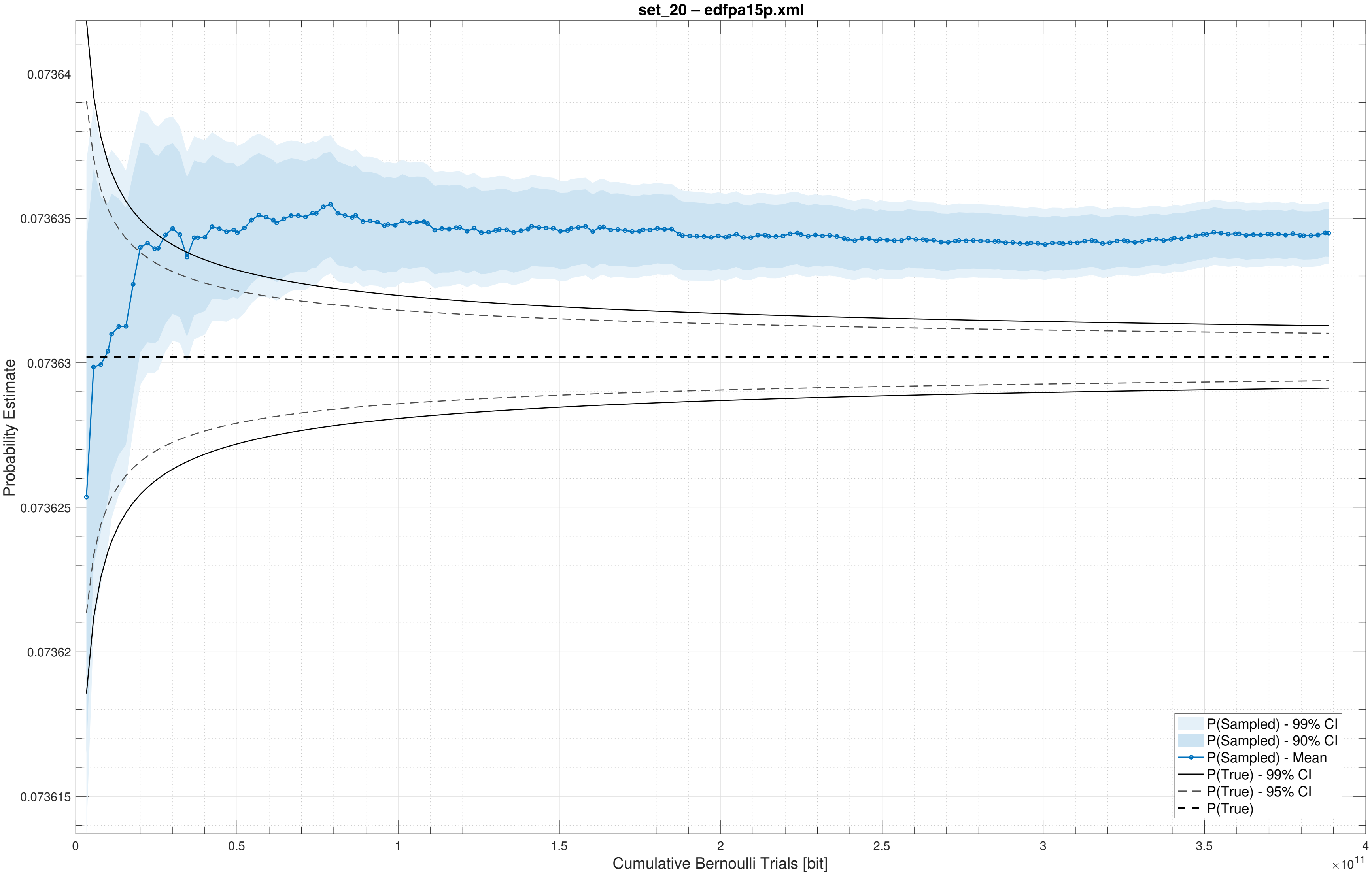


Figure A.30: Convergence trace for Aralia fault tree 30, 60 second timeout

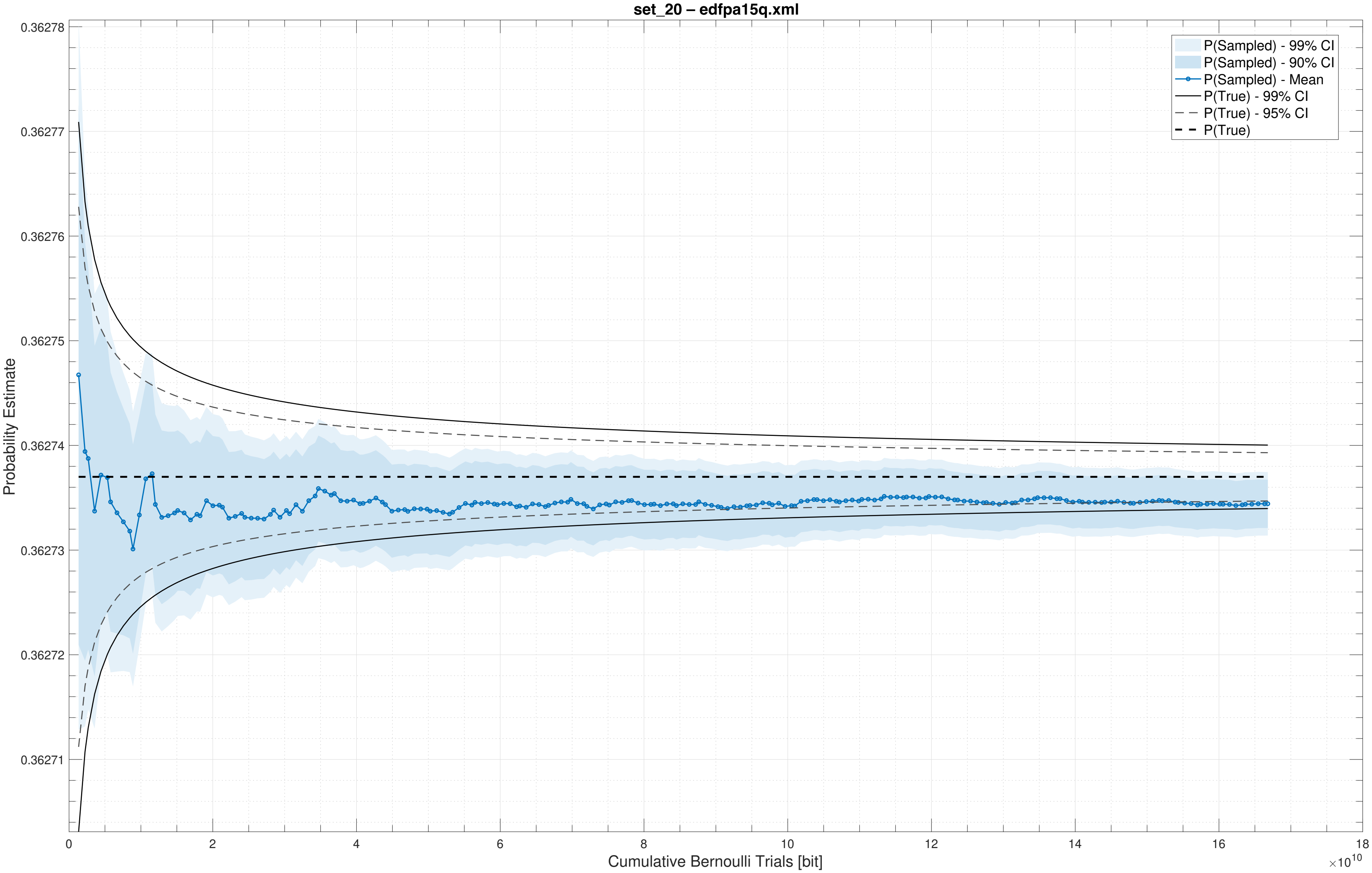


Figure A.31: Convergence trace for Aralia fault tree 31, 60 second timeout

