Multi-Paradigm Modeling

Prof. William H. Sanders
Department of Electrical and Computer Engineering,
Information Trust Institute and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
whs@uiuc.edu

www.mobius.uiuc.edu
www.perform.csl.uiuc.edu
www.iti.uiuc.edu

Multi-Paradigm Modeling

- Multiple:
  - Modeling Formalisms
  - Model Composition Methods
  - Measures and Measure Specification Methods
  - Model Solution Methods
  - Model Connection Methods
- Developed by multiple research groups

Outline

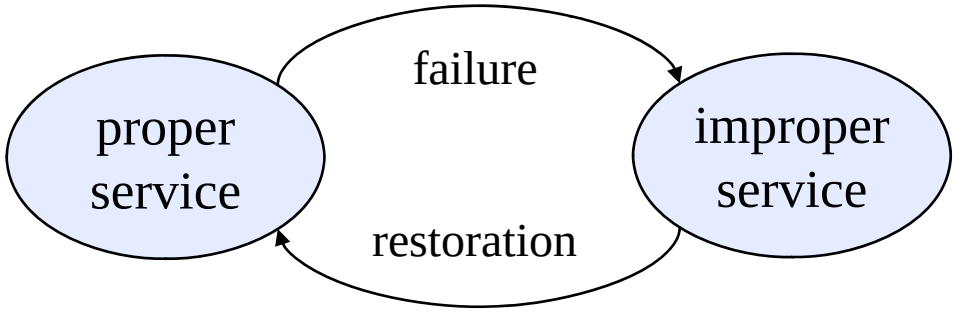
- Motivation: Dependability, Performance, and Performability Evaluation
- The need for multi-formalism, multi-solution evaluation frameworks
  - The Möbius modeling framework
- Model Specification Methods
  - Atomic Models (e.g. SANs and PEPA)
  - Reward Variable Specification
  - Model Composition (and state space generation)
  - Model Connection
- Model Solution Methods
  - Simulation
  - Analytic Methods
- Putting it all together

Outline

- Motivation: Dependability, Performance, and Performability Evaluation
- The need for multi-formalism, multi-solution evaluation frameworks
  - The Möbius modeling framework
- Model Specification Methods
  - Atomic Models (e.g. SANs and PEPA)
  - Reward Variable Specification
  - Model Composition (and state space generation)
  - Model Connection
- Model Solution Methods
  - Simulation
  - Analytic Methods
- Putting it all together

Motivation: Dependability Evaluation

- *Dependability* is the ability of a system to deliver a specified service.
- System service is classified as *proper* if it is delivered as specified; otherwise it is *improper*.
- System *failure* is a transition from proper to improper service.
- System *restoration* is a transition from improper to proper service.



⇒ The "properness" of service depends on the user's viewpoint!

Reference: J.C. Laprie (ed.), *Dependability: Basic Concepts and Terminology*, Springer-Verlag, 1992.

Examples of Specifications of Proper Service

- *k* out of *N* components are functioning.
- every working processor can communicate with every other working processor.
- every message is delivered within *t* milliseconds from the time it is sent.
- all messages are delivered in the same order to all working processors.
- the system does not reach an unsafe state.
- 90% of all remote procedure calls return within *x* seconds with a correct result.
- 99.999% of all telephone calls are correctly routed.

$\Rightarrow$ Notion of "proper service" provides a specification by which to evaluate a system's dependability.

Dependability Measures: Availability

Availability - quantifies the alternation between deliveries of proper and improper service.

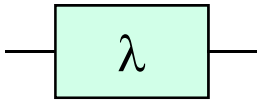
- $A(t)$ is 1 if service is proper at time t, 0 otherwise.
- $E[A(t)]$ (Expected value of $A(t)$) is the probability that service is proper at time t.
- $A(0,t)$ is the fraction of time the system delivers proper service during $[0,t]$.
- $E[A(0,t)]$ is the expected fraction of time service is proper during $[0,t]$.
- $P[A(0,t) > t^*]$ $(0 \leq t^* \leq 1)$ is the probability that service is proper more than $100t^*$% of the time during $[0,t]$.
- $A(0,t)_{t\to\infty}$ is the fraction of time that service is proper in steady state.
- $E[A(0,t)_{t\to\infty}]$, $P[A(0,t)_{t\to\infty} > t^*]$ as above.

Other Dependability Measures

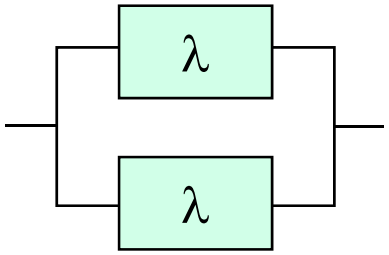
- *Reliability* - a measure of the continuous delivery of service
  - $R(t)$ is the probability that a system delivers proper service throughout $[0,t]$.
- *Safety* - a measure of the time to catastrophic failure
  - $S(t)$ is the probability that no catastrophic failures occur during $[0,t]$.
  - Analogous to reliability, but concerned with catastrophic failures.
- *Time to Failure* - measure of the time to failure from last restoration. (Expected value of this measure is referred to as *MTTF - Mean time to failure*.)
- *Maintainability* - measure of the time to restoration from last experienced failure. (Expected value of this measure is referred to as *MTTR - Mean time to repair*.)
- *Coverage* - the probability that, given a fault, the system can tolerate the fault and continue to deliver proper service.

Illustration of the Impact of Coverage on Dependability

- Consider two well-known architectures: simplex and duplex.

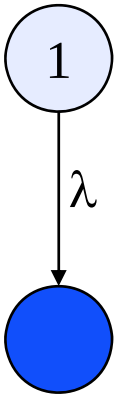


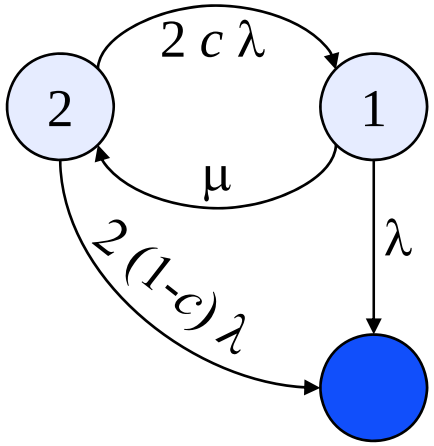
Simplex System



Duplex System

- The Markov model for both architectures is:

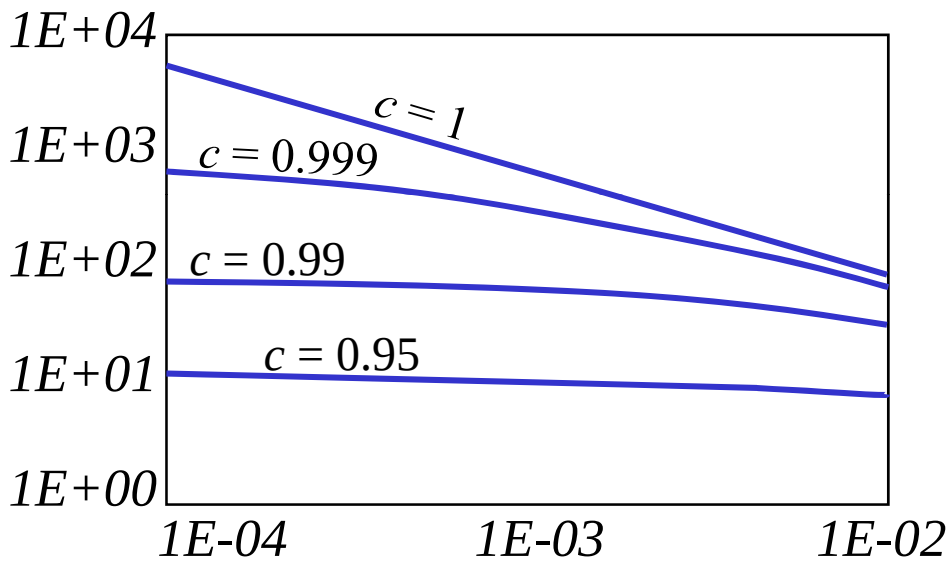




- The analytical expression of the MTTF can be calculated for each architecture using these Markov models.

Illustration of the Impact of Coverage, cont.

- The following plot shows the ratio of MTTF (duplex)/MTTF (simplex) for different values of coverage (all other parameter values being the same).
- The ratio shows the dependability gain by the duplex architecture.



- We observe that the coverage of the detection mechanism has a significant impact on the gain: a change of coverage of only 10^{-3} reduces the gain in dependability by the duplex system by a full order of magnitude.

Motivation: A Combined Performance/Dependability Concept - Performability

- *Performability* quantifies how well a system performs, taking into account behavior due to the occurrence of faults.
- It generalizes the notion of dependability in two ways:
  - includes performance-related impairments to proper service.
  - considers multiple levels of service in specification, possibly an uncountable number.
- Performability measures are truly user-oriented, quantifying performance as perceived by users.

Original reference: J. F. Meyer, "On Evaluating the Performability of Degradable Computing Systems," *Proceedings of the 8th International Symposium on Fault-Tolerant Computing*, Toulouse, France, June 1978, pp. 44-49.

THE MYTHICAL FIVE NINES. 99.999%. AS CLOSE TO PERFECT AS YOU CAN GET WITHOUT BREAKING SOME LAW OF NATURE.

For a server operating system, the five nines are a measure of reliability that translates into just over five minutes of server downtime per year.* For your business, that means servers are up and running when people need them. Of course, rumors of this 99.999% uptime usually start under ideal lab conditions. But where are these five nines when your business needs them? If you're using Microsoft® Windows® 2000 Server–based solutions, they may be closer than you think. Today Starbucks, FreeMarkets and MortgageRamp, an affiliate of GMAC Commercial Mortgage, are using Windows 2000 Server–based systems designed to deliver 99.999% server uptime. Of course, not all installations require this level of reliability, but one thing is for sure: The Windows 2000 Server family can help you get to the level of reliability you need. In fact, industry leaders such as Compaq, Hewlett-Packard, Unisys, Stratus and Motorola Computer Group can work with you to deliver solutions with up to five nines uptime. To learn more about server solutions you can count on, visit **microsoft.com/windows2000/servers** Software for the Agile Business.

Microsoft Windows 2000 Server Family

Microsoft

*This level of availability is dependent on many factors outside of the operating system, including other hardware and software technologies, mission-critical operational processes and professional services.

Outline

- Motivation: Dependability, Performance, and Performability Evaluation
- The need for multi-formalism, multi-solution evaluation frameworks
  - The Möbius modeling framework
- Model Specification Methods
  - Atomic Models (e.g. SANs and PEPA)
  - Reward Variable Specification
  - Model Composition (and state space generation)
  - Model Connection
- Model Solution Methods
  - Simulation
  - Analytic Methods
- Putting it all together

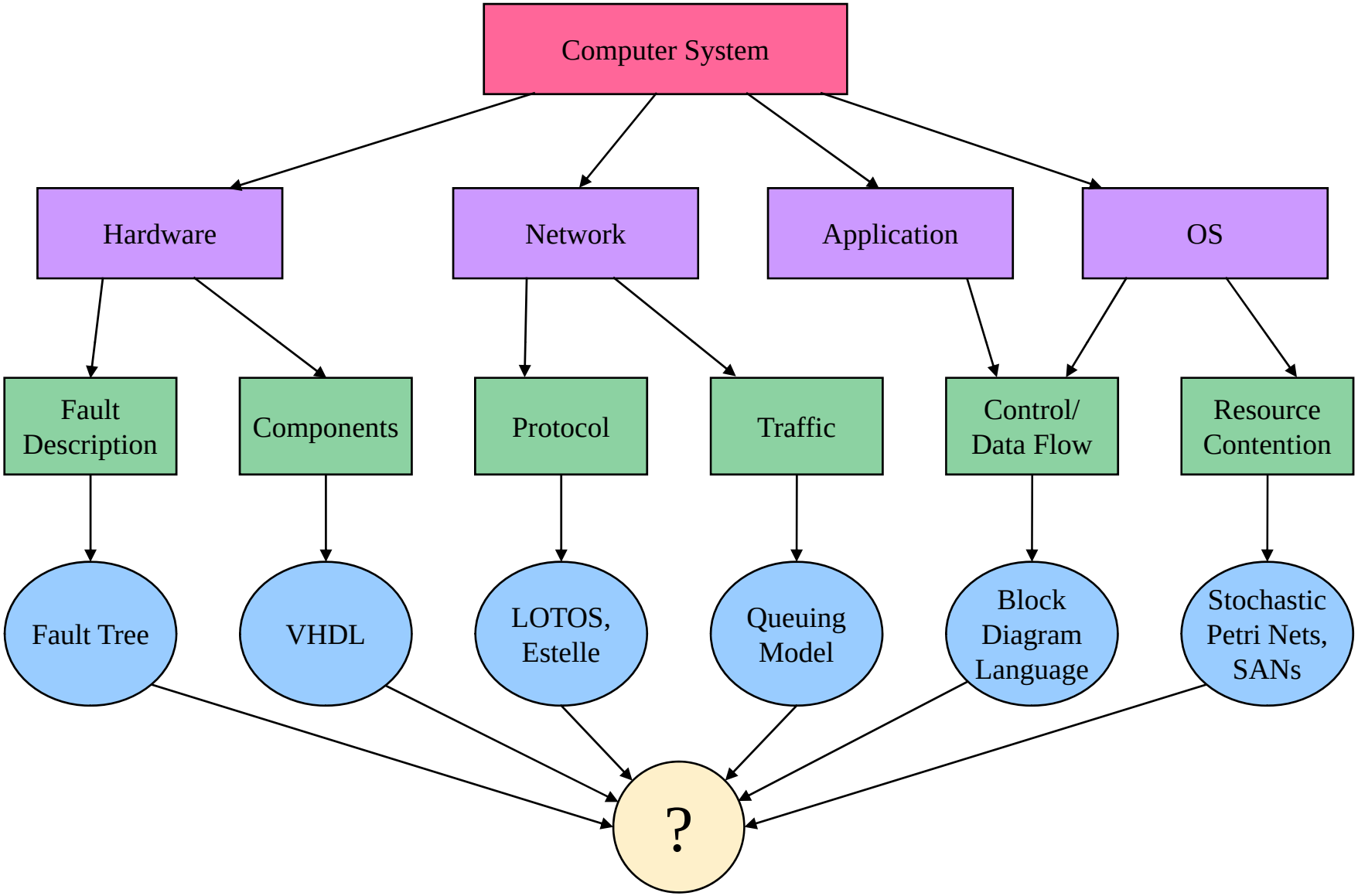
Integrated Modeling Framework Motivation

- No single formalism is best for representing all parts of a distributed computing/communication system
  - Computer hardware, networks, protocols, and applications each call for a different representation
  - Even within a "class" of application, different industry segments use very different ways of representing a particular design
- No single solution method is adequate to solve all models
  - Discrete-event simulation is efficient in many cases, but is extremely slow in others (e.g., significant, but rare events), or extreme system complexity)
- Research in new modeling methods and tools is significantly hampered by the close link between model specification and model solution methods, and the closed nature of existing tools

Modeling Complex System Behavior

- Modeling approach focuses on capturing system behaviors and then measuring desired system properties
- Supports system with complex system behaviors, such as:
  - Dynamic, state-dependant failure rates and probabilities
  - Correlated failures and repairs
  - Time- and state-dependent sequences of events
  - User-specified redundancy, fail-over, recovery, repair strategies
  - Multiple distribution functions for event delays
  - Custom behaviors defined by logical expressions

Example Heterogeneous Model



State of the Art: Single Formalism Tools

- Many performance/dependability evaluation tools have been developed that provide a single modeling formalism, and support multiple solution methods, e.g.,
  - Queueing networks, e.g., DyQN-Tool, HIT, LQNS, QNAP2, RESQ, RESQME. Most tools support both simulation and product-form based solutions.
  - Stochastic Petri nets and extensions, e.g., DSPNExpress, ESP, GreatSPN, HiQPN-Tool, QPN-tool, SPNP, SPN2MGM, SPNL, SURF-2, TimeNET, and *UltraSAN*. All tools support analytical/numerical solution; some support simulation.
  - Stochastic Process Algebras, e.g. EMPA, Dragon, PEPA Workbench, TIPPtool, Two Towers, and Spades. All tools support analytical/-numerical evaluation, some support simulation.
  - Other modeling approaches, sometimes tailored to a specific application domain, e.g., MARCA, DEPEND, Figaro, HARP, HIMAP, Peps, SAVE, SPE*ED, and TANGRAM-II.

$\Rightarrow$ In most cases, each tool has multiple model solution methods (recognizing the fact that no single solution method is sufficient in all cases), but a single model specification method.

State of the Art: Combination of Multiple Tools in a Single Software Environment

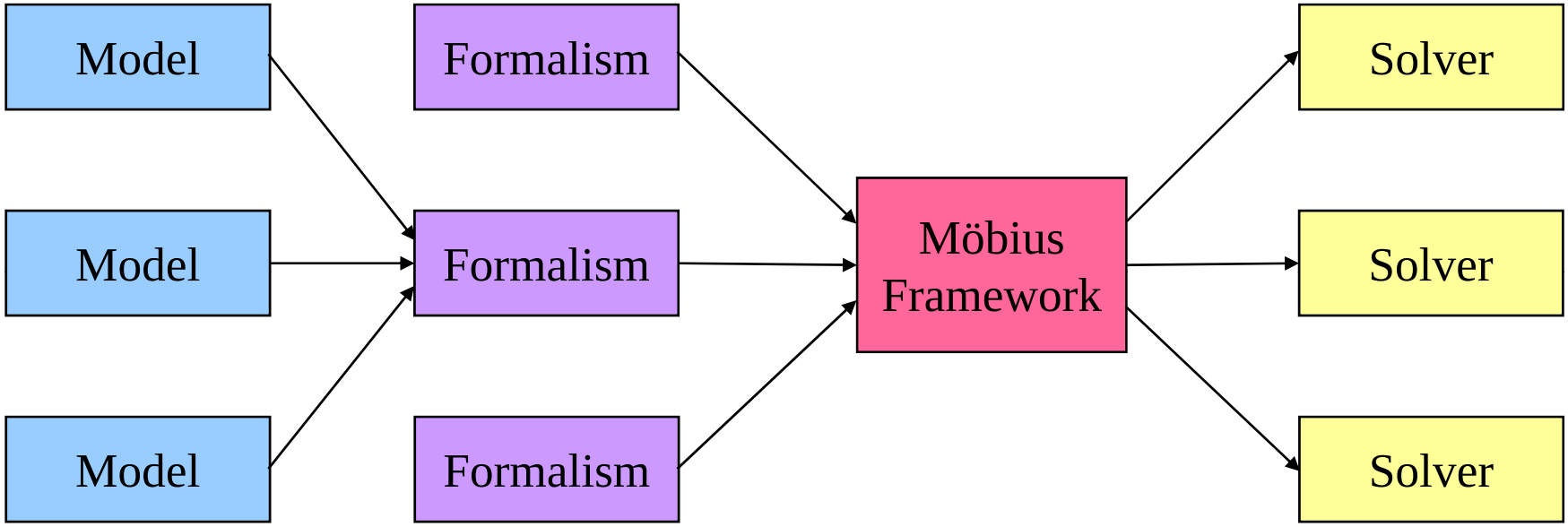
- Several tools have been constructed the facilitate the combination of multiple tools into a single environment, e.g.
  - IMSE (Integrated Modeling Support Environment) [Pooley 91]
    - Contains tools for modeling, workload analysis, and system specification
  - IDEAS (Integrated Design Environment for Assessment of Computer Systems and Communication Networks) [Fricks 96]
    - Provides user interface to multiple tools without requiring a user to learn multiple interface languages and output formats
  - Freud [van Moorsel 98]
    - Aims similar to those of ISME and IDEAS, but focuses on providing a uniform interface to a variety of web-enabled tools

⇒ Focus is on building a common graphical interface for accessing multiple tools and a common methods for reporting results.