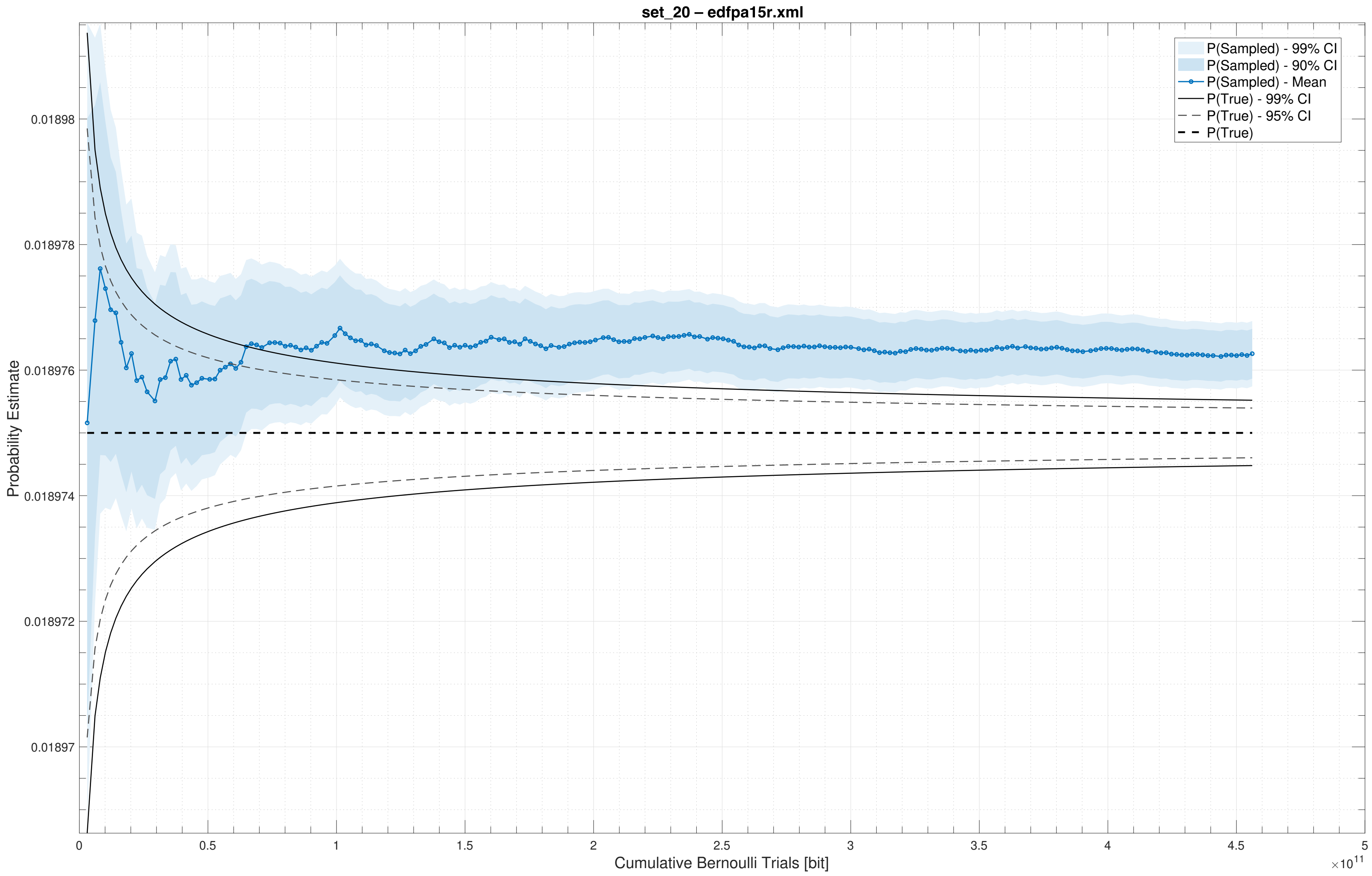


Figure A.32: Convergence trace for Aralia fault tree 32, 60 second timeout

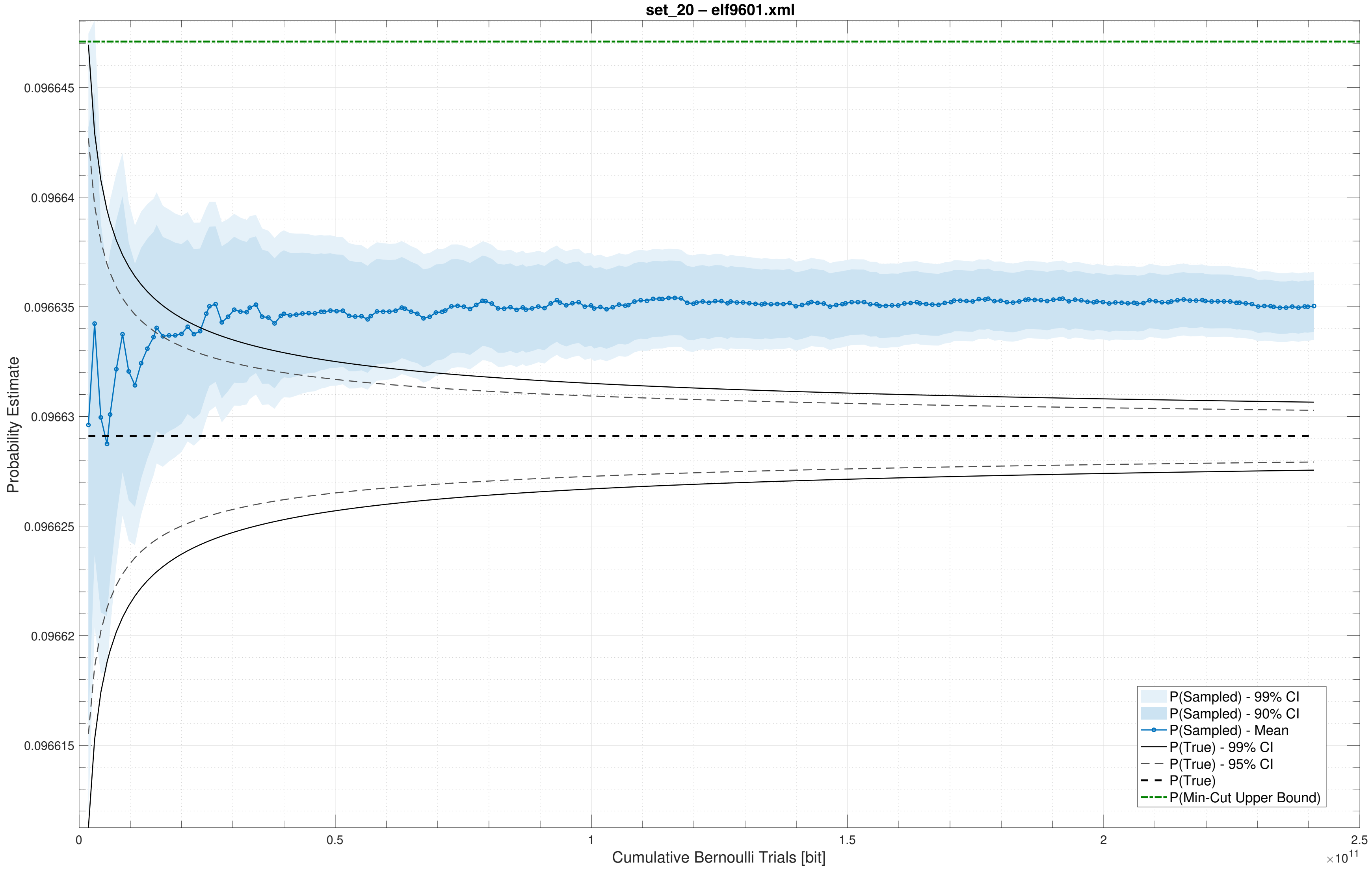