State of the Art: Integrated Modeling Frameworks

- Integrated modeling frameworks aim to define an environment that can accommodate multiple modeling formalisms, one or more ways to combine models expressed in possibly different formalisms, and multiple model solution methods, e.g.
  - SHARPE [Sahner 86, Sahner 96 …]
    - Models expressed as combinatorial models, directed acyclic graphs, Markov and semi-Markov models, product-form queueing networks, and GSPNs can be solved, and can exchange results expressed as exponential-polynomial distribution functions
  - SMART [Ciardo 96, Ciardo 97 …]
    - Models expressed as SPNs, "software modeling language," and Markov chains can be solved, and can exchange results, possibly repeatedly, using fixed-point iteration. Implements symbolic representation of CTMC.
  - APNN toolbox [Bause 98]
    - Models converted to common "abstract PN notation" (APNN) and can be solved in a variety of quantitative and functional analysis methods.

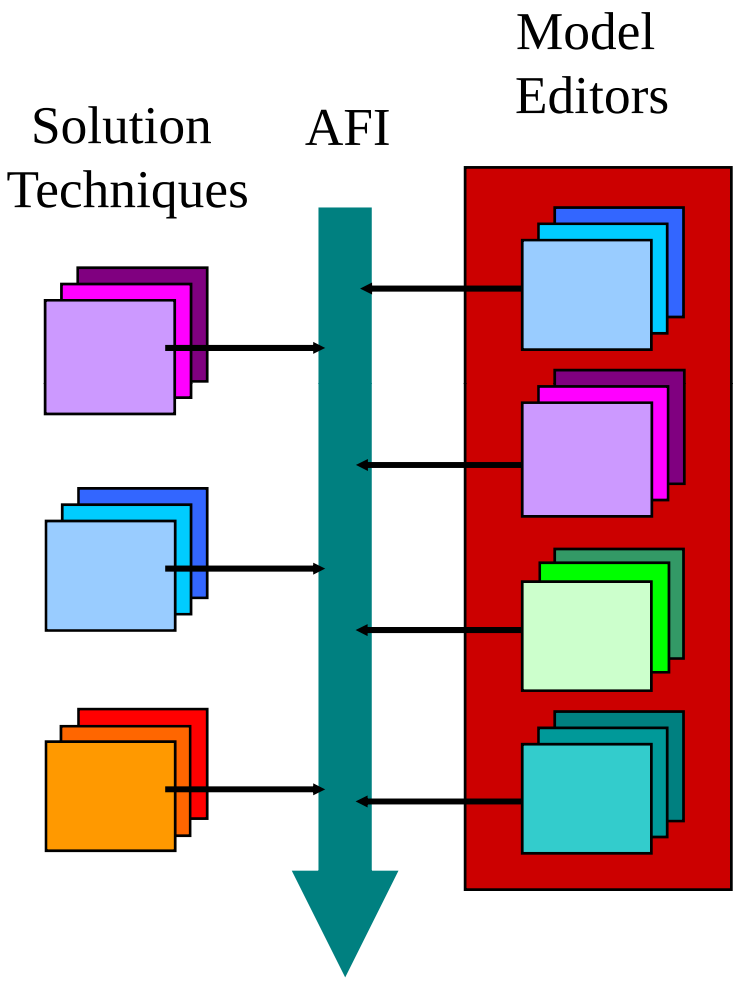
Möbius Framework



- Model: An abstract representation of some system
- Formalism: A modeling language
- Framework: A “language” in which modeling languages may be expressed

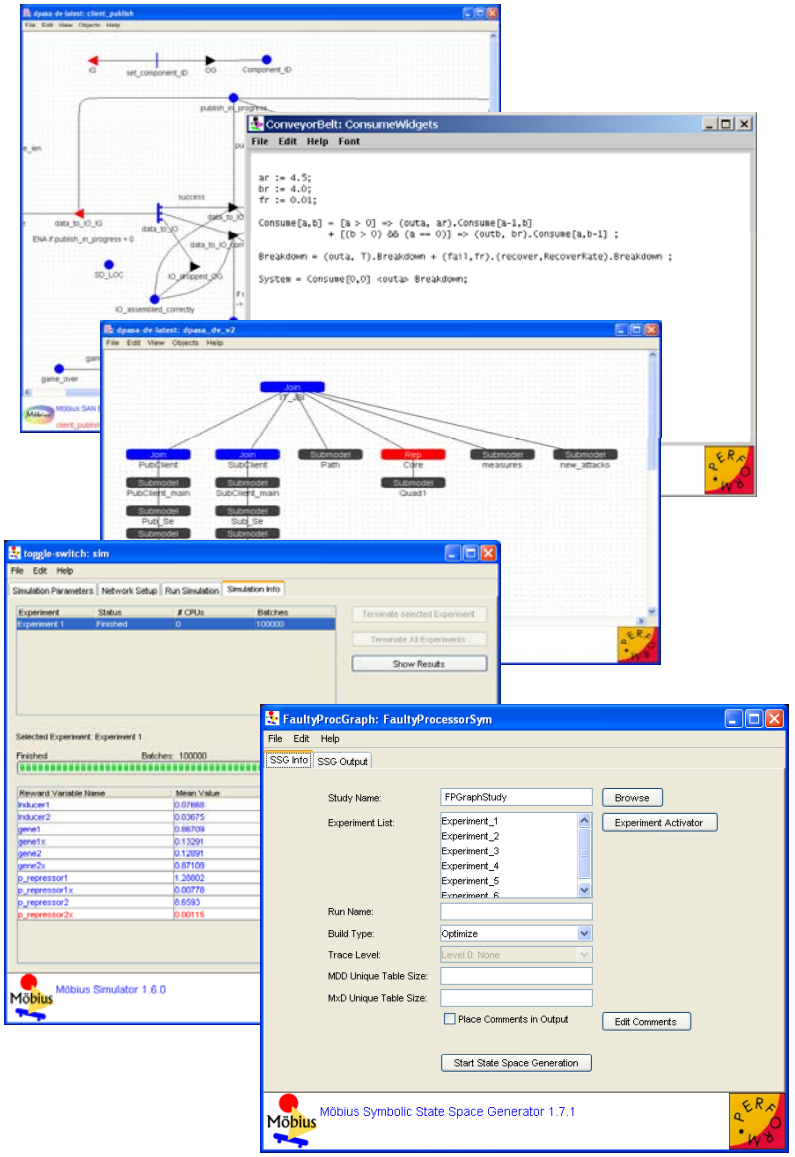
Möbius Framework

- Open-ended, flexible modeling environment.
- Multi-formalism and multi-solution framework
  - Several model specification languages
  - Includes both simulation and numerical analysis solution techniques
  - Supports additional plug-in modules using abstract functional interface (AFI).

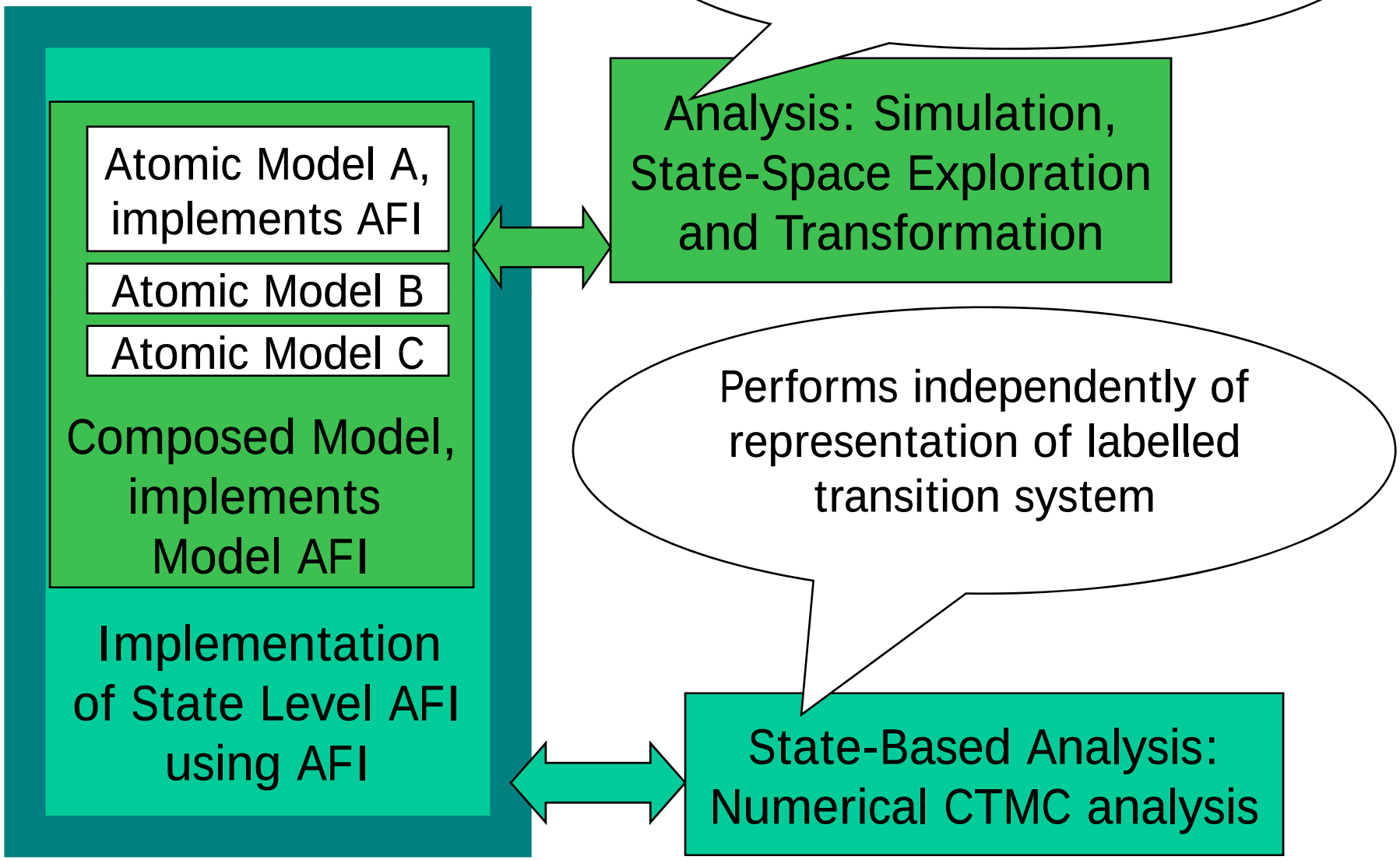


(Partial list of) Existing Möbius Modules

- **Atomic Model formalisms**
  - Represent detailed component models
  - Stochastic Activity Networks, Buckets & Balls (Markov Chains), PEPA, Fault Trees, MODEST, AADL
- **Compositional formalisms**
  - Combine atomic models
  - Rep Join, Graph, Action Synchronization
- **Reward variables**
  - Define measures to evaluate
  - Rate rewards, Impulse rewards
- **Experiment Studies**
  - Vary model parameters
  - Set and Range studies
- **Solution techniques**
  - Distributed Simulation
  - Analytical Solutions



Model- and State-Level Abstract Functional Interfaces



Model Support of the Abstract Functional Interface: State Variables, Actions, and Properties

- Formally, a model in the Möbius framework is a set of "state variables," a set of "actions," and set of "properties"
- State variables "contain" information about the state of the system being modeled
  - They have a type, which defines their "structure"
  - They have a value, which defines the "state" of the variable
- Actions prescribe how the value of state variables may change as a function of time
- Properties specify characteristics that may effect the solution of a model
- Other models and solvers may request information regarding or change to state of a model's state variables, actions, and groups via the abstract functional interface
- The format of this information is determined by the structure of a model's state variables and attributes of its actions

State Variable Specification in the Möbius Framework

- The set of all state variable types is denoted T.
- The set T is constructed by repeated application of the following rules:

$Z \in T.$

$\Re \in T.$

$S \in T.$ Set of names of state variables

If $t \in T$, then $2^t \subset T.$ Restriction of a type

If $t \in T$, then $2^t \in T.$ Sets of types (unordered)

If $t_1, t_2 ..., t_n \in T$, then $t_1 \times t_2 \times ... \times t_n \in T.$ Ordered lists of types

Action Specification in the Möbius Framework

- An action's attributes specify how and when it changes state:

$$\text{Enabled}: \Sigma \rightarrow \{\text{true}, \text{false}\}$$

$$\text{Delay} : \Sigma \rightarrow (\Re \geq \rightarrow [0,1])$$

$$\text{Effort} : \Sigma \rightarrow (\Re \geq \rightarrow [0,1])$$

$$\text{Interrupt}: \Sigma \rightarrow \{\text{true}, \text{false}\}$$

$$\text{Rank} : \Sigma \rightarrow Z^{+} \cup \{\textit{Undefined}\}$$

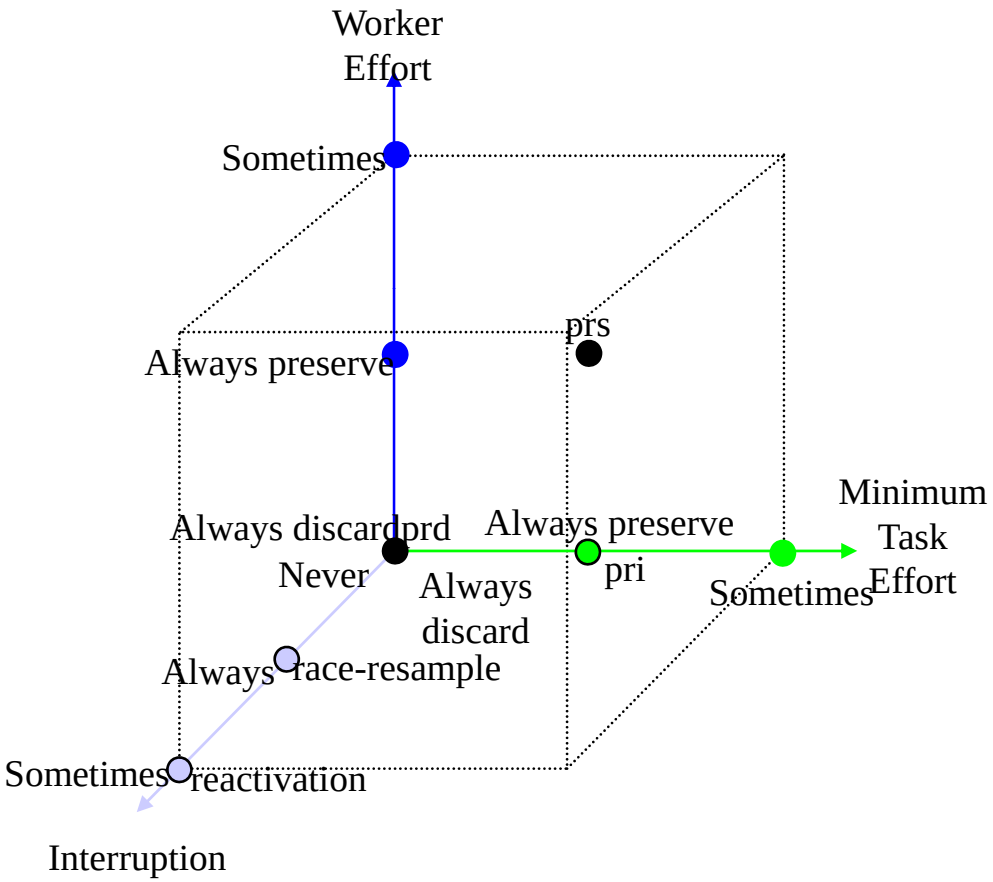
$$\text{Weight} : \Sigma \rightarrow \Re^{\geq} \cup \{\textit{Undefined}\}$$

$$\text{Complete} : \Sigma \rightarrow \Sigma$$

$$\text{Policy}: \Sigma \rightarrow \{\text{DDD}, ..., \text{PPP}\}$$

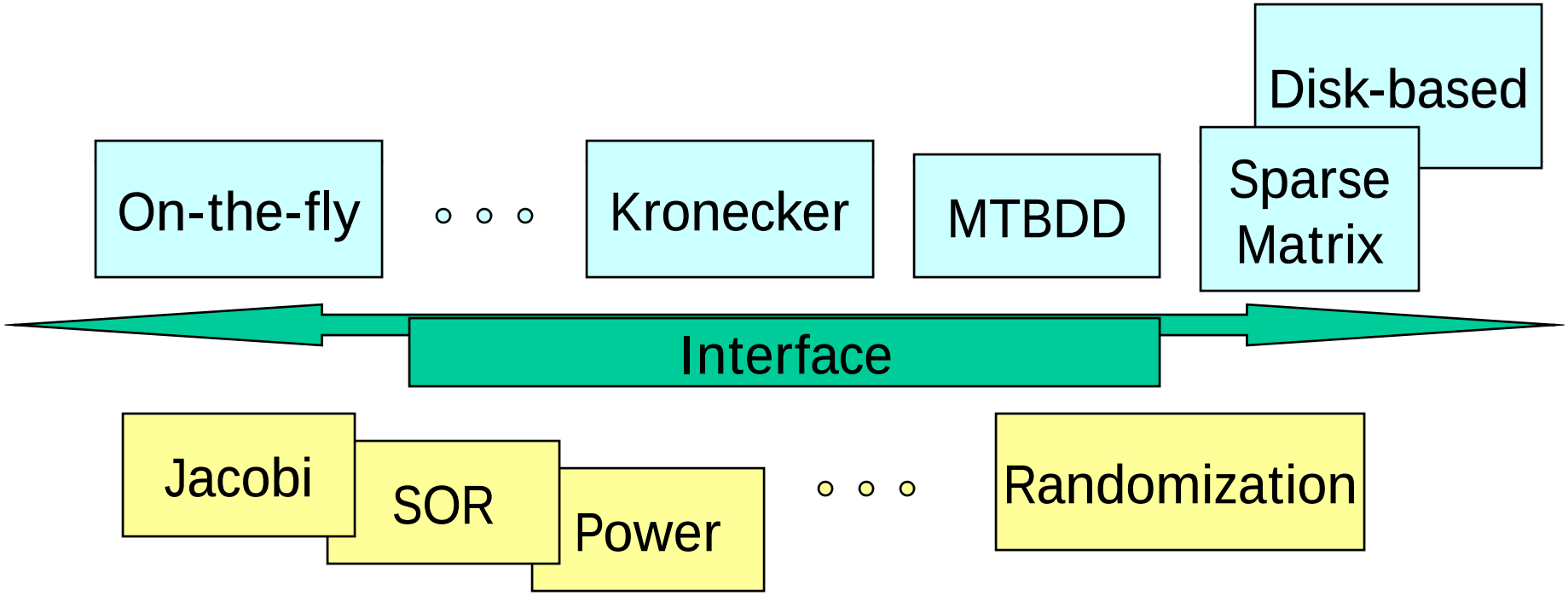
Model Execution Policies

- Models change state by the firing of actions, according to formalism-specific rules hidden by AFI
- The execution policy is defined on a per action and per state basis
  - Actions can be interrupted, reset, continue with a different distribution, or continue with time same distribution
  - Generalizes and unifies existing execution policies
  - Details can be in Deavours and Sanders, PNPM 2001.



State-Level Abstract Functional Interface Motivation

- Many ways towards few numerical procedures
  - Variety of modeling formalisms
  - Variety of CTMC representations



Goal: Interface to support common usage of different representations

Definition of State-Level AFI

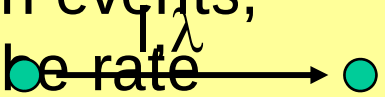
- General idea: Represent modeling formalism abstractly as a labeled transition system (LTS):

S: Set of states

δ: State transition relation with labels l and real value λ

Label l is used to distinguish events,

Value λ is used to carry the rate



In addition:

- Rate rewards per state
- Impulse rewards per transition in δ

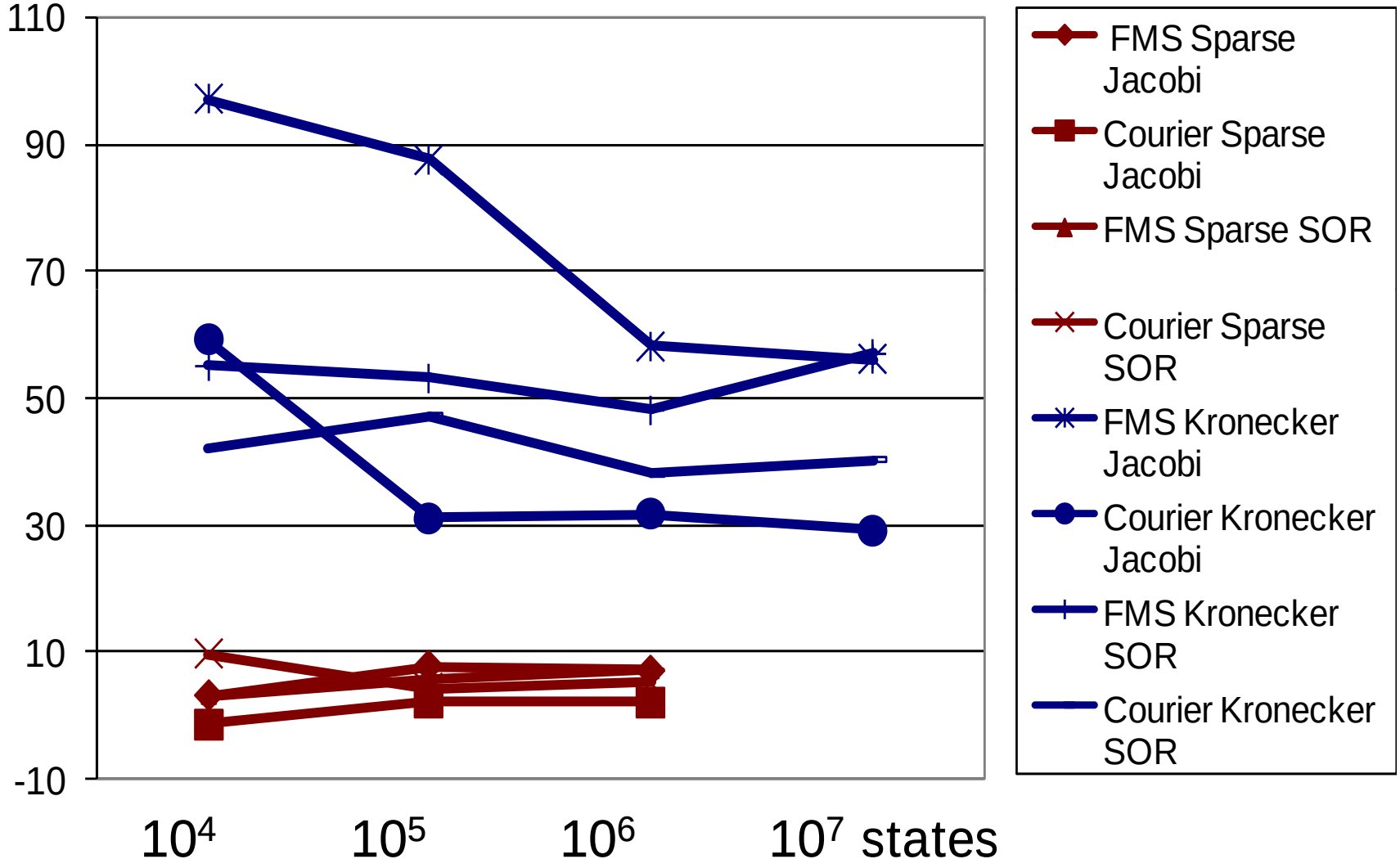
- The interface exports information of concerning the LTS in an OO way

State-Level AFI Functionality

- Container classes:
  - getRow all (nonzero) elements of a row
  - getColumn all (nonzero) elements of a column
  - getAllEdges all (nonzero) elements of a matrix
- Each of the methods returns a container which provides an iterator class to access its elements one after the other
- Elements are transition objects derived from a template class which allows access via functions:

```
StateType &row() ;          StateType &col();
RateType &rate();           LabelType &label();
RateType &reward(int RewardNumber) ;
```

Observed Performance, Slowdown Percentage per Iteration Step

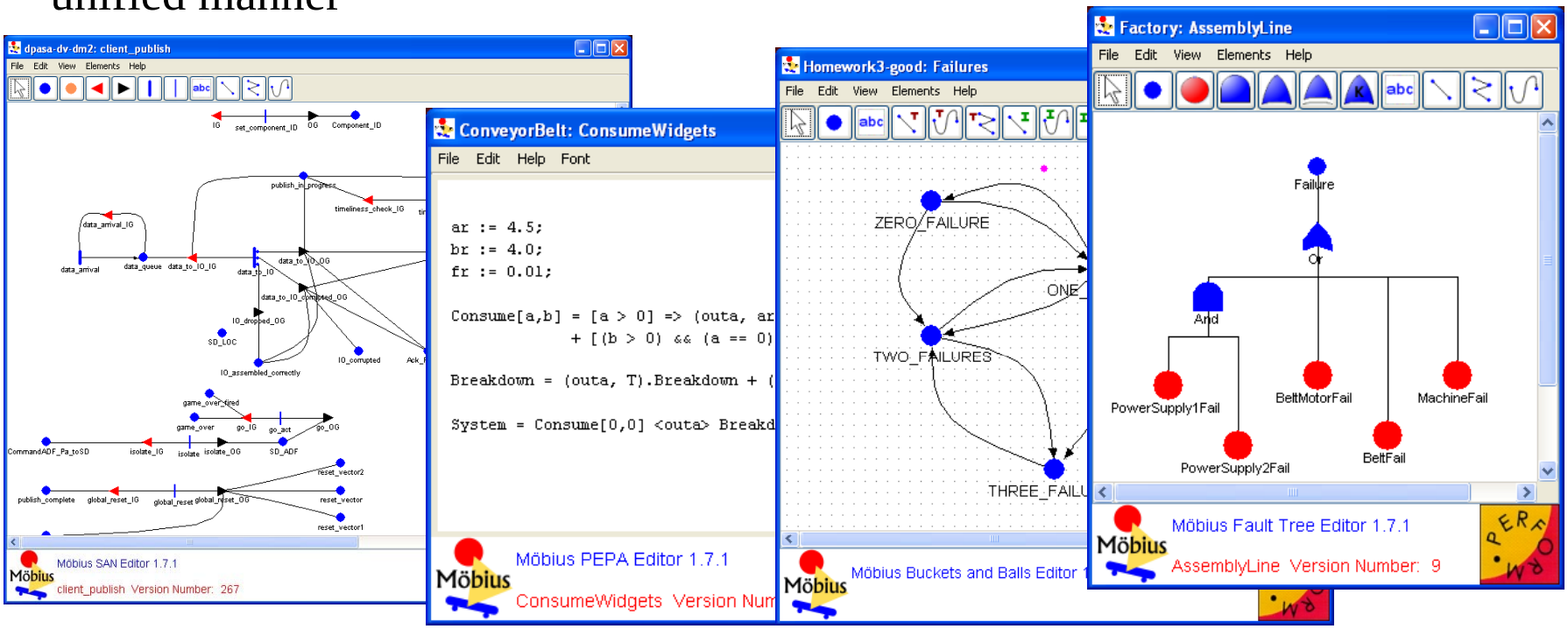


Outline

- Motivation: Dependability, Performance, and Performability Evaluation
- The need for multi-formalism, multi-solution evaluation frameworks
  - The Möbius modeling framework
- Model Specification Methods
  - Atomic Models (e.g. SANs and PEPA)
  - Reward Variable Specification
  - Model Composition (and state space generation)
  - Model Connection
- Model Solution Methods
  - Simulation
  - Analytic Methods
- Putting it all together

Atomic Model Representation

- Multiple modeling languages available, including:
  - Stochastic Activity Networks ('SANs'), PEPA (textual-based process algebra), Markov chains, Fault trees,
  - Parameters of the model can be specified variables and set at analysis time.
- Facilitate modeling of hardware, software, protocols, and application in a unified manner



Notes on Stochastic Petri Nets

- SPNs are much easier to read, write, modify, and debug than Markov chains.
- SPN to Markov chain conversion can be automated to afford numerical solutions to Markov chains.
- Most SPN formalisms include a special type of arc called an *inhibitor arc*, which enables the SPN if there are zero tokens in the associated place, and the identity (do nothing) function. Example: modify SPN to give writes priority.
- Limited in their expressive power: may only perform +, -, >, and test-for-zero operations.
- These very limited operations make it very difficult to model complex interactions.
- Simplicity allows for certain analysis, e.g., a network protocol modeled by an SPN may detect deadlock (if inhibitor arcs are not used).
- More general and flexible formalisms are needed to represent real systems.

Stochastic Activity Networks

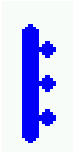
The need for more expressive modeling languages has led to several extensions to stochastic Petri nets. One extension that we will examine is called *stochastic activity networks*. Because there are a number of subtle distinctions relative to SPNs, stochastic activity networks use different words to describe ideas similar to those of SPNs.

Stochastic activity networks have the following properties:

- A general way to specify that an activity (transition) is enabled
- A general way to specify a completion (firing) rule
- A way to represent zero-timed events
- A way to represent probabilistic choices upon activity completion
- State-dependent parameter values
- General delay distributions on activities

SAN Symbols

Stochastic activity networks (hereafter SANs) have four new symbols in addition to those of SPNs:

- Input gate:  used to define complex enabling predicates and completion functions
- Output gate:  used to define complex completion functions
- Cases:  (small circles on activities) used to specify probabilistic choices
- Instantaneous activities: used to specify zero-timed events

SAN Enabling Rules

An input gate has two components:

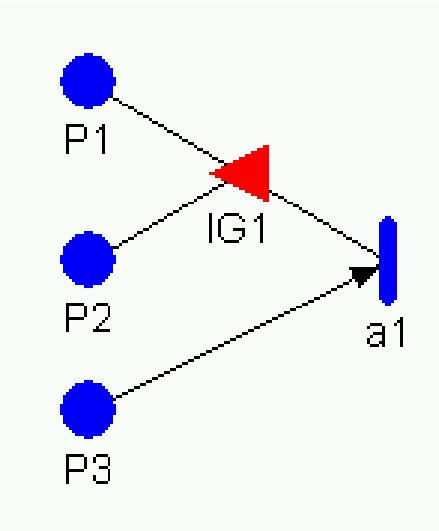
- enabling_function (state) → boolean; also called the *enabling predicate*
- input_function(state) → state; rule for changing the state of the model

An activity is *enabled* if for every connected input gate, the enabling predicate is true, and for each input arc, the number of tokens in the connected place ≥ number of arcs.

We use the notation *MARK*(*P*) to denote the number of tokens in place *P*.

Example SAN Enabling Rule

Example:



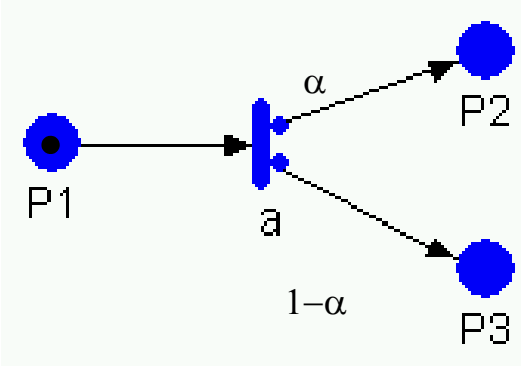
IG1 Predicate:

```
if((MARK(P1)>0 && MARK(P2)==0)||
      (MARK(P1)==0 && MARK(P2)>0))
            return 1;
else return 0;
```

Activity *a*1 is enabled if *IG*1 predicate is true (1) and *MARK*(*P*3) > 0.
(Note that in *Möbius*, "1" is used to denote true.)

Cases

Cases represent a probabilistic choice of an action to take when an activity completes.



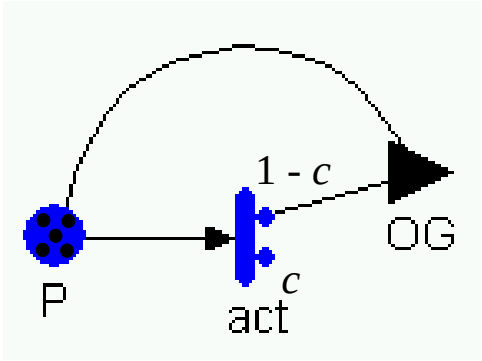
When activity *a* completes, a token is removed from place *P*1, and with probability α, a token is put into place *P*2, and with probability 1 - α, a token is put into place *P*3.

Note: cases are numbered, starting with 1, from top to bottom.

Output Gates

When an activity completes, an output gate allows for a more general change in the state of the system. This output gate function is usually expressed using pseudo-C code.

Example

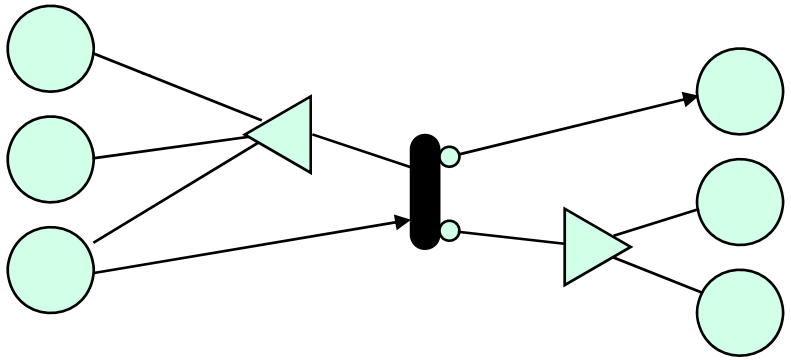


OG Function

```
MARK(P) = 0;
```

Instantaneous Activities

Another important feature of SANs is the instantaneous activity. An *instantaneous activity* is like a normal activity except that it completes in zero time after it becomes enabled. Instantaneous activities can be used with input gates, output gates, and cases.



Instantaneous activities are useful when modeling events that have an effect on the state of the system, but happen in negligible time, with respect to other activities in the system, and the performance/dependability measures of interest.

SAN Terms

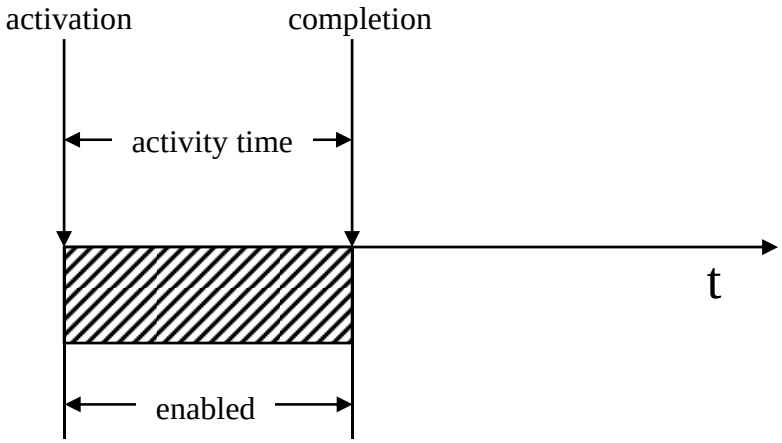
1. *activation* - time at which an activity begins

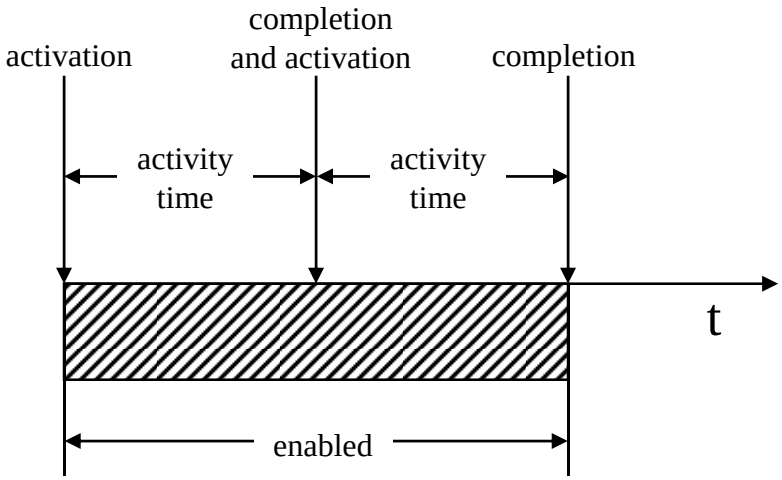
2. *completion* - time at which activity completes

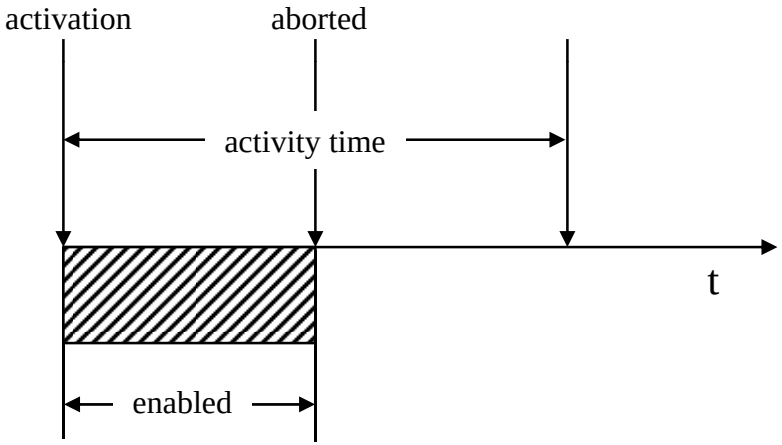
3. *abort* - time, after activation but before completion, when activity is no longer enabled

4. *active* - the time after an activity has been activated but before it completes or aborts.

Illustration of SAN Terms







Completion Rules

When an activity *completes*, the following events take place (in the order listed), possibly changing the marking of the network:

1. If the activity has cases, a case is (probabilistically) chosen.
2. The functions of all the connected input gates are executed (in an unspecified order).
3. Tokens are removed from places connected by input arcs.
4. The functions of all the output gates connected to the chosen case are executed (in an unspecified order).
5. Tokens are added to places connected by output arcs connected to the chosen case.

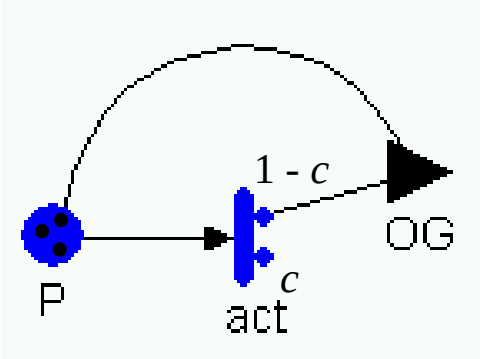
Ordering is important, since effect of actions can be marking-dependent.

Marking Dependent Behavior

Virtually every parameter may be any function of the state of the model. Examples of these are

- rates of exponential activities
- parameters of other activity distributions
- case probabilities

An example of this usefulness is a model of three redundant computers where the coverage (probability that a single computer crashing crashes the whole system) increases after a failure.