Figure A.33: Convergence trace for Aralia fault tree 33, 60 second timeout

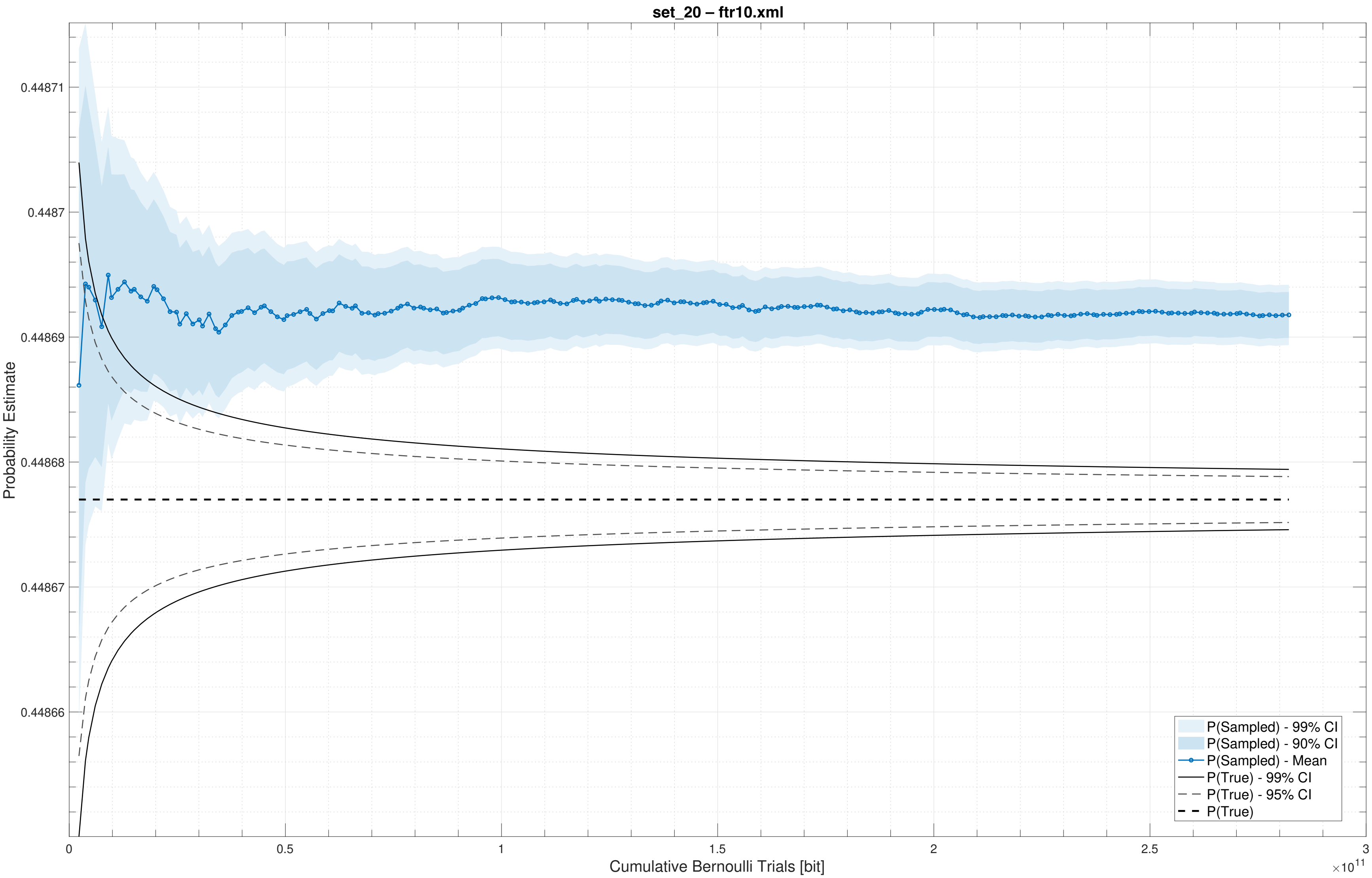


Figure A.34: Convergence trace for Aralia fault tree 34, 60 second timeout

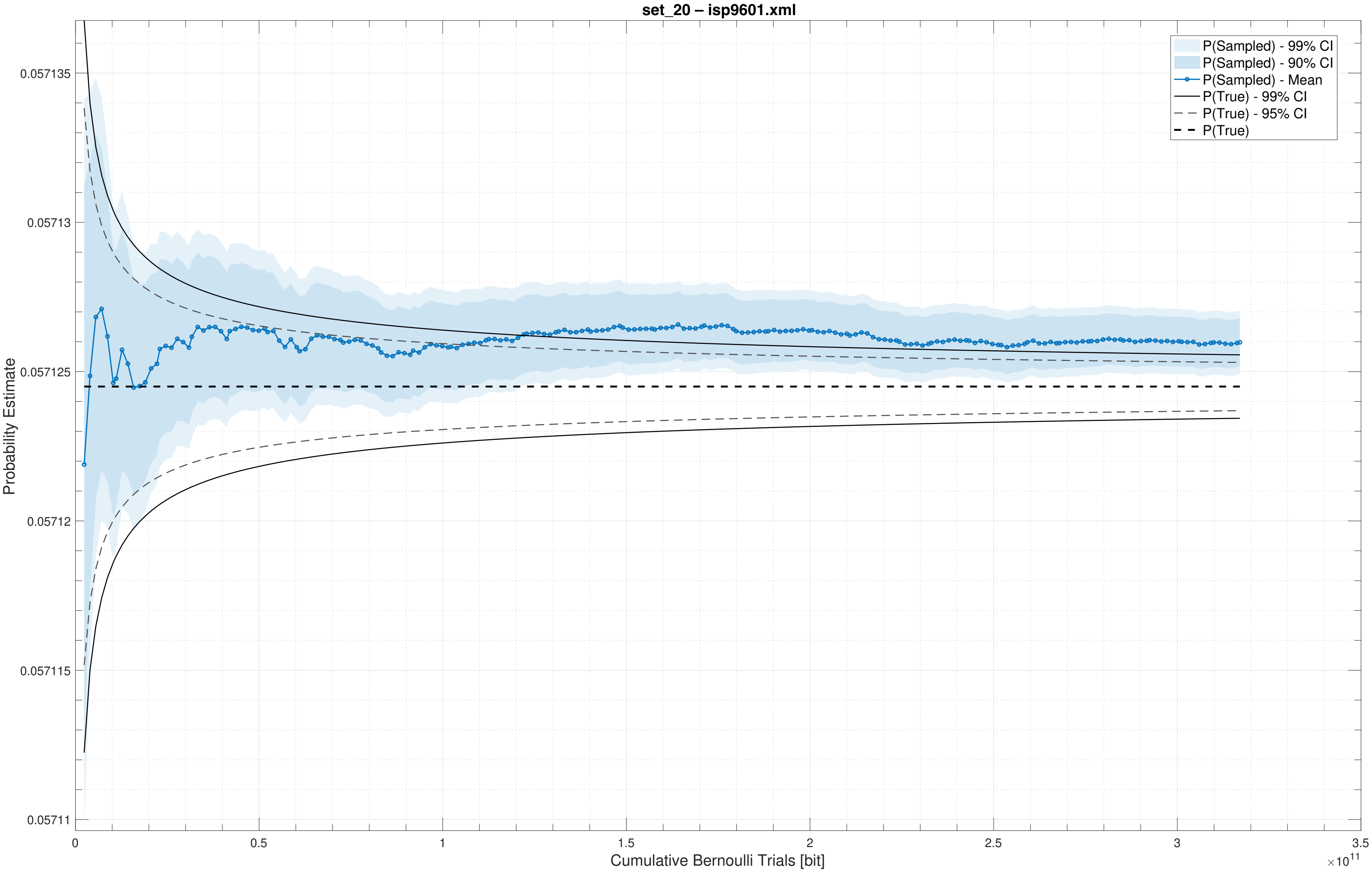