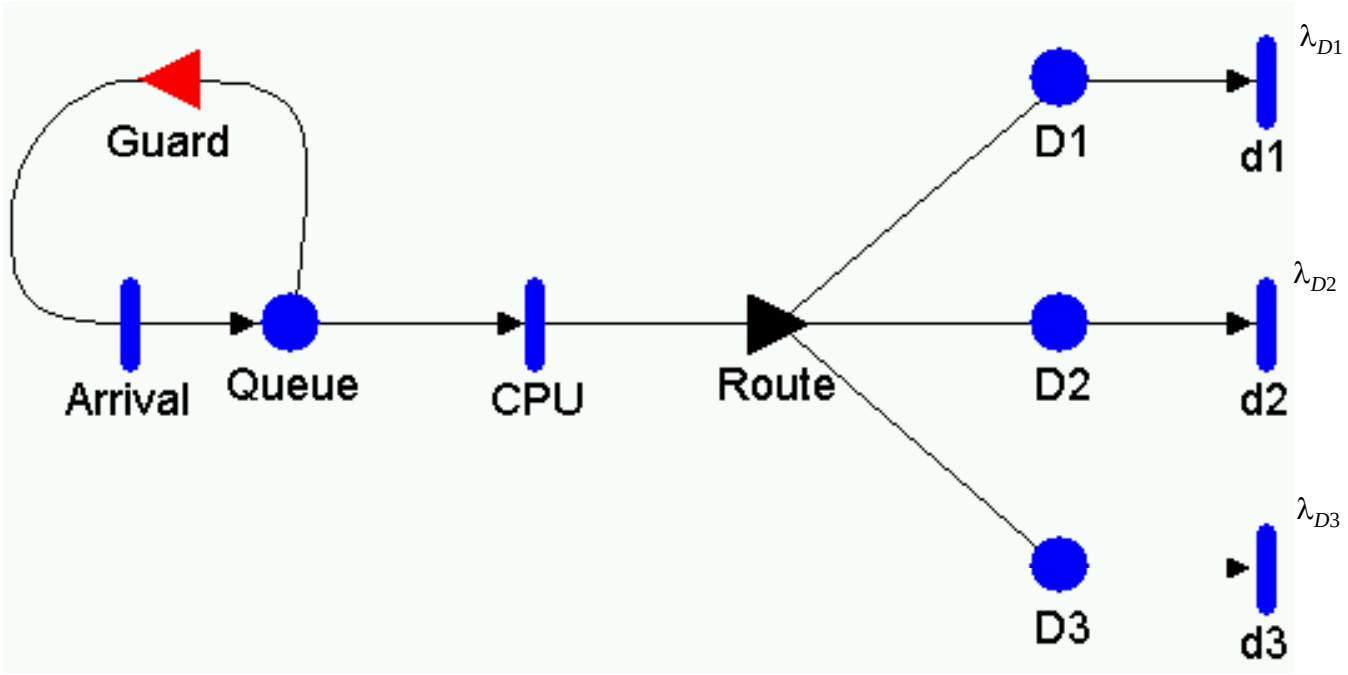


	a
case 1	0.1 + 0.02 * MARK(P)
case 2	0.9 – 0.02 * MARK(P)

Example Problem

- A database server is composed of a compute server and three file servers, and can queue up to N_c requests at a time (including the one in service).
- Requests arrive at rate λ_a and spend on average $1/\lambda_{CPU}$ time at the compute server being processed.
- The request is then forwarded to the file server that has the fewest outstanding requests.
- Requests are processed at a rate of λ_{D1}, λ_{D2}, and λ_{D3} for file servers D1, D2, and D3 respectively.
- File server buffers may hold at most N_f requests (including requests in service); if all buffers are full, the request is discarded.

SAN Representation of Example Database Problem



Gate Functions for SAN

Gate	*Definition*
Guard	Predicate *MARK(Queue) < GLOBAL_S(Nc)* Function *;*

Gate	*Definition*
Route	Function *if (MARK(D1) == GLOBAL_S(Nf) &&* *MARK(D2) == GLOBAL_S(Nf) &&* *MARK(D3) == GLOBAL_S(Nf))* *return;* *if (MARK(D1) < MARK(D2)) {* *if (MARK(D1) < MARK(D3)) {* *MARK(D1)++;* *} else {* *MARK(D3)++;* *}* *} else {* *if (MARK(D2) < MARK(D3)) {* *MARK(D2)++;* *} else {* *MARK(D3)++;* *}* *}*

A Summary of PEPA

- PEPA stands for "*Performance Evaluation Process Algebra*" [Hillston 96]
- Primitive process algebra *actions* become timed PEPA *activities*:

```
enter.exit.Spec  <----->  (enter,r).(exit,s).Spec
```

- r and s are the parameters of exponentially distributed random variables which determine the time it takes for each activity to complete
  - What are the primitives for building PEPA models?

PEPA Combinators

1. Prefix: given an activity `(a,r)`, and a process `P`, `(a,r).P` is a process which performs the activity `(a,r)` and then becomes `P`

2. Choice: `P + Q` is a process which expresses competition between `P` and `Q`

3. Cooperation: given processes `P` and `Q`, and a set of activity names `L`, the process `P <L> Q` expresses the parallel composition of `P` and `Q` with synchronization on `L` activities; c.f. increasing the number of tokens in a SAN place

4. Hiding: given a process `P`, and a set of activity names `L`, the process `P/L` hides those names in `L` from further interaction

Incorporating PEPA into Möbius

Due to the Möbius AFI, incorporating PEPA as an atomic model formalism requires identifying the following:

- A useful notion of state variable
- A notion of action for changing state

We would like PEPA to be useful with respect to current composed model formalisms, so state variables should

- Be meaningful to a model with which the PEPA model is composed (partner model), and
- When changed, effect an understandable change in the state of the PEPA model

State Based Model Composition:

- Actions seem straightforward… (PEPA activities)
- … but how can a partner model cause a meaningful change in the state of a PEPA model?

Parameterized PEPA (PEPA*k*)

- "Extend" PEPA to explicitly include process parameters in the syntax
- Employs a well-understood theory that dates back to the 1980s (e.g. [Milner 89]). The theory is **not** new, but the application **is**!

For example – a *M/M/s/n* queue:

```
Queue[m,s,n] := [m < n] => (in,lambda).Queue[m+1,s,n]
              + [m > 0] => (out,mu*min(s,m)).Queue[m-1,s,n]
```

- For modeling convenience, we also include guards (and variable communication via cooperating activities, not shown here)
  - But have we invented a *new* stochastic process algebra?

PEPA Semantics for PEPA*k*

- Not really a new process algebra – we can translate a PEPA*k* process definition into a set of *indexed* PEPA process definitions
  - Process parameters become instantiated as indices to the defining variable
  - Evaluate guards; eliminate subsequent behavior from definition if guard is *false*
  - Evaluate expressions used in activity rates
  - Use Milner's construction to turn output activities into sums over activities with indexed types (rates need no modification!)
- The behavior of a PEPA*k* model in Möbius is the same as the behavior of the PEPA model to which its semantics translate it.

The Mapping

- The state variables available for sharing with a partner model are (approximately) the process parameters used in the definitions of the PEPA*k* processes
  - When the process algebra model consists of the parallel cooperation of several sequential components, we provide the union of the process parameters (with some scoping technicalities employed)
- The overall state of the process algebra model is the state of the process parameters, ***and*** the symbolic state of each sequential component
- The Möbius actions are taken to be the individual unsynchronized activities of each PEPA*k* component, along with ***every*** possible combination of synchronized activities

Implications for the Process Calculus

- The ability of a partner model to unilaterally change some shared state has consequences for the process algebra – consider the following example:

```
S[x] := [x!=1] => (a1,r).(b,s).S[1]
      + [x!=2] => (a2,r).(b,s).S[2]
      + [x!=3] => (a3,r).(b,s).S[3]
```

```
S[x] := [x!=1] => (a1,r).S'[1]
      +  [x!=2] => (a2,r).S'[2]
      +  [x!=3] => (a3,r).S'[3]
       S'[x] := (b,s).S[x]
```

- Furthermore, equivalence relations may not be employed for aggregation of parameterized definitions – although they be used for non-parameterized components (e.g., (P[2] || P[3]) may not be reduced).

What does Möbius use and generate?

- For example, every Möbius action must implement a method to determine whether it enabled to later "fire". For a PEPA model, the method below is always used, and provided in the PEPA base classes:

```
bool PEPAActivity::Enabled(){
  OldEnabled=NewEnabled;
  NewEnabled=(EnabledByCurrentTerms() && GuardSatisfied());
  return NewEnabled;
}
```

- Möbius generates C++ code for each specific PEPA model – for example, to determine if a guard is satisfied in the current state, or to determine the effect of firing an activity:

```
#include "PAPM-presentation/Atomic/Example/ExamplePEPA.h"
...
inline bool ExamplePEPA::out_Act::GuardSatisfied() {
  return (! (m->getValue() <= 0));
}

BaseActionClass *ExamplePEPA::out_Act::Fire() {
  UpdateCurrentTerms();
  m->setValue((m->getValue() - 1));
  s->setValue(s->getValue());
  n->setValue(n->getValue());
  return this;
}
```

Reward Variables in Mobius

Reward variables are a way of measuring performance- or dependability-related characteristics about a model.

Examples:

- Expected time until service
- System availability
- Number of misrouted packets in an interval of time
- Processor utilization
- Length of downtime
- Operational cost
- Module or system reliability

Reward Structures

Reward may be “accumulated” two different ways:

- A model may be in a certain state or states for some period of time, for example, “CPU idle” states. This is called a *rate reward*.
- An activity may complete. This is called an *impulse reward*.

The reward variable is the sum of the rate reward and the impulse reward structures.

Reward Structure Example

A web server failure model is used to predict profits. When the web server is fully operational, profits accumulate at $\$N$/hour. In a degraded mode, profits accumulate at $\frac{1}{6}N$/hour. Repairs cost $\$K$.

$$R(m) = \begin{cases} N & m \text{ is a fully functioning marking} \\ \frac{1}{6}N & m \text{ is a degraded-mode marking} \\ 0 & \text{otherwise} \end{cases}$$

$$C(a) = \begin{cases} -K & a \text{ is an activity representing repair} \\ 0 & \text{otherwise} \end{cases}$$

By carefully integrating the reward structure from 0 to t, we get the profit at time t. This is an example of an "interval-of-time" variable.

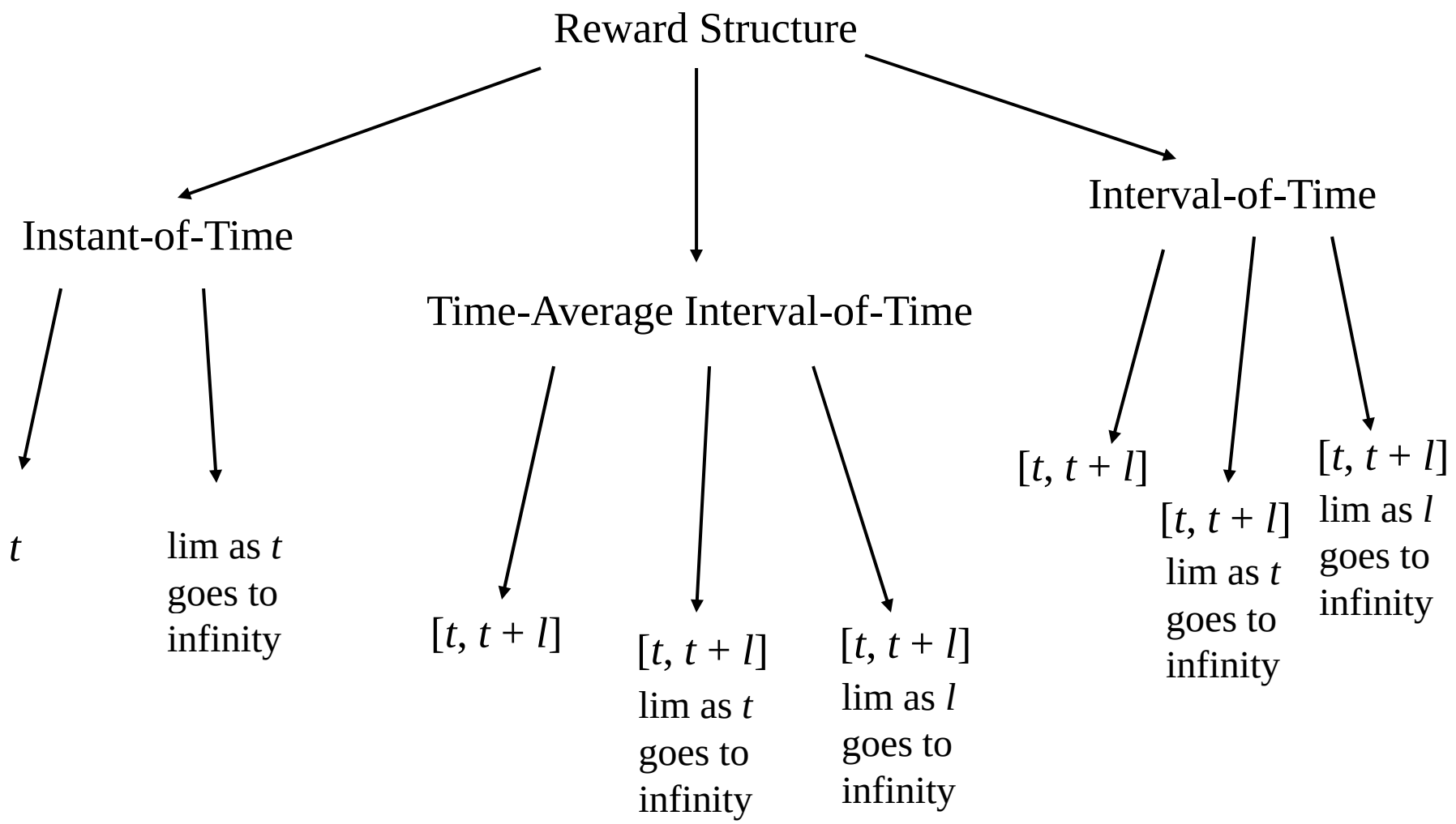
Reward Variables

A *reward variable* is the sum of the impulse and rate reward structures over a certain time.

Let $[t, t + l]$ be the interval of time defined for a reward variable:

- If l is 0, then the reward variable is called an *instant-of-time* reward variable.
- If $l > 0$, then the reward variable is called an *interval-of-time* reward variable.
- If $l > 0$, then dividing an interval-of-time reward variable by l gives a *time-averaged interval-of-time* reward variable.

Reward Variable Specification



Specifying Reward Variables in *Möbius*

- When specifying a rate portion of a reward structure in *Möbius*, you must define a predicate and function.
  - predicate: while true (i.e., integer greater than 0 in C), accumulate the reward
  - function: the value (i.e., double in C) to accumulate
- Note that both the predicate and function may be any C statement or expression.

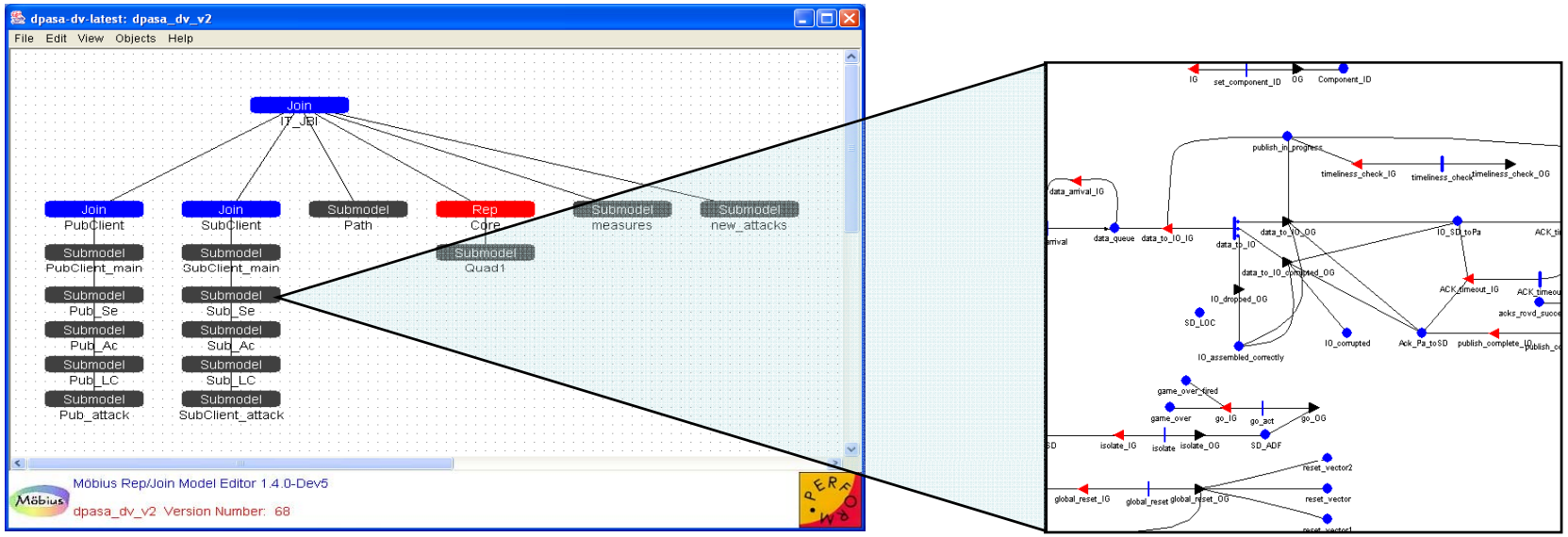
Queue Length	
	Rate rewards *subnet = database* Predicate: *1* Function: *MARK(Queue)*
	Impulse reward *none*

Model Composition Basics

- Model composition formalisms permit the construction of models from other models by sharing state variables or actions between models
- New model implements AFI, just like an atomic model
- State variable sharing can be of two types:
  - Equivalence sharing, where a state variable or "part" of state variable from one model is identified with a state variable or "piece" of state variable in another model (information flow is bi-directional)
  - Functional sharing, where the state of a state variable in one model is defined to be a function of another submodel's state (information flow is one-direction)
- Two complete or partial state variables can be equivalently shared if:
  - Their structure is the same (as defined by the state variable specification syntax presented earlier - in short, they are of the same type).
  - They have the same initial value

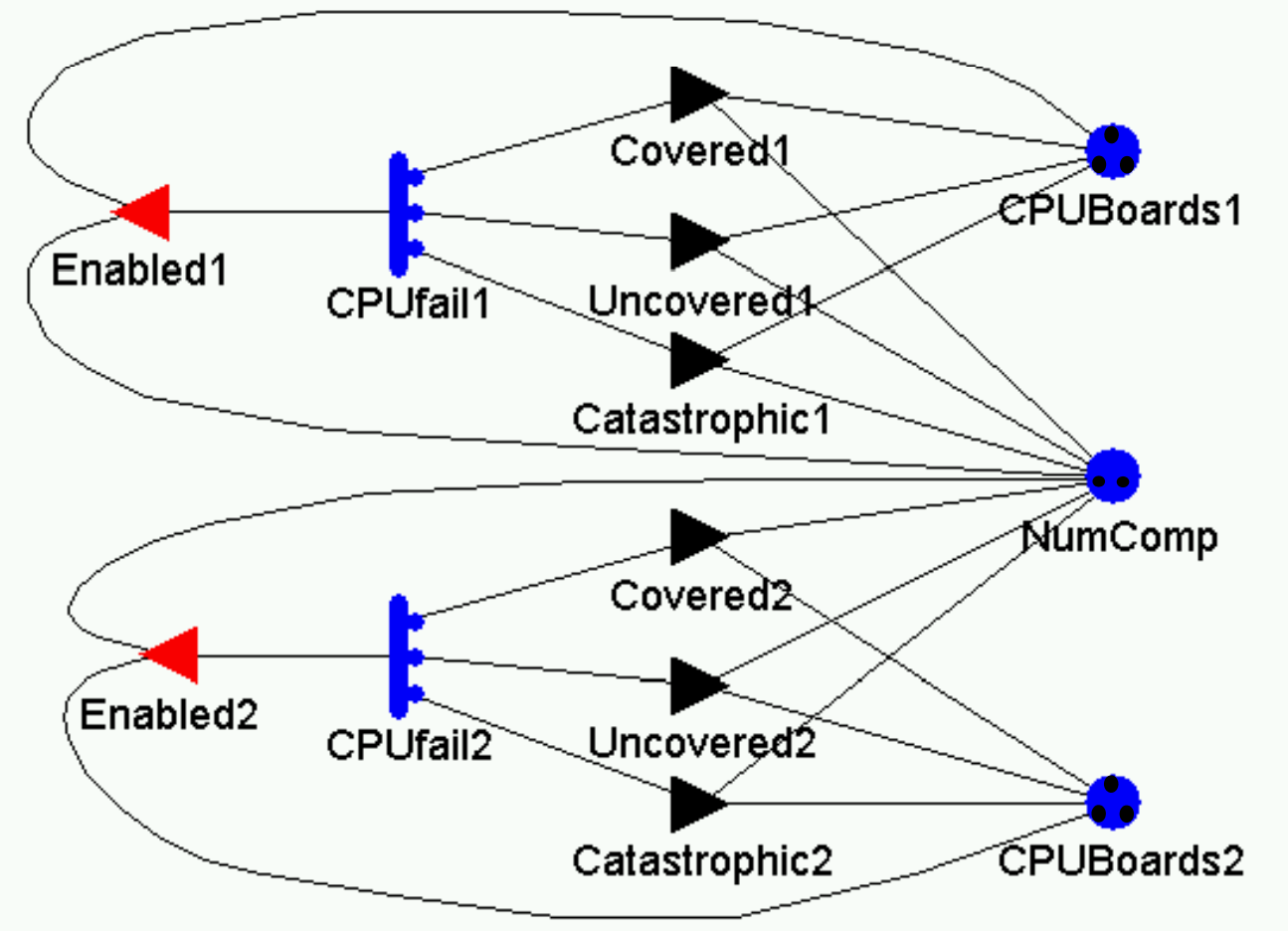
Model Composition in Mobius

- Hierarchical model construction
  - System model constructed by assembling multiple component models.
  - Can combine models built with different formalisms
- Rapid model development
  - Simple initial component models can be swapped out later for more complex ones.
- Multiple composition techniques provide flexibility in model construction
  - Rep / Join Graph Composition (state sharing)
  - General Grap Composition (State sharing)
  - General Graph composition (action syncronization)



Basic Model State-Space Generation in Mobius

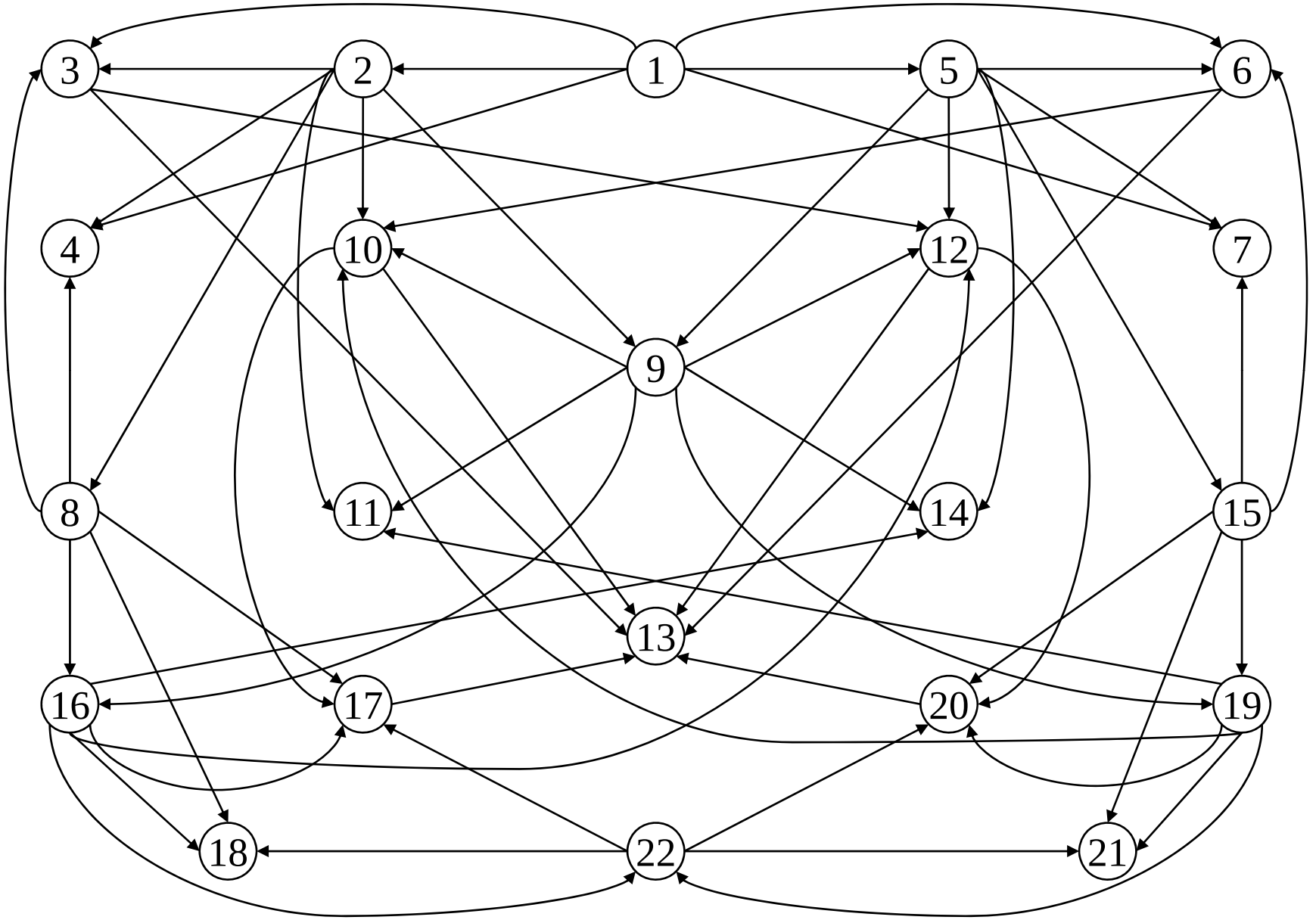
If the activity delays are exponential, it is straightforward to convert a SAN to a CTMC. We first look at the simple case, where there is no composed model.



State Space (Generated by *Möbius*)

State No.	CPUboards1	CPUboards2	NumComp	(Next State, Rate)
1	3	3	2	(2,.p1λ),(3,p2λ),(4,P3λ),(5,p1λ),(6,p2λ,),(7,p3λ)
2	2	3	2	(8,p1λ),(3,p2λ),(4,p3λ),(9,p1λ),(10,p2λ),(11,p3λ)
3	0	3	1	(12,p1λ),(13,(p2+p3) λ)
4	0	3	0	
5	3	2	2	(9,p1λ),(12,p2λ),(14,p3λ),(15,p1λ),(6,p2λ),(7,p3λ)
6	3	0	1	(10,p1λ),(13,(p2+p3) λ)
7	3	0	0	
8	1	3	2	(3,(p1+p2) λ),(4,p3λ),(16,p1λ),(17,p2λ),(18,p3λ)
9	2	2	2	(16,p1λ),(12,p2λ),(14,p3λ),(19,p1λ),(10,p2λ),(11,p3λ)
10	2	0	1	(17,p1λ),(13,(p2+p3) λ)
11	2	0	0	
12	0	2	1	(20,p1λ),(13,(p2+p3) λ)
13	0	0	0	
14	0	2	0	
15	3	1	2	(19,p1λ),(20,p2λ),(21,p3λ),(6,(p1+p2) λ),(7,p3λ)
16	1	2	2	(12,(p1+p2) λ),(14,p3λ),(22,p1λ),(17,p2λ),(18,p3λ)
17	1	0	1	(13, λ)
18	1	0	0	
19	2	1	2	(22,p1λ),(20,p2λ),(21,p3λ),(10,(p1+p2λ),(11,p3λ)
20	0	1	1	(13, λ)
21	0	1	0	
22	1	1	2	(20,(p1+p2) λ),(21,p3λ),(17,(p1+p2) λ),(18,p3λ)

Underlying Markov Model (State Transition Rates Not Shown)

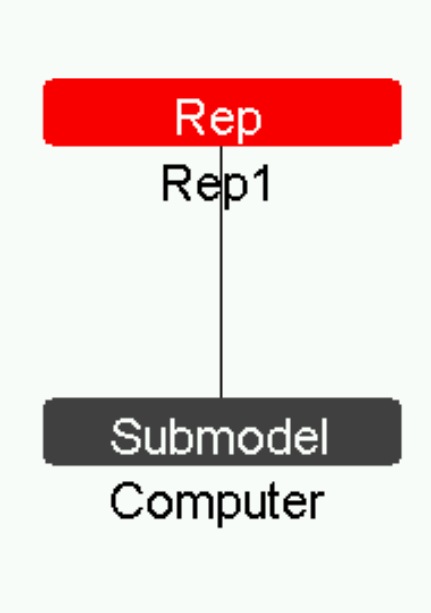


Reduced Base Model Construction in Mobius

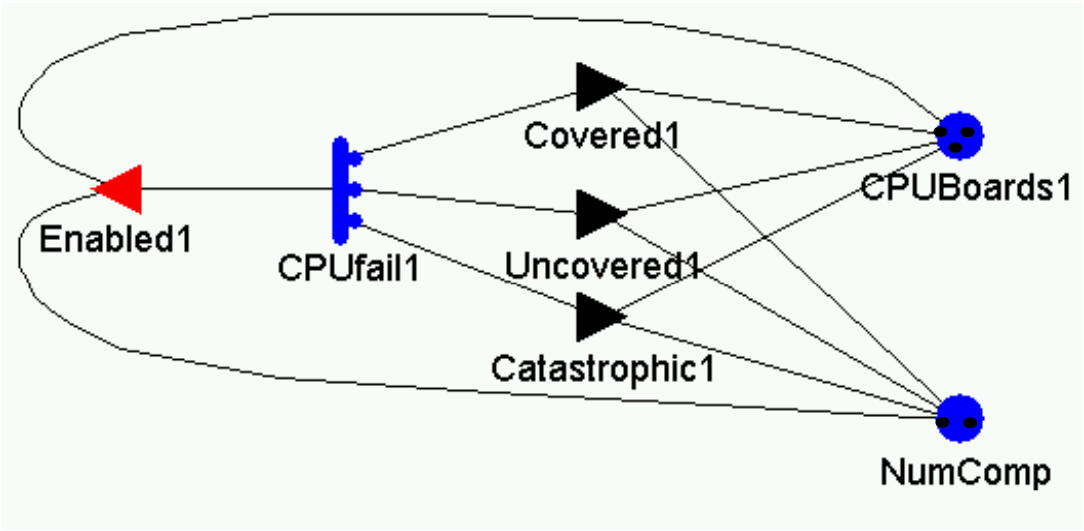
- "Reduced Base Model" construction techniques make use of composed model structure to reduce the number of states generated.
- A state in the reduced base model is composed of a state tree and an impulse reward.
- During reduced base model construction, the use of state trees permits an algorithm to automatically determine valid lumpings based on symmetries in the composed model.
- The reduced base model is constructed by finding all possible (state tree, impulse reward) combinations and computing the transition rates between states.
- Generation of the detailed base model is avoided.

Example Reduced Base Model State Generation

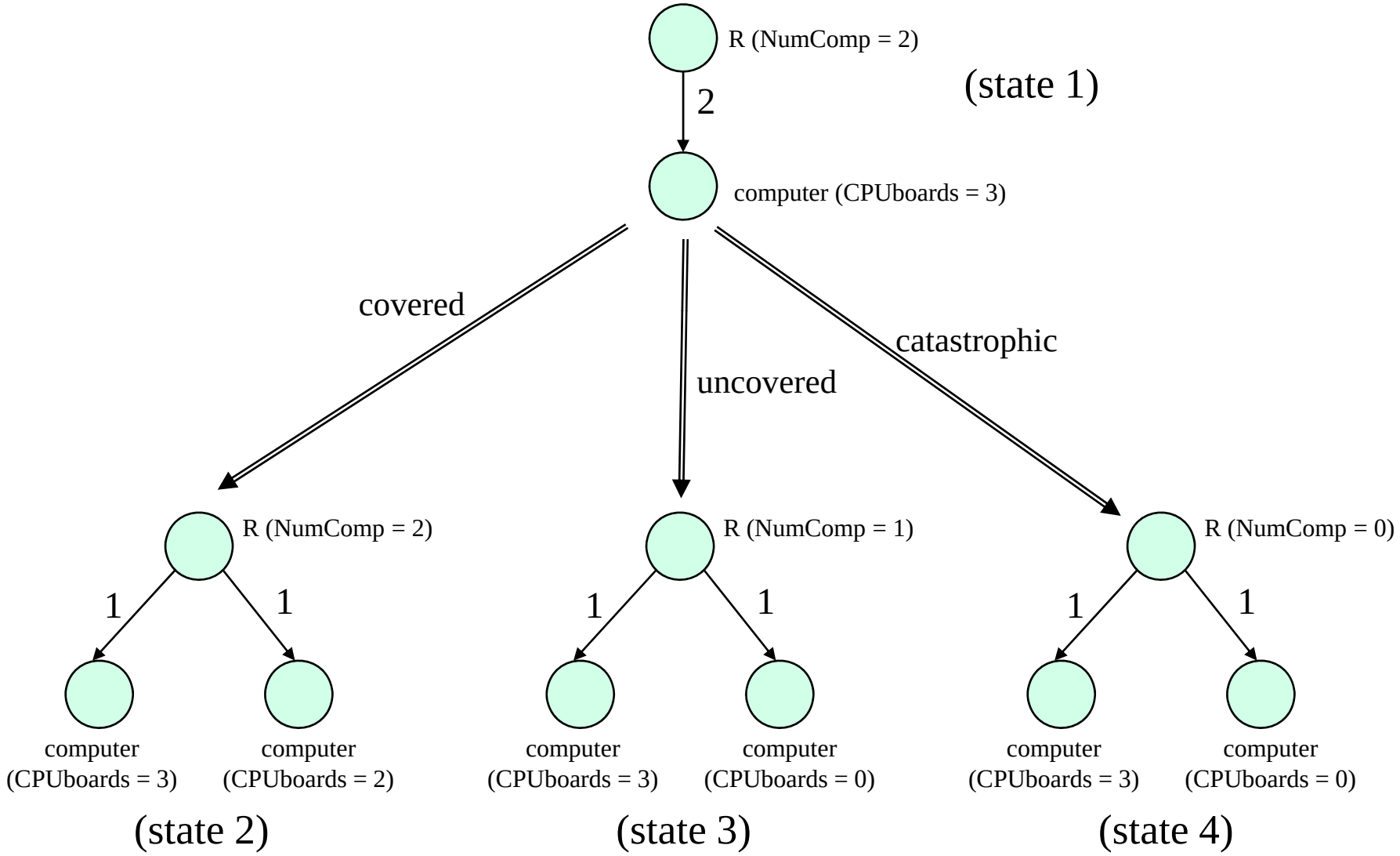
Composed Model



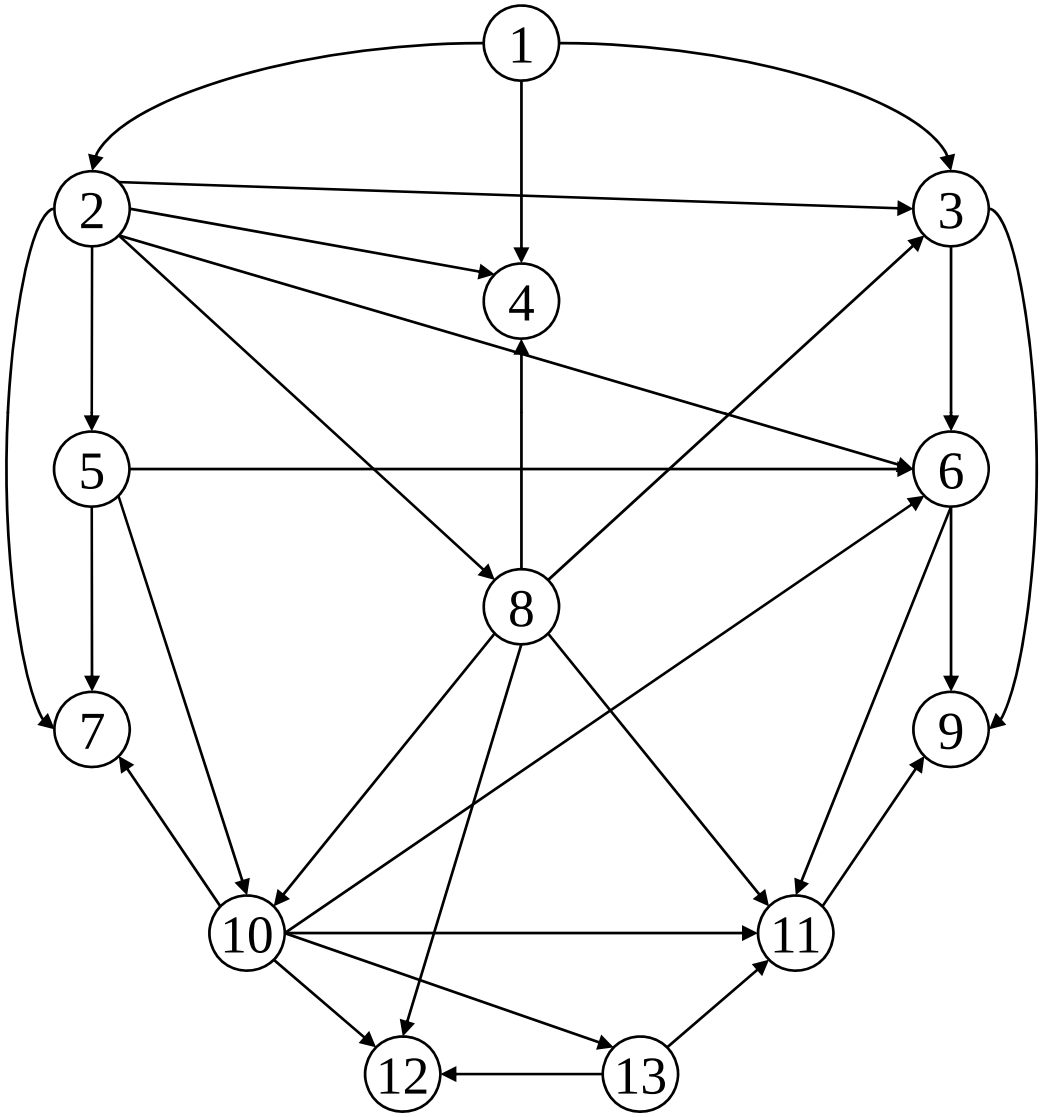
computer



Example Reduced Base Model States and Transitions

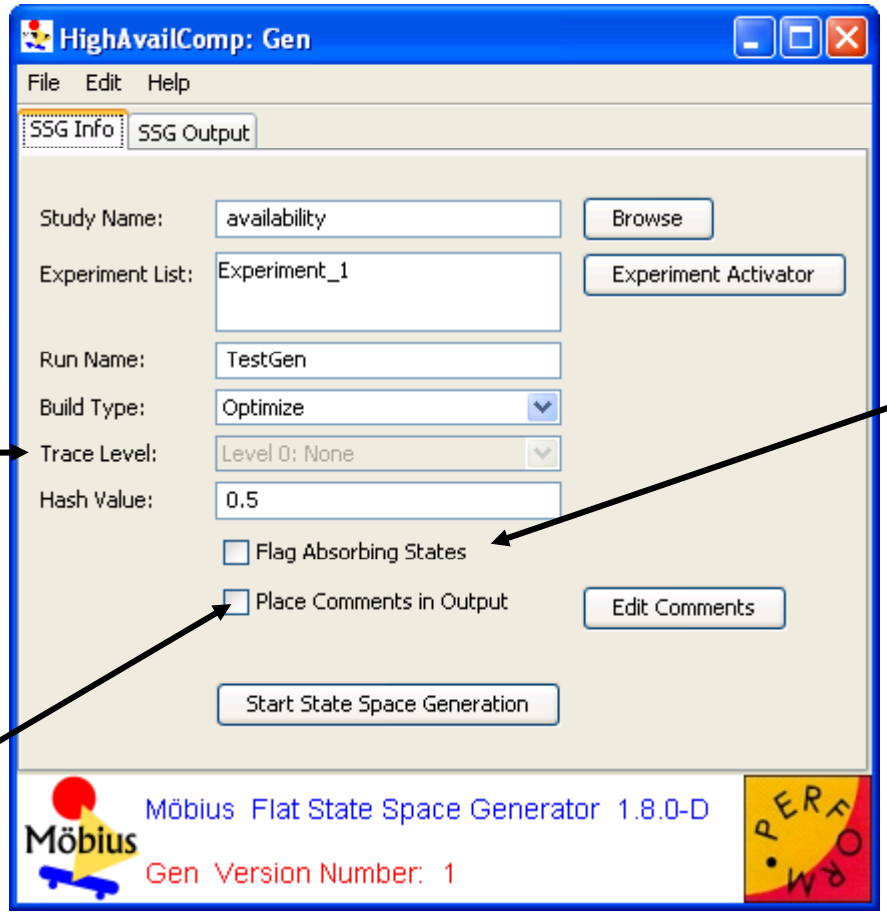


Markov Chain of Reduced Base Model (State Transition Rates not Shown)