Figure A.35: Convergence trace for Aralia fault tree 35, 60 second timeout

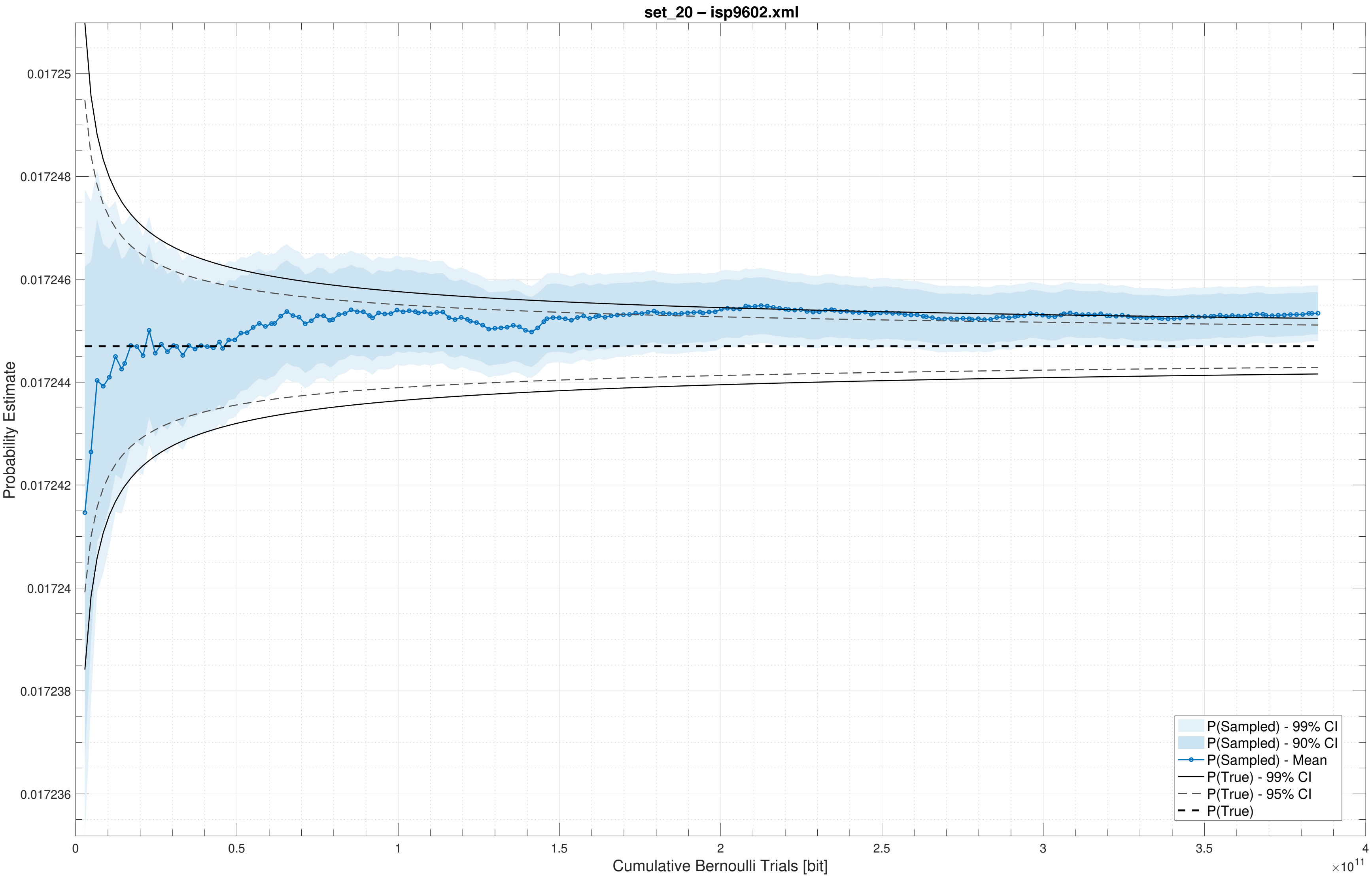


Figure A.36: Convergence trace for Aralia fault tree 36, 60 second timeout

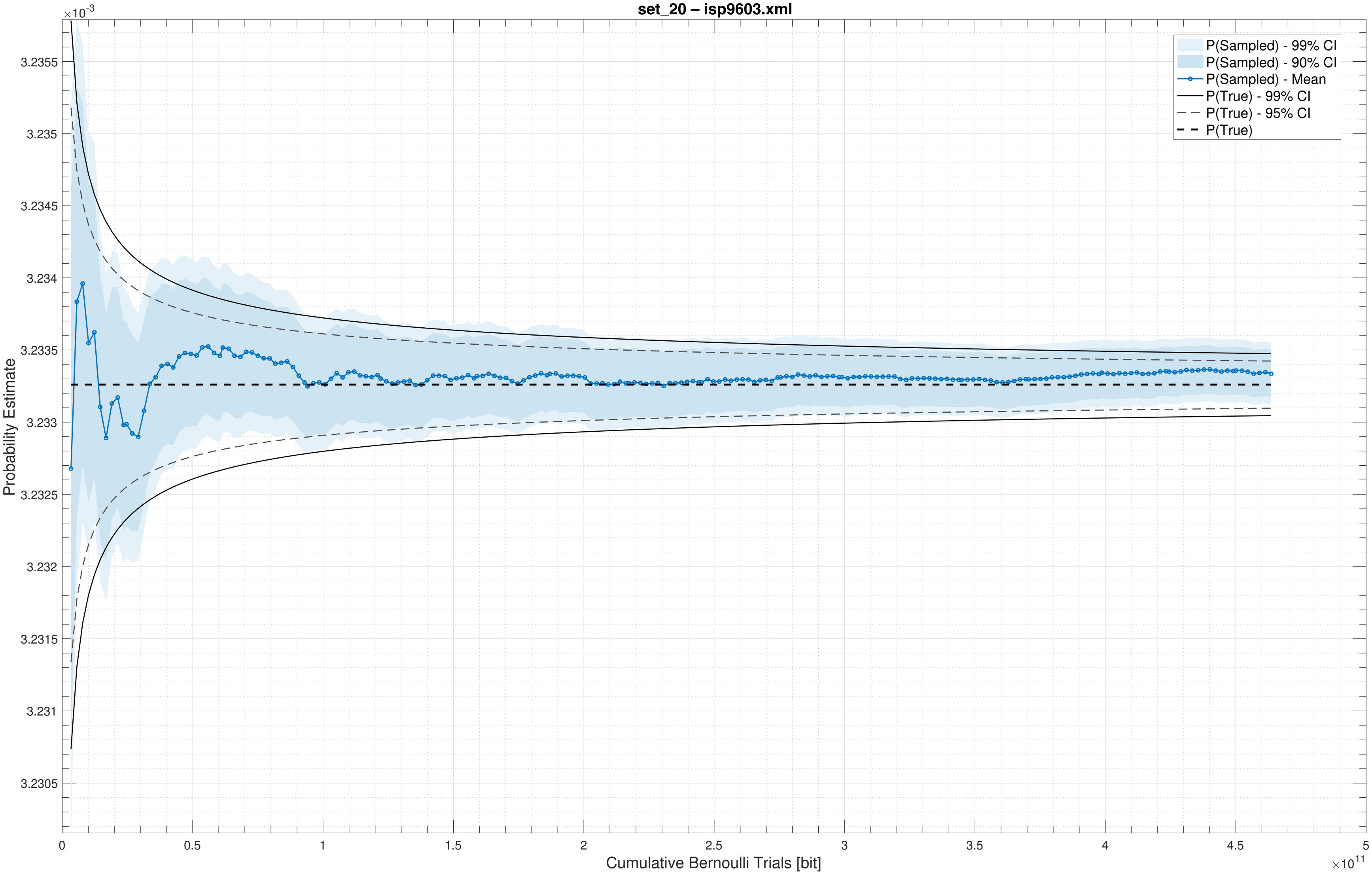


Figure A.37: Convergence trace for Aralia fault tree 37, 60 second timeout

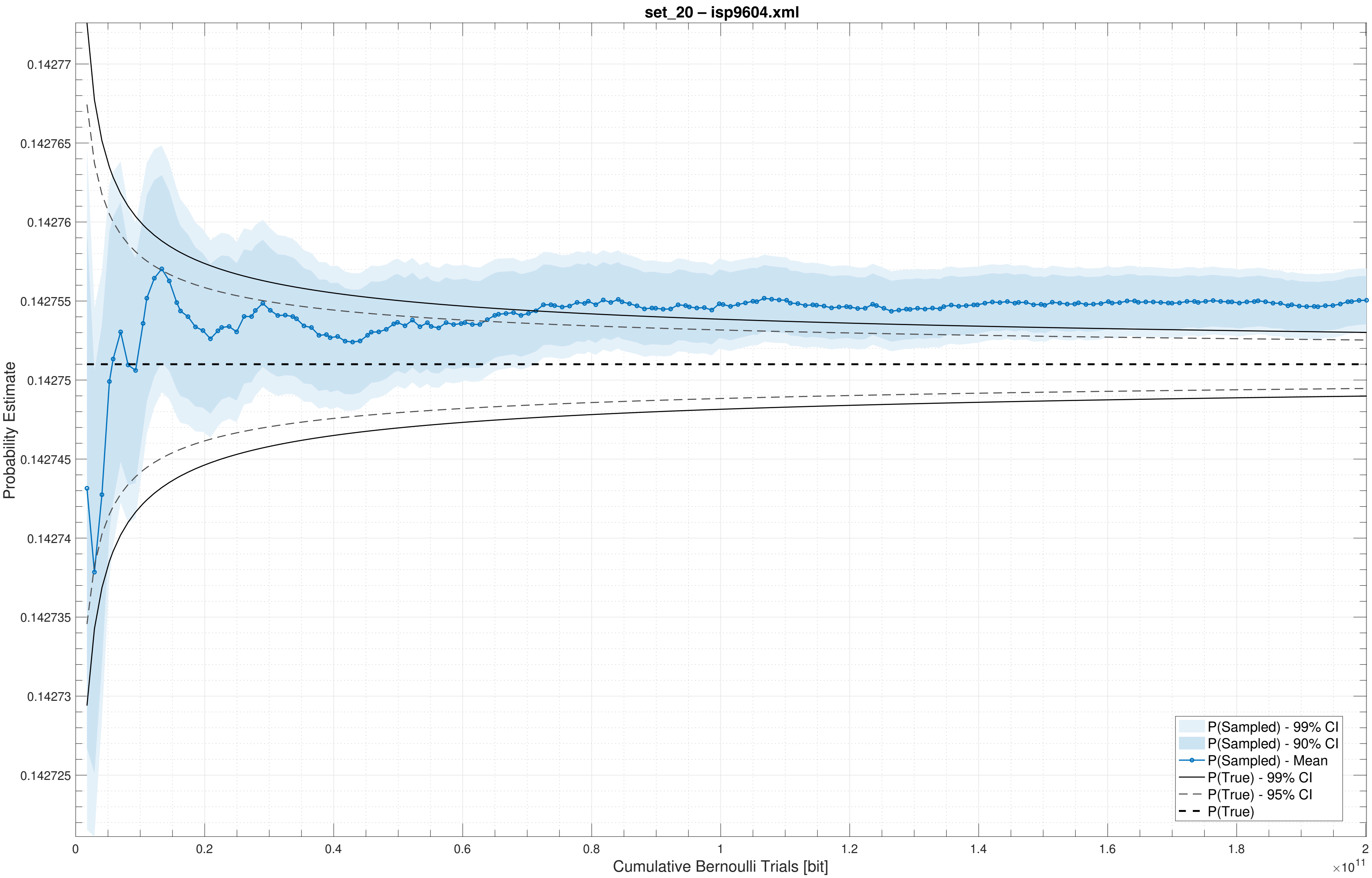


Figure A.38: Convergence trace for Aralia fault tree 38, 60 second timeout