State-Space Generation in Möbius

(For generating random process representations of models with all exponential or exponential/deterministic timed activities)



Print out states and reward variables

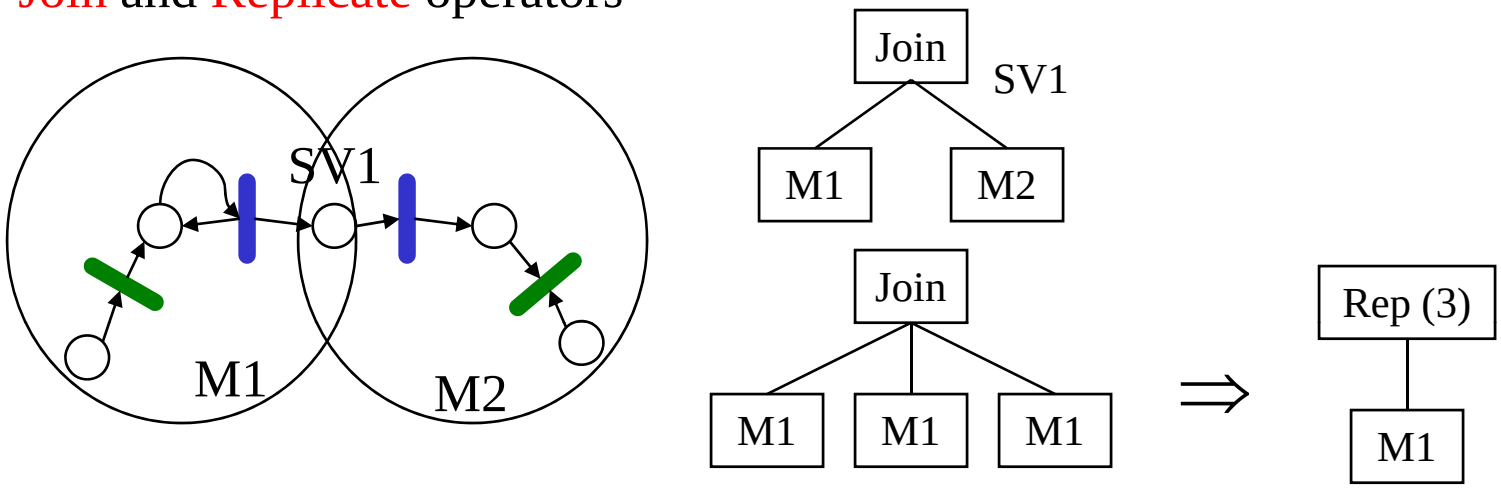
Print out absorbing states. Useful to detect problems when attempting a steady-state solution.

Place comments, as specified by edit comments, in file.

State-space generation must be done before all analytic/numerical solutions are done.

MDD Representation of State-sharing Composed Models (NSMC '03)

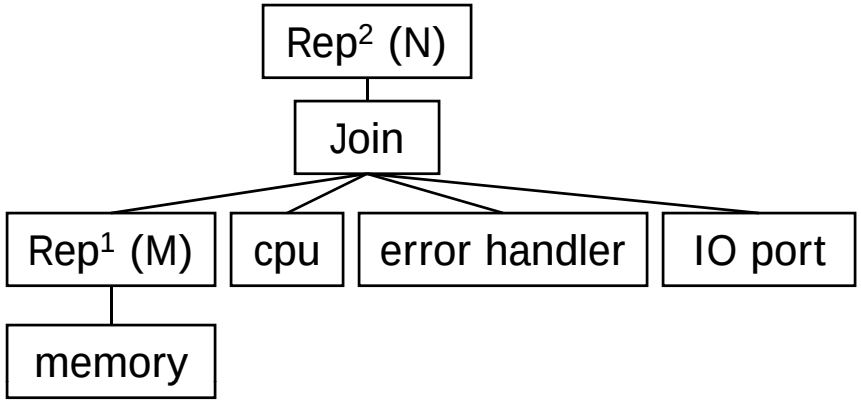
- Join and Replicate operators

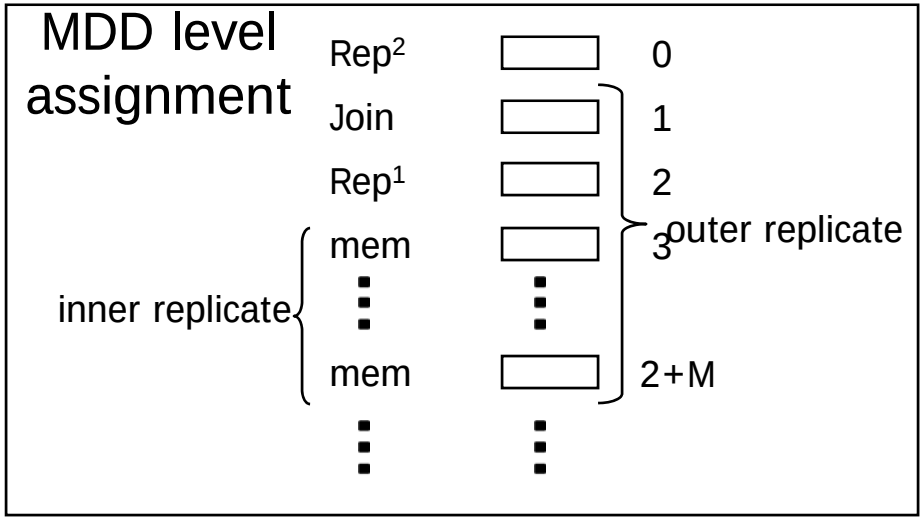


- Any atomic model formalism that can share state variables
  - E.g., SAN, PEPA$_k$, and Buckets and Balls
- Replicate induces symmetry
- Global and local actions

MDD Data Structure by Example

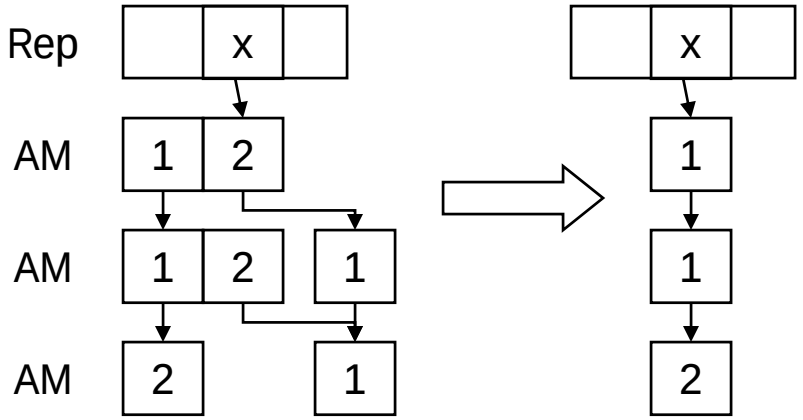
- Partitioning SVs based on composition structure
  - Maximizing efficiency of local SS exploration
  - Simplifying global SS exploration
- Dependability model of spacecraft flight control system





Composed-Model Induced Lumping of CTMC

- Redundant states (paths)
- Rep node c implies equivalence relation R_c



$$(v, v') \in R_c \Leftrightarrow$$
$$v_i = v'_i \text{ for all } i \in \{1, \ldots, c-1, c+n_c l_c+1, \ldots, m\}$$
$$\exists p : \{0, \ldots, n_c - 1\} \rightarrow \{0, \ldots, n_c - 1\} \text{ s.t.}$$
$$v(c, i) = v'(c, p(i)) \text{ for all } i = 0, \ldots, n_c - 1$$
$$\text{where } v(c, i) = (v_{c+il_c+1}, \ldots, v_{c+il_c+l_c})$$

- Overall equivalence relation $R = \cup_{c \text{ is replicate}} R_c$
- Canonical representative state in each class min(v)
- $\mathcal{R}$ may become exponentially large ⇒ break it up $\mathcal{S}_{\text{lumped}}$ into many extremely smaller MDDs ⇒ faster computation of

SSG and Lumping Performance

N	unlumped SS (MDD)			lumped SS (MDD)				generation time (sec)	lumping time (sec)
	# states	# nodes	mem (KB)	# states	final # nodes	mem (KB) final	mem (KB) peak		
1	4.14×10^{2}	14	1.46	1.16×10^{2}	18	2.06	15.0	0.027	$\sim$0
2	2.57×10^{5}	43	4.54	1.01×10^{4}	340	44.9	118	1.30	0.0078
3	1.24×10^{8}	99	10.3	4.63×10^{5}	1494	194	441	25.1	0.06
4	5.50×10^{10}	167	17.3	1.48×10^{7}	3395	424	1050	200	0.42
5	2.35×10^{13}	247	25.5	3.67×10^{8}	5724	705	2050	1310	1.45
6	9.9×10^{15}	339	34.8	7.53×10^{9}	8481	1040	3560	5250	2.38

- This is a worst case example: No local behavior
- Drastic decrease in number of states in the lumped SS (up to 6 orders of magnitude)
- Increase in number of nodes in the lumped state space but still small compared to other entities
- Very small unlumped and lumped SS representation

CTMC Enumeration Performance

N	# states	# transitions	sorting MDD			time/iteration (sec)		slowdown factor
			# nodes	mem (KB)	gen. time (sec)	matrix diagram	sparse matrix	
2	1.01×10^{4}	5.51×10^{4}	1.50×10^{3}	1.56×10^{2}	0.066	1.39×10^{-2}	2.83×10^{-3}	4.91
3	4.63×10^{5}	3.51×10^{6}	3.47×10^{4}	4.00×10^{3}	4.81	9.59×10^{-1}	1.75×10^{-1}	5.48
4	1.48×10^{7}	1.43×10^{8}	6.91×10^{5}	8.38×10^{4}	235	4.42×10^{1}	–	–

TWS	SS				SSG time (sec)	transient solution (sec/iteration)		slowdown
	# states	final # nodes	mem (KB) final	mem (KB) peak		MxD	APNN	
3	2.38×10^{6}	23	13.3	2.24×10^{2}	1.07	1.26	0.885	1.42
4	9.71×10^{6}	29	31.0	7.25×10^{2}	4.76	5.54	3.76	1.47
5	3.24×10^{7}	35	67.6	1.97×10^{3}	21.0	19.15	12.76	1.50
6	9.33×10^{7}	41	135	4.76×10^{3}	85.4	-	-	-
7	2.40×10^{8}	47	254	1.06×10^{4}	325.3	-	-	-
8	5.62×10^{8}	53	453	2.19×10^{4}	1020	-	-	-

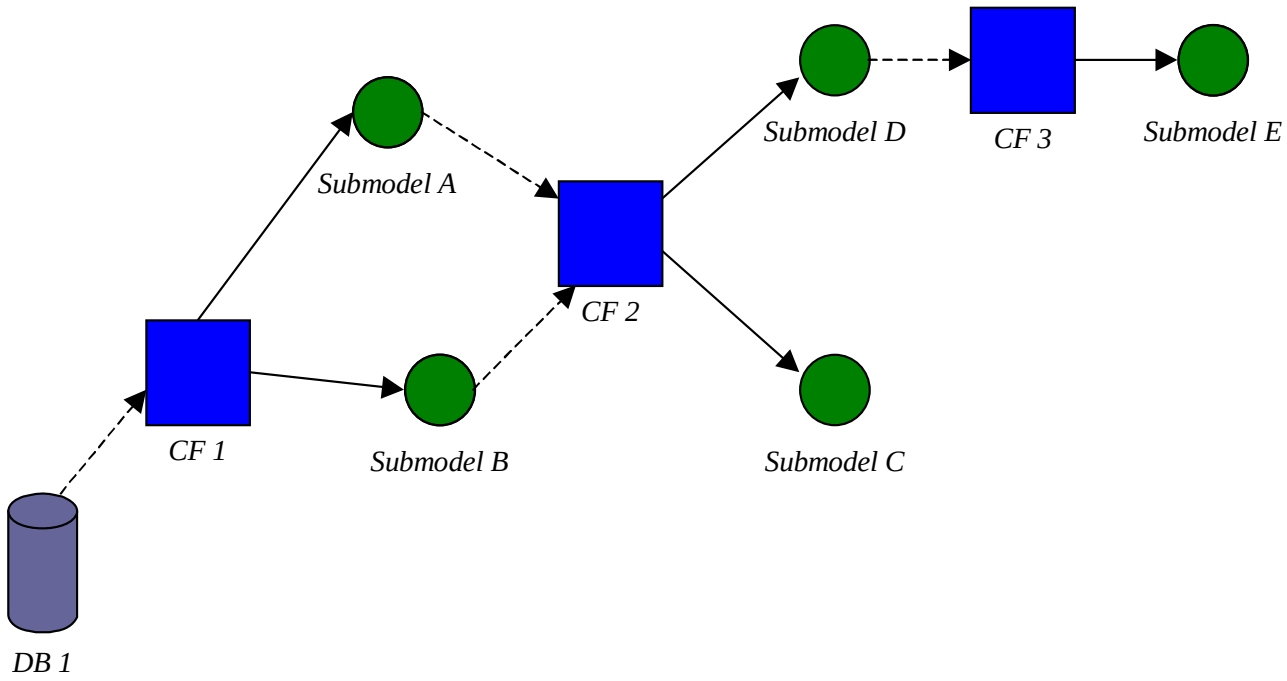
- Fairly fast iteration: Slowdown of 1 to 5 times, relative to direct sparse matrix access
- Solve dramatically larger larger CTMCs

Model Connection

- Model connection formalisms permit the construction of model solutions from a set of models by exchanging "results" between the models
- The abstract functional interface provides the infrastructure necessary to build connection formalisms
- "Results" can be:
  - The mean and/or variance of a the a performance variable
  - The density or distribution function of a performance variable (e.g. exponential-polynomial distribution)
  - Some automatically-constructed more-abstract model representation, e.g.,
    - Hidden Markov model
    - Markov-modulated Poisson processes

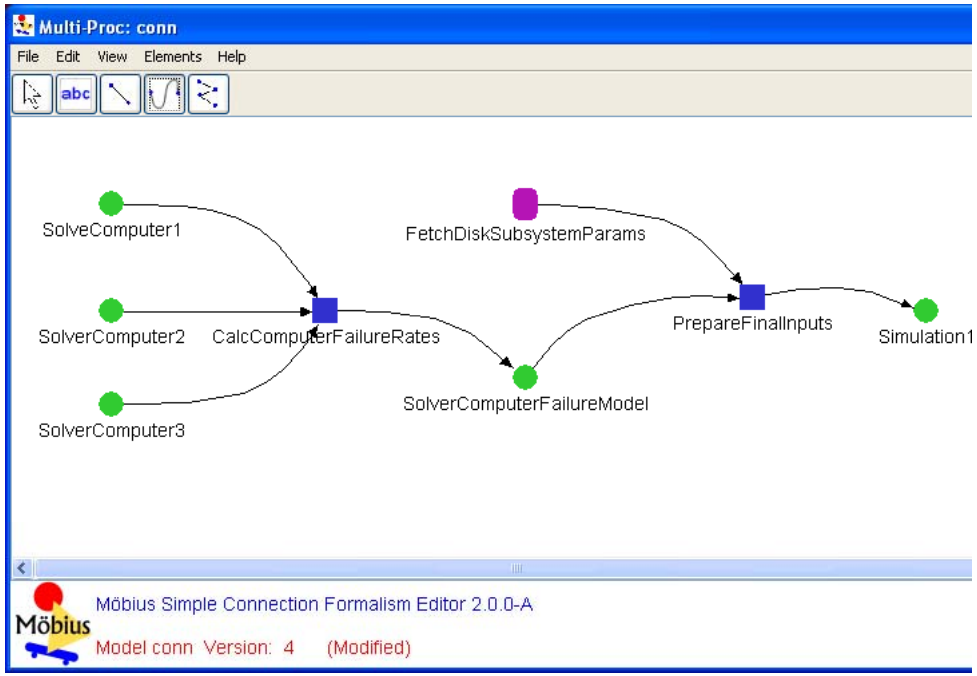
General Connection Formalism

- A connected model is a set of solvable models, and a diagram for passing results between those models
  - Heterogeneous submodels
  - General graph structure (cyclic or acyclic) submodels



Connection Solver

- The **connection solver** defines a multi-step solution process
  - Solution from one model become inputs to a subsequent model.
  - Perform fixed point iteration by defining a starting point and ending condition on a cyclical graph.
- Four connected model elements:
  - **Solver** nodes define a sol. step
  - **Connection function** nodes combine multiple results using mathematical and logical expressions.
  - **Database** nodes import previously computed solution results from the results database.
  - **Conduits** connect nodes in the graph and represent data passing between the nodes.



Outline

- Motivation: Dependability, Performance, and Performability Evaluation
- The need for multi-formalism, multi-solution evaluation frameworks
  - The Möbius modeling framework
- Model Specification Methods
  - Atomic Models (e.g. SANs and PEPA)
  - Reward Variable Specification
  - Model Composition (and state space generation)
  - Model Connection
- Model Solution Methods
  - Simulation
  - Analytic Methods
- Putting it all together

Types of Discrete-Event Simulation in *Mobius*

- Basic simulation loop specifies how the trajectory is generated, but does not specify how measures are collected, or how long the loop is executed.

- How measures are collected, and how long (and how many times) the loop is executed depends on type of measures to be estimated.

- Two types of discrete-event simulation are implemented in *Mobius*. The choice of which one to use depends on what type of measures are to be estimated.