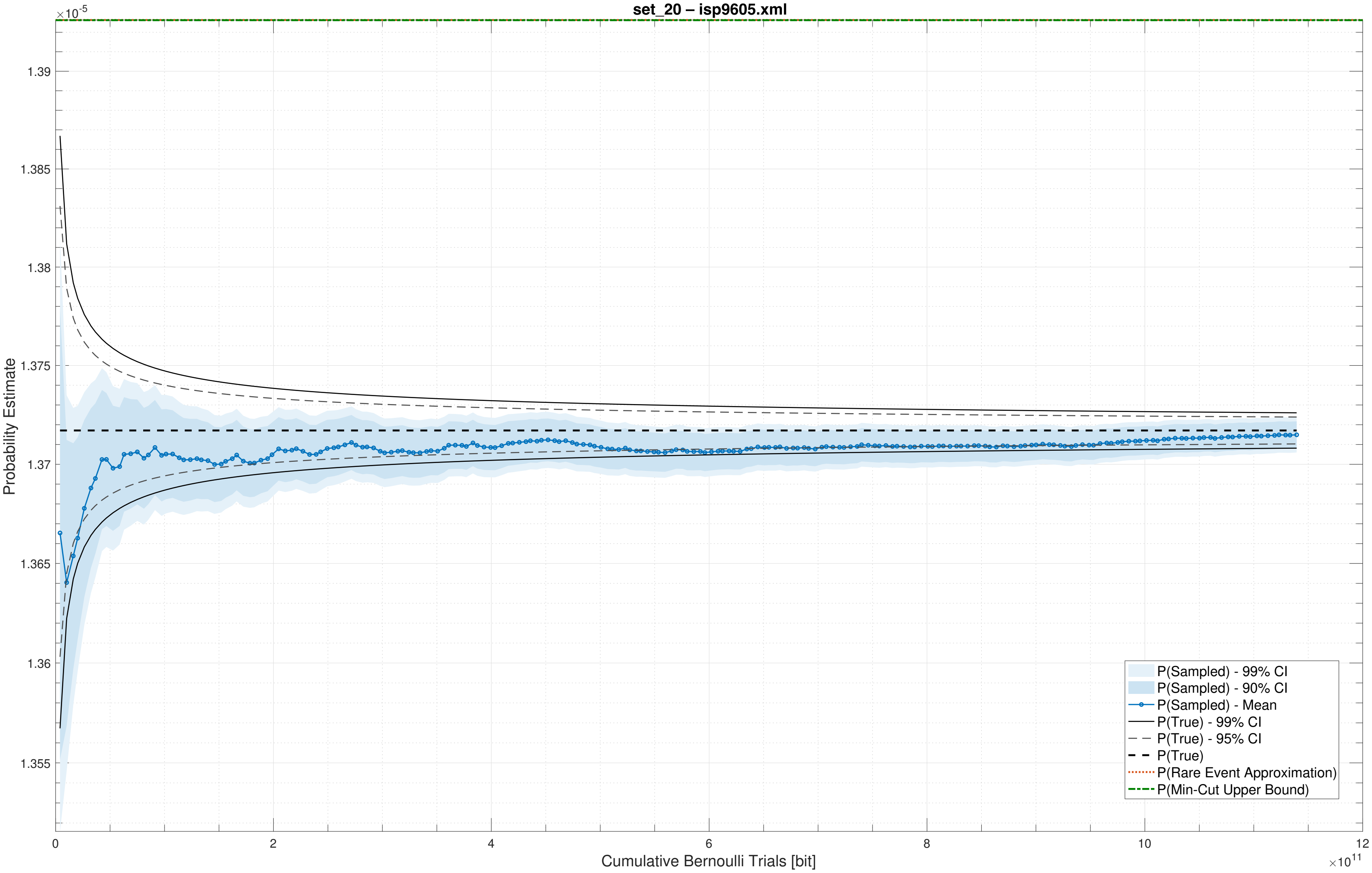


Figure A.39: Convergence trace for Aralia fault tree 39, 60 second timeout

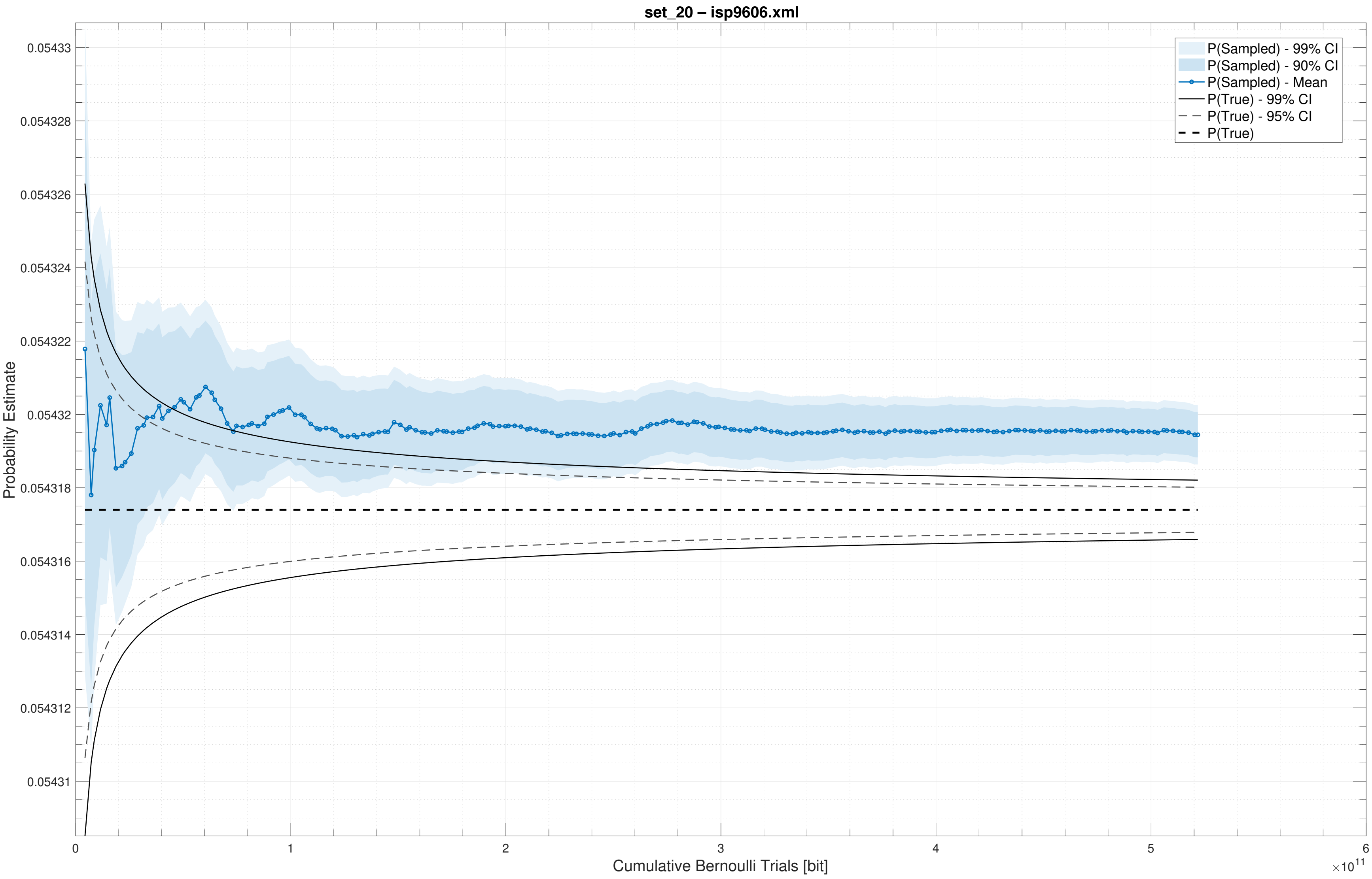