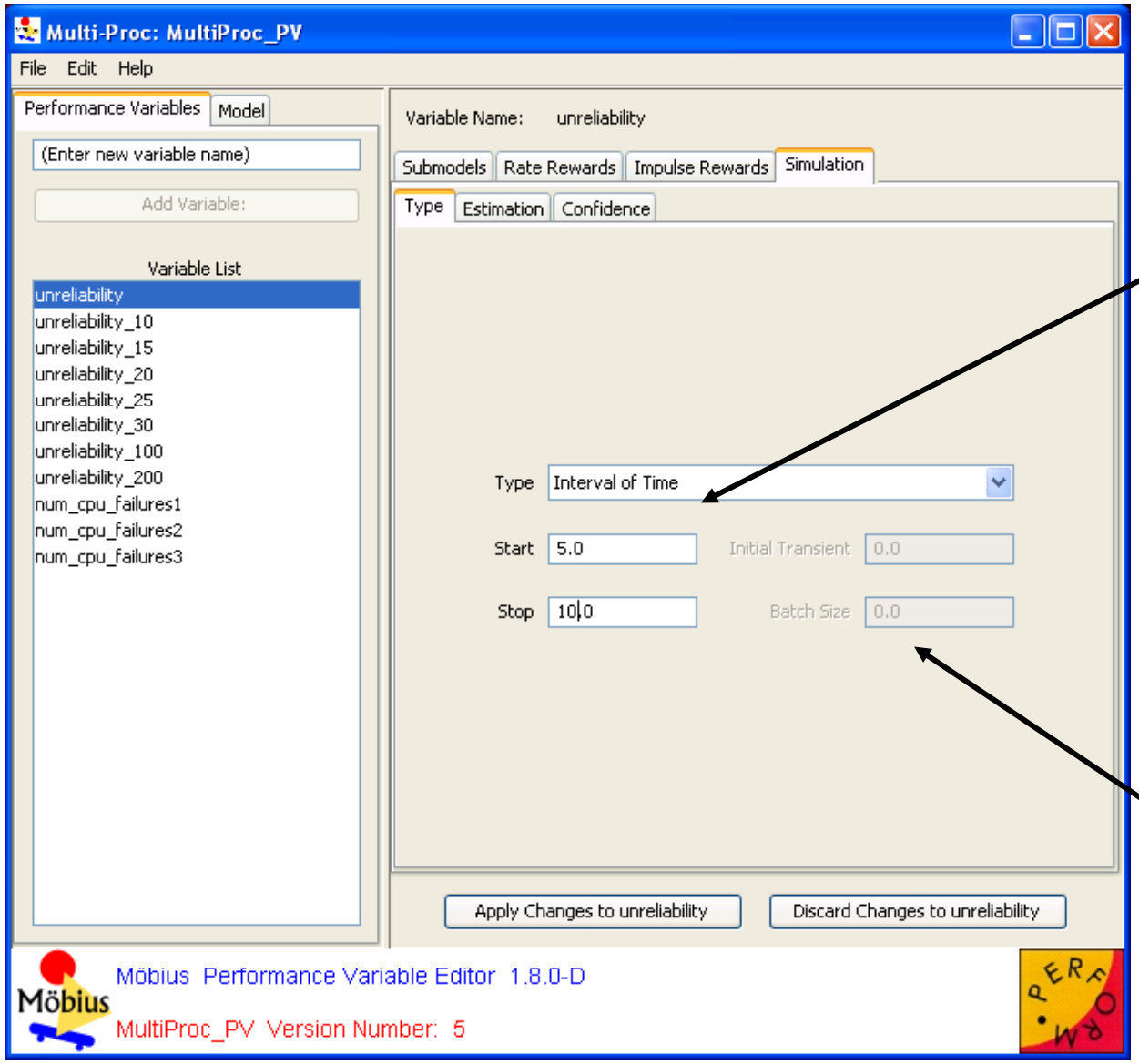
  - *Terminating* - Measures to be estimated are measured at fixed instants of time or intervals of time with fixed finite point and length.

  - *Steady-state* - Measures to be estimated depend on instants of time whose starting points are taken to be $t \rightarrow \infty$.

Support for Generally Distributed Action Timings in *Mobius*

- *Mobius* supports use of the following activity time distributions:
  - exponential
  - deterministic
  - geometric
  - weibull
  - normal
  - lognormal
  - erlang
  - cond weibull
  - gamma
  - beta
  - uniform
  - binomial
  - negative binomial
  - hyperexponential
- Parameters for distributions are as given in the manual, and are described in detail in [Law 91]
- Note that all distributions are truncated at zero (since time does not flow backwards) and hence a distribution's mean (or other characteristics) may not be as specified.

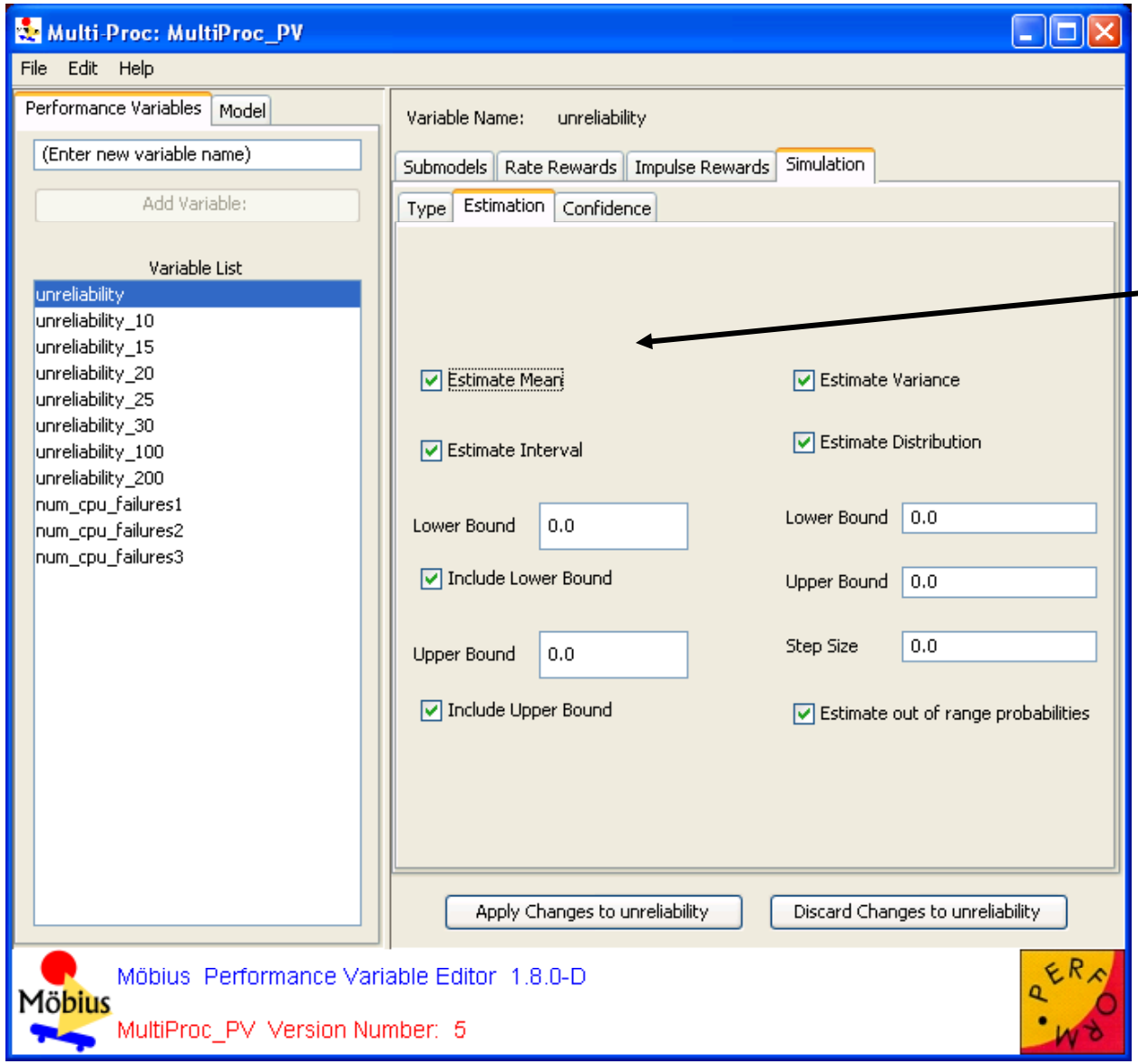
Simulator Statistics Editor



Variable Type and Times for Terminating Simulation

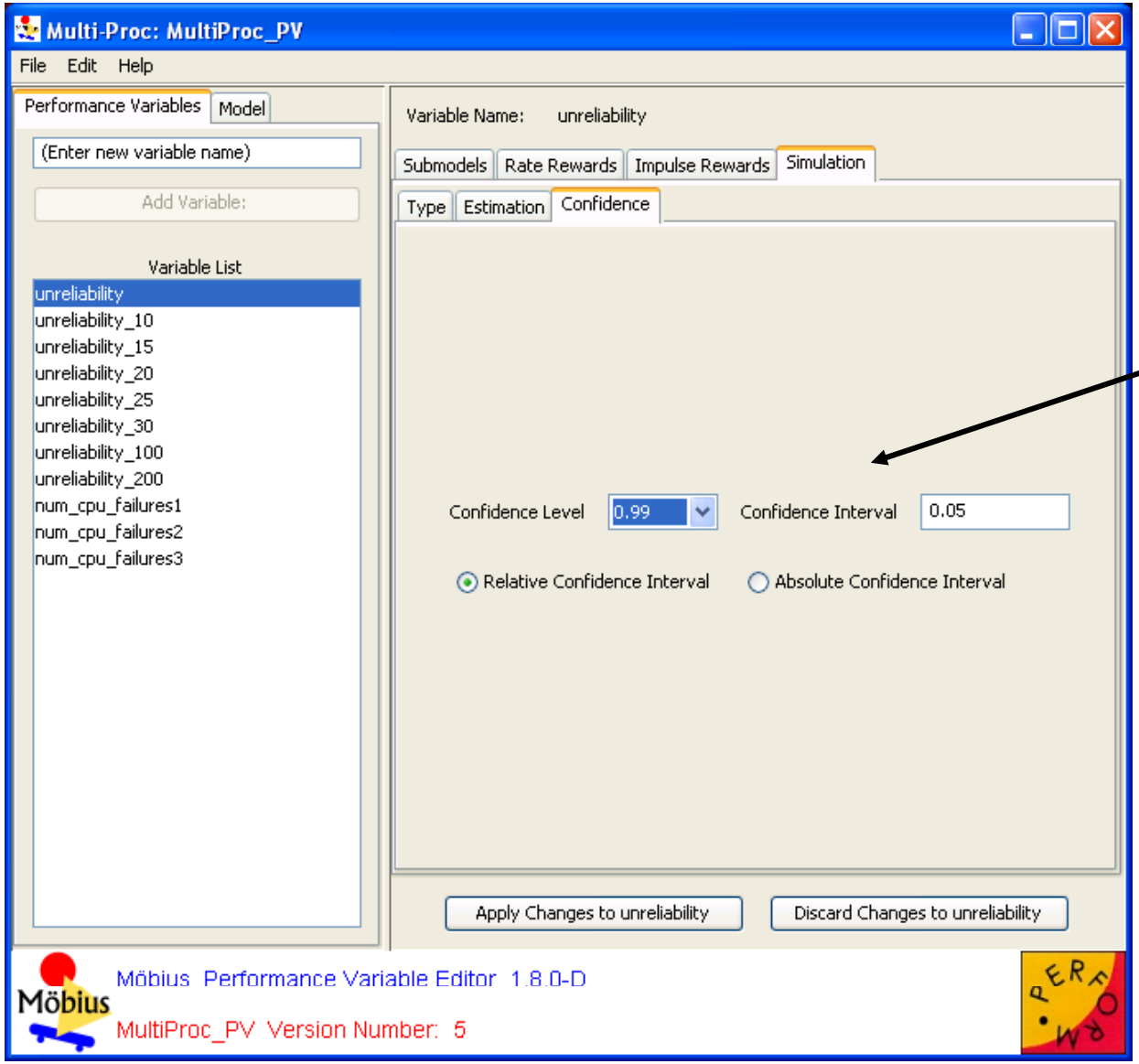
Batch Size and Initial Transient in Steady-State Simulation

Simulator Statistics Editor



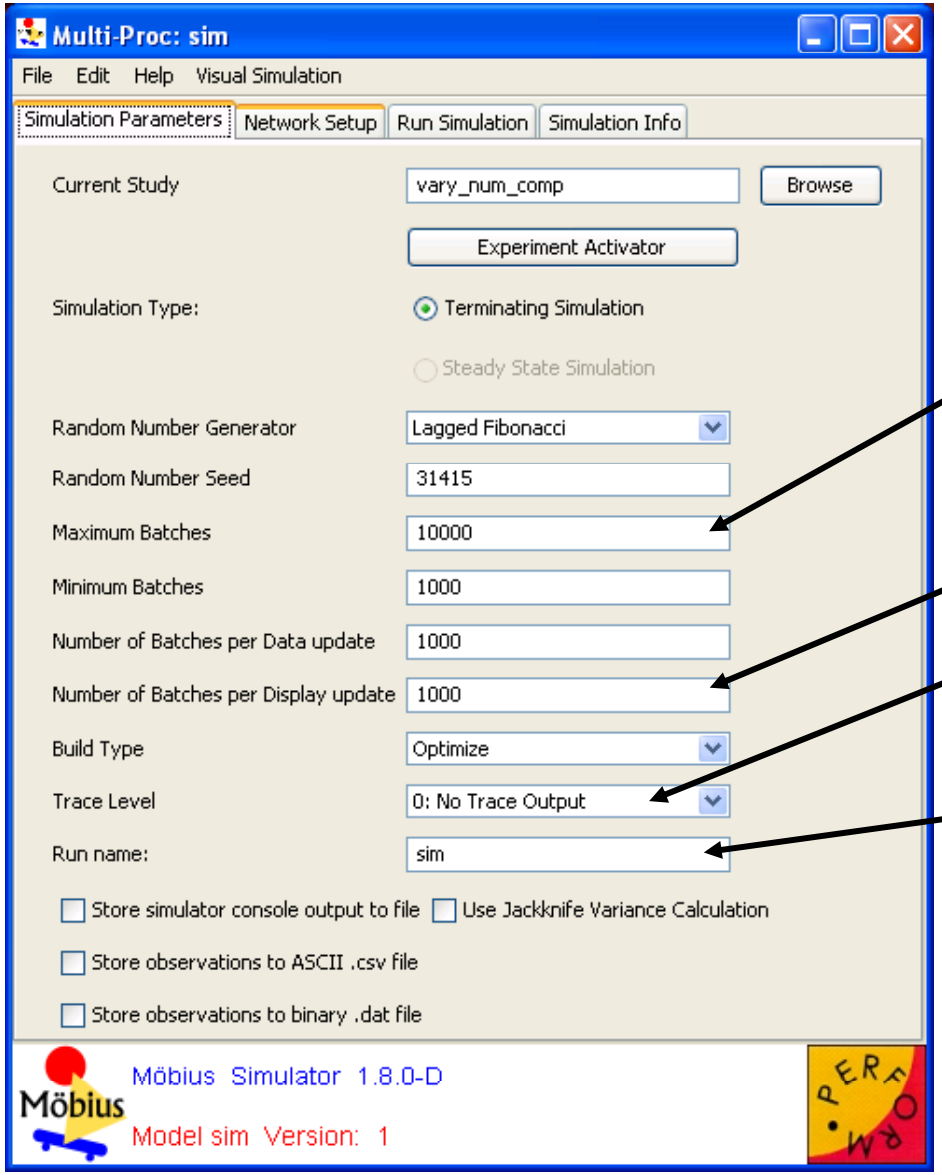
Estimator Types

Simulator Statistics Editor



Confidence Interval Width and Level

Simulator Editor



Maximum and Minimum Number of Replications to Run

Number of Batches between each calculation of the variance

Trace-Level for Debugging

File Name of Output File

Numerical/Analytical Solution Techniques

1) Transient Solution
   - Standard Uniformization (instant-of-time variables)
   - Adaptive Uniformization (instant-of-time variables)
   - Interval-of-time Uniformization (expected value, interval-of-time variables)

2) Steady-state Solution
   - Direct Solution (instant-of-time steady-state variables)
   - Iterative Solution (instant-of-time steady-state variables)

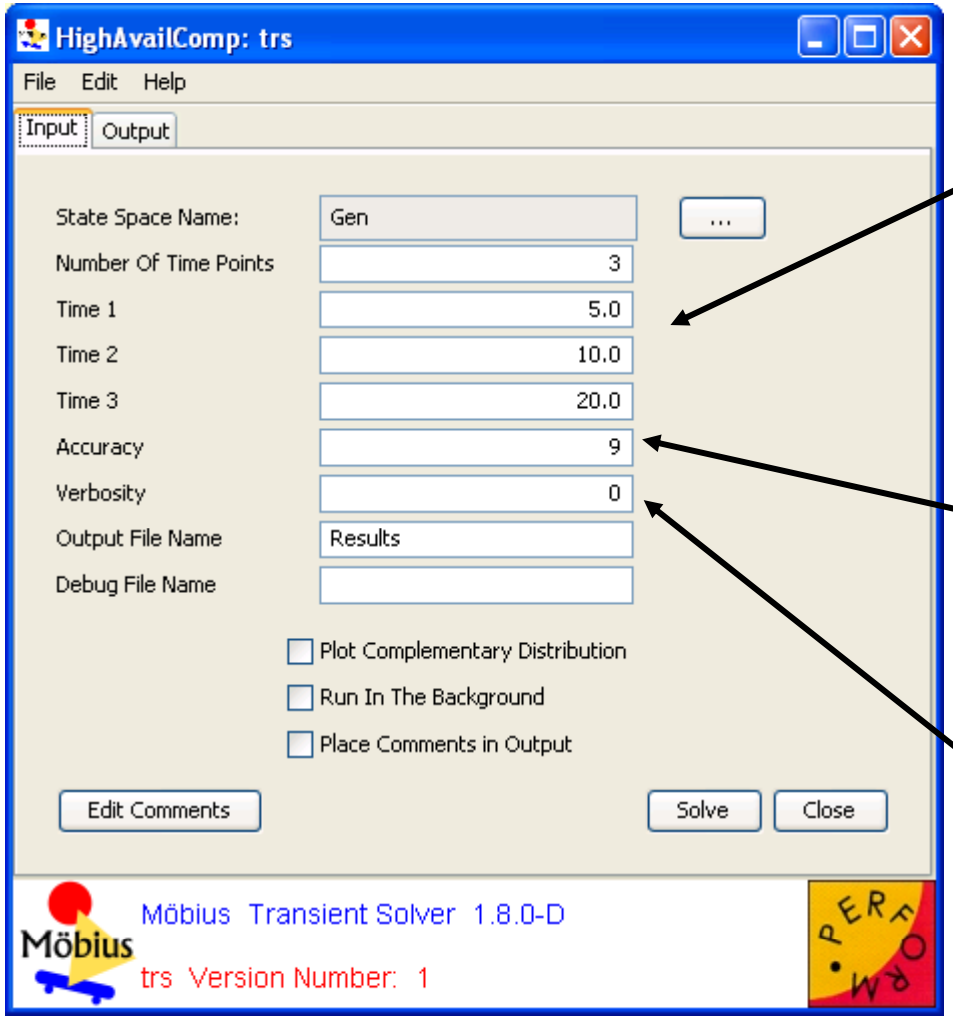
Möbius Analytical Solvers

Analytic Solvers (for reward variables only)				
Model Class	*Steady-state or Transient*	*Instant-of-time or Interval-of-time*	*Mean, Variance, or Distribution*	*Applicable Analytic Solver*
All activities exponential	Steady-state	Instant-of-time[a]	Mean, Variance, and Distribution	*dss* and *iss*
	Transient	Instant-of-time	Mean, Variance, and Distribution	*trs* and *atrs*
		Interval-of-time	Mean	*ars*
Exponential and Deterministic activities	Steady-state	Instant-of-time[b]	Mean, Variance, and Distribution	*diss* and *adiss*

[a] if only rate rewards are used, the time-averaged interval-of-time steady-state measure is identical to the instant-of-time steady-state measure (if both exist).

[b] provided the instant-of-time steady-state distribution is well-defined. Otherwise, the time-averaged interval-of-time steady-state variable is computed and only results for rate rewards should be derived.