Figure A.40: Convergence trace for Aralia fault tree 40, 60 second timeout

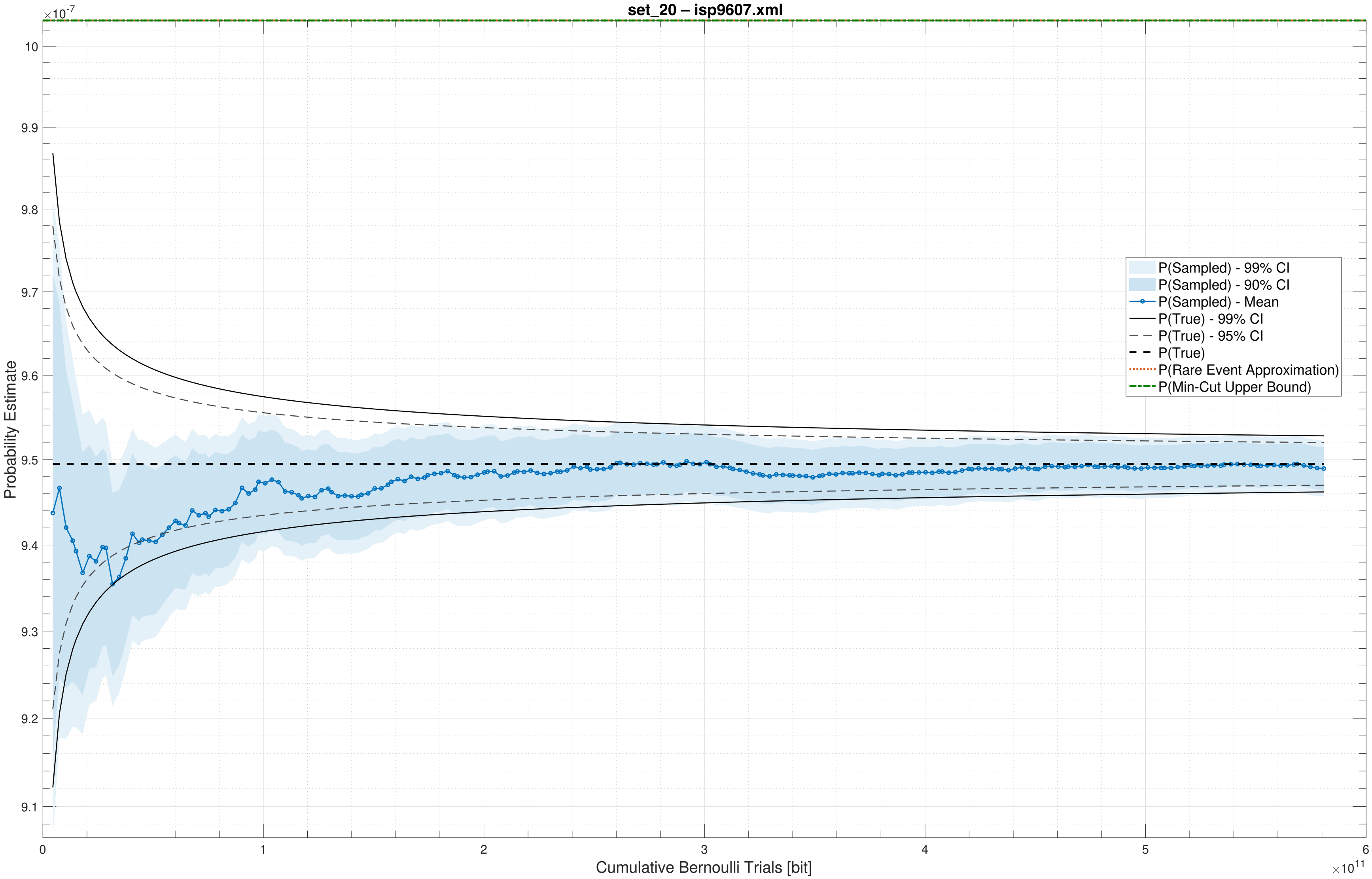


Figure A.41: Convergence trace for Aralia fault tree 41, 60 second timeout

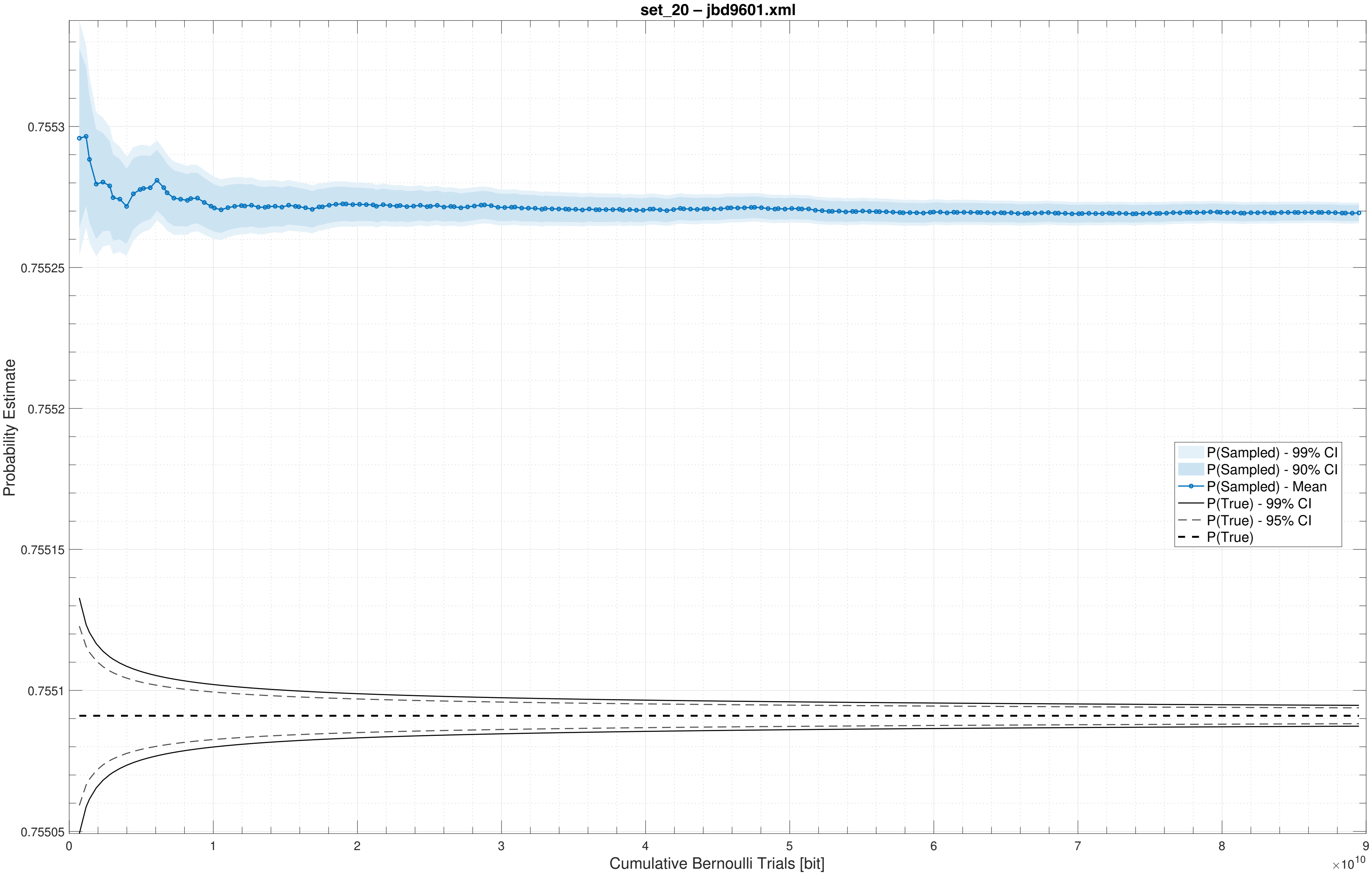