Transient Uniformization Solver
(for transient solution of instant-of-time variables)

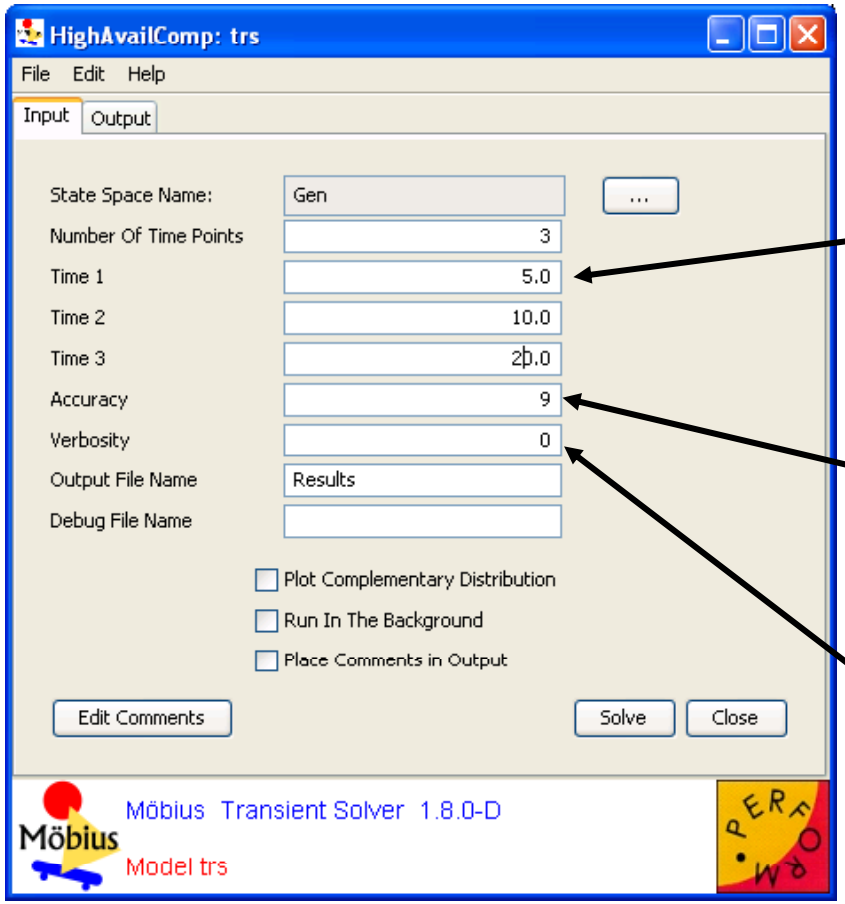


Instant-of-time variable time points of interest. Multiple time points may be specified, separated by spaces.

Number of digits of accuracy in the solution. Solution reported is a lower bound.

Volume of intermediate results reported. "1" gives the greatest volume, greater numbers less.

Adaptive Uniformization Solver (atrs)
(for transient solution of instant-of-time variables)



Instant-of-time variable time points of interest. Multiple time points may be specified.

Number of digits of accuracy in the solution. Solution reported is a lower bound.

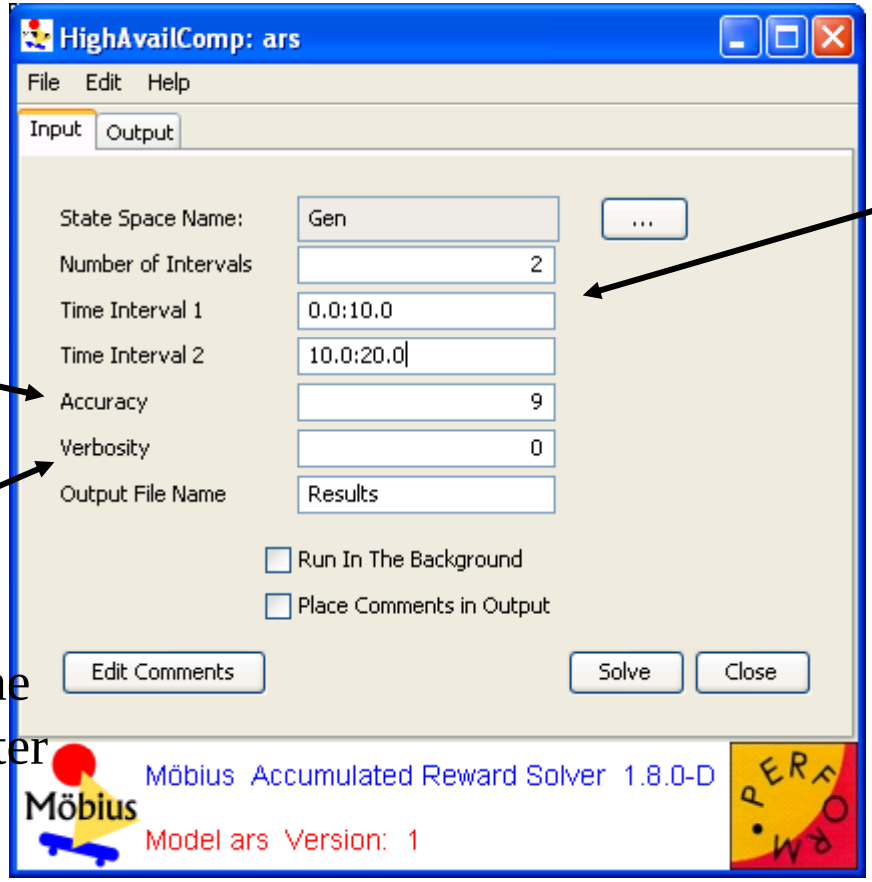
Volume of intermediate results reported. "1" gives the greatest volume, greater numbers less.

Accumulated Reward Solver (ars)

(solves for expected values of interval-of-time and time-averaged interval-of-time variables on intervals $[t_0, t_1]$ when both t_0 and t_1 are finite)

Number of digits of accuracy in the solution. Solution reported is a lower bound.

Volume of intermediate results reported. "1" gives the greatest volume, greater numbers less.

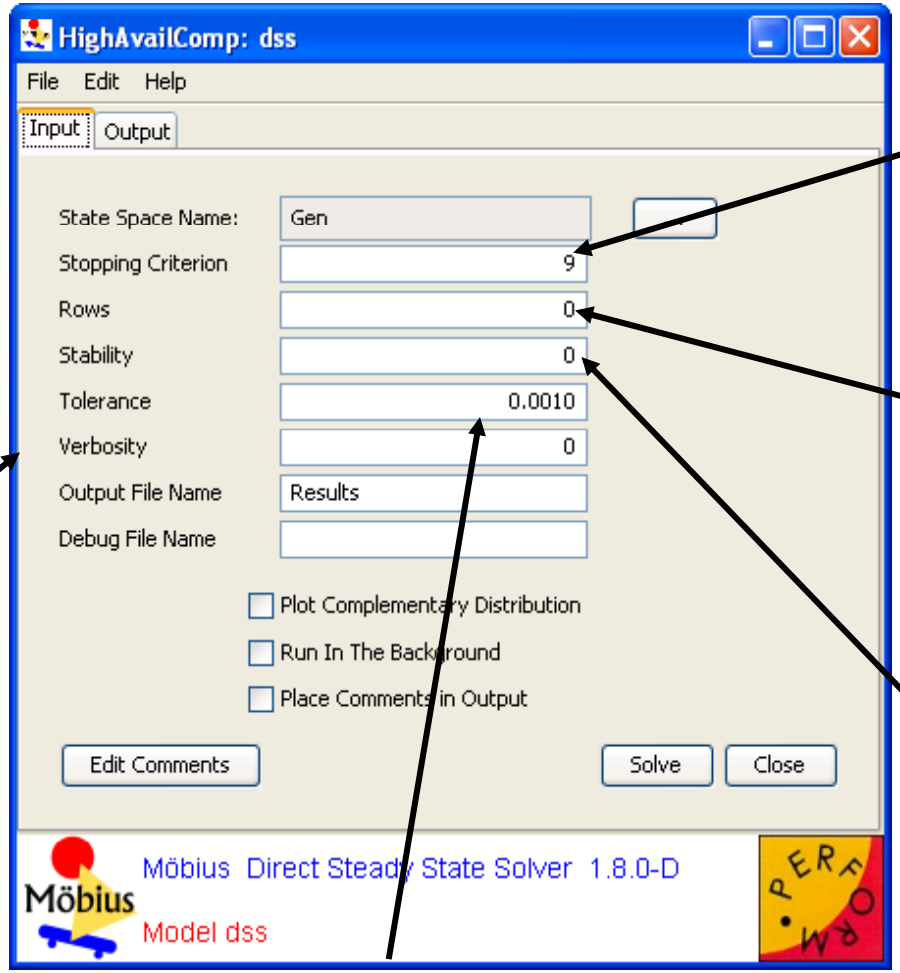


Series of time intervals for which solution is desired. Intervals are separated by spaces. Each interval can be specified as t_1:t_2.

The accumulated reward solver is based on uniformization, so the hints given for the transient solver apply here as well.

Direct Steady-State Solver (dss)

(for steady-state solution of instant-of-time variables)



Stopping criterion used in iterative refinement phase, after direct solution is done.

Number of rows to search for the "best" pivot when performing LU decomposition

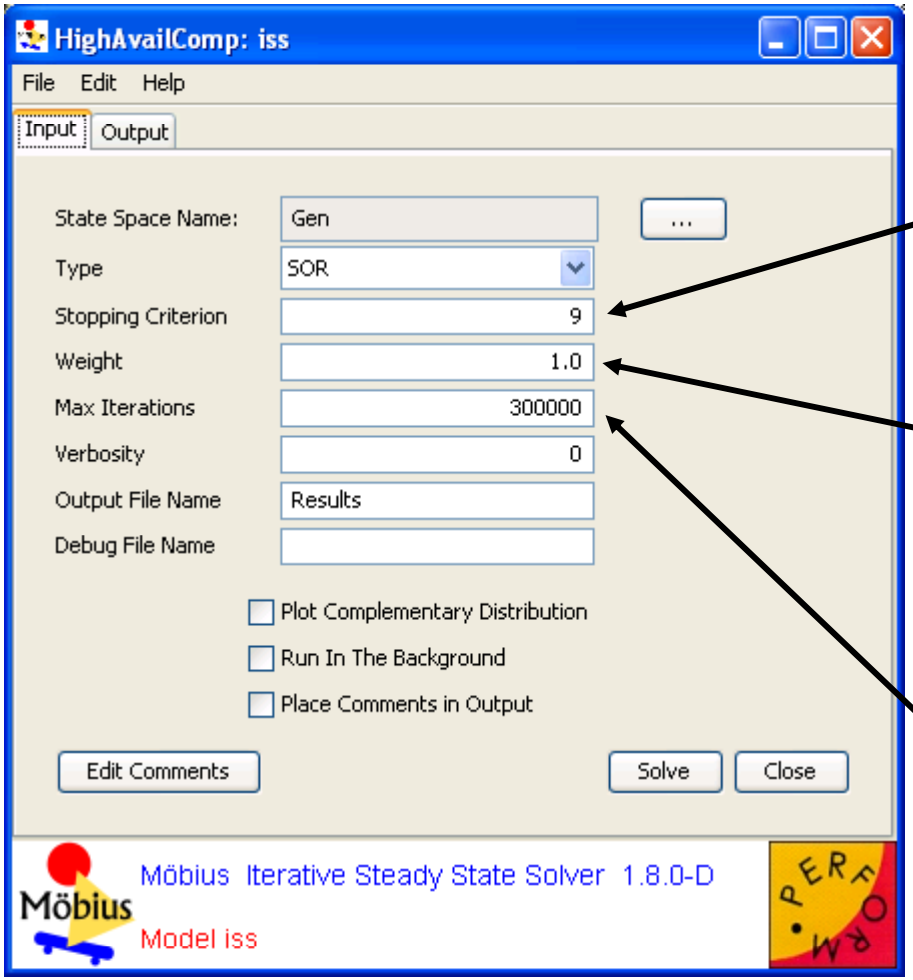
"Grace" factor by which elements may become pivots

Volume of intermediate results reported. "1" gives the greatest volume, greater numbers less.

Value that, when multiplied by smallest matrix element, is threshold at which elements may be dropped in LU decomposition.

Iterative Steady-State Solver (iss)

(for steady-state solution of instant-of-time variables)



Stopping criterion, expressed as 10^{-x}, where x is given. The criterion used is the infinity difference norm.

SOR weight factor. Values < 1 guarantee convergence, but slow it. Values >= 1 speed convergence, but may not converge.

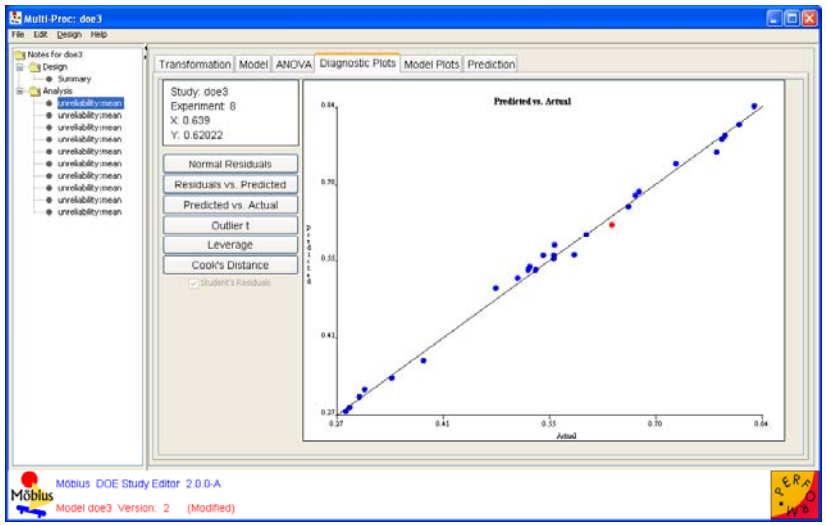
Maximum number of iterations allowed.

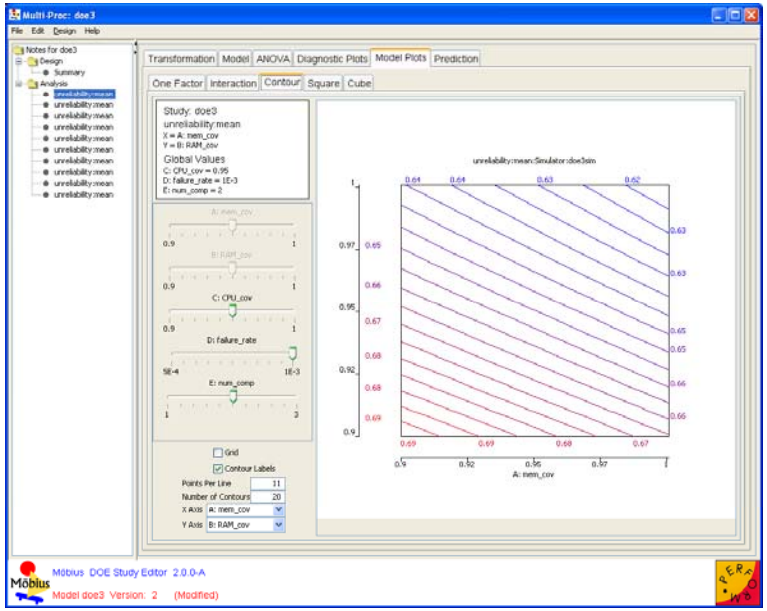
Outline

- Motivation: Dependability, Performance, and Performability Evaluation
- The need for multi-formalism, multi-solution evaluation frameworks
  - The Möbius modeling framework
- Model Specification Methods
  - Atomic Models (e.g. SANs and PEPA)
  - Reward Variable Specification
  - Model Composition (and state space generation)
  - Model Connection
- Model Solution Methods
  - Simulation
  - Analytic Methods
- Putting it all together

Design of Experiments

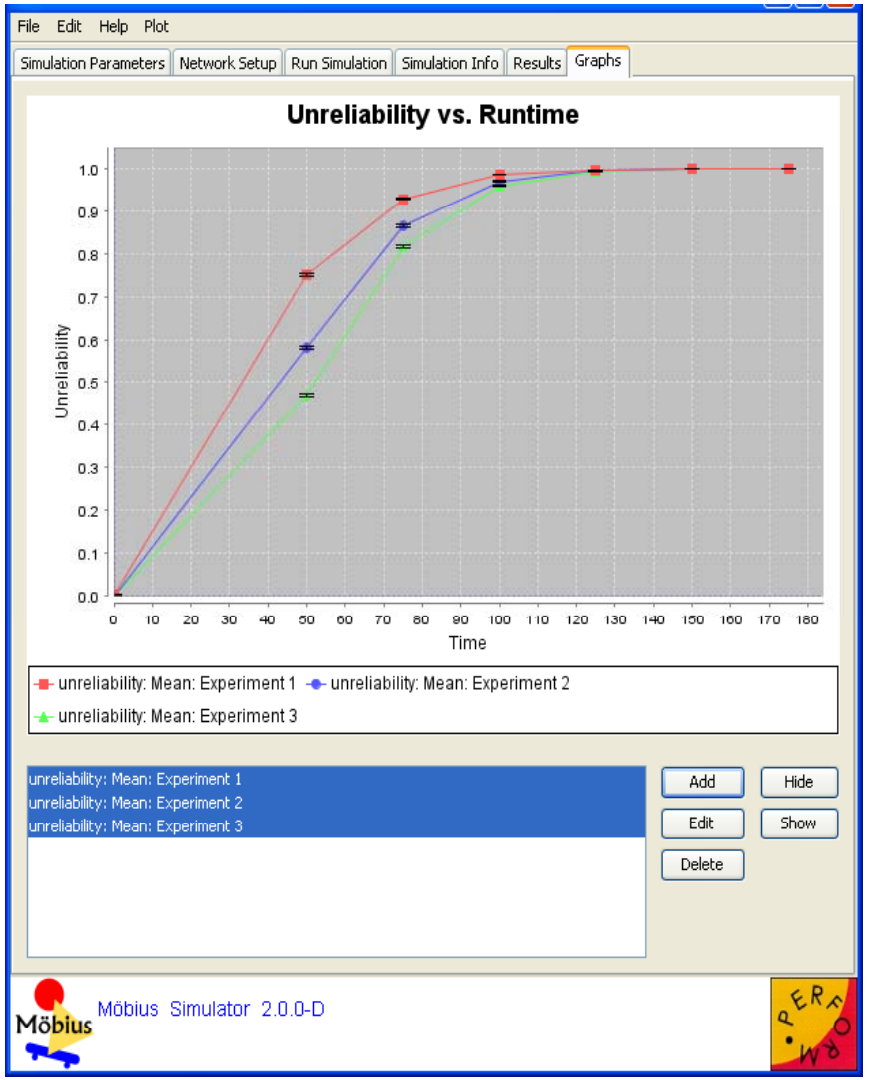
- Models of complex systems contain **many input parameters** that define the behavior of the system.
  - Desire to know which parameter values optimize specific output measures.
  - Exhaustive exploration of the parameter space of a large model is computationally expensive.
- **Design of Experiments** techniques:
  - Determine the degree of **sensitivity** each response (output measure) has for the various factors (input parameters) in the model.
  - Build a regression model of the response and generate a **response surface** to predict system behavior.
  - Guide the user toward the factors that optimize the desired response.
- Results imported from database.





Archiving and Visualization of Analysis Results

- **Integrated results database** stores results produced by solution techniques within Möbius.
- **Share** results between modules in Möbius
  and third-party SQL applications.
- **Multiple plot types** are supported:
  - reward value vs. experiment
  - reward value vs. time
  - probability distributions
- **Compare** results from different model configurations and input values
- **Export plot data** to external graphing software.
- Built on Postgres open-source SQL database system and JFreeChart plotting library.



Möbius Users

- **Government**:
  - Used by multiple DARPA projects for probabilistic quantification of security:
    - OASIS ITUA – Intrusion Tolerance By Unpredictable Adaptation
    - OASIS Demo/Val – DPASA – Designing Protection and Adaptation into a Survivability Architecture
  - NSF research projects on Next Generation Systems
- **Academia**:
  - Site licenses at hundreds of academic sites for teaching and research.
  - Used in graduate level system analysis course at Univ. of Illinois.
  - Many others have used Möbius and developed materials in their classes.
  - World-wide research community: Collaboration and with other researchers to further develop new capabilities: Univ. of Dortmund, Univ. of Edinburgh, Univ. of Erlangen, Univ. of Twente, Carleton University, and many others
- **Industry**:
  - Corporate licenses to a range of industries:
    - Defense/Military, satellites, telecommunications, biology/genetics
  - Adopted as one of three Motorola corporate-wide "Availability Evaluation Tools".
  - Biologists and chemists use it to model genetic and chemical reactions

Next Steps

- For more information:
  - Möbius Software Web pages (www.mobius.uiuc.edu)
  - Performability Engineering Research Group Web pages (www.perform.csl.uiuc.edu)
- Mobius is available free for academic use
- We welcome others to work with us and become Mobius developers