Figure A.42: Convergence trace for Aralia fault tree 42, 60 second timeout

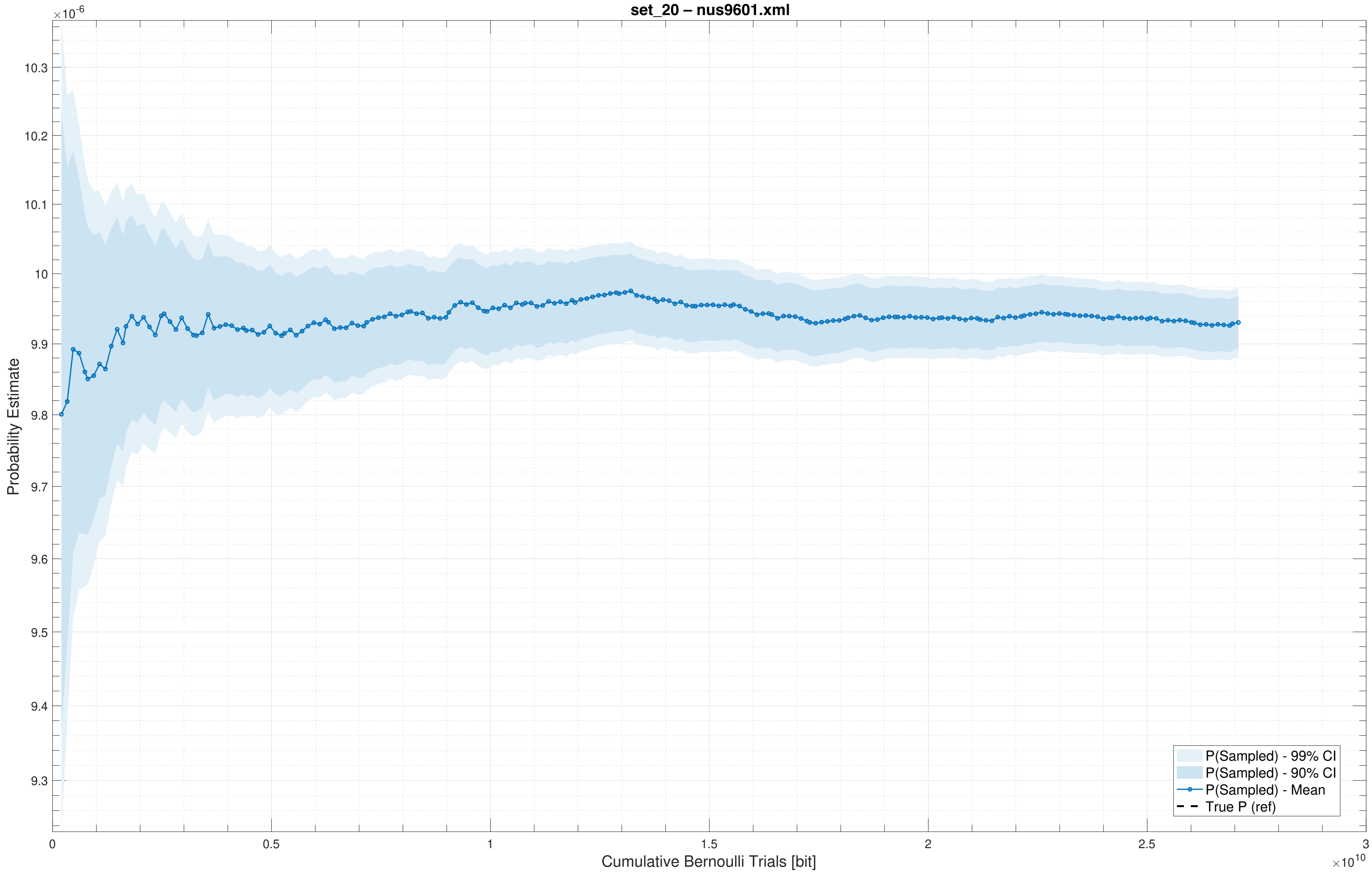


Figure A.43: Convergence trace for Aralia fault tree 43, 60 second timeout