

Finding Short Counterexamples in Promela Models Using Estimation of Distribution Algorithms

Jan Staunton and John Clark

University of York, SEBASE EPSRC project

GECCO 2011, SBSE Track, 15th of July 2011

Presentation Outline

- Introduction to Model Checking
- EDA-based Model Checking
- Experimentation
- Summary

Introduction to Model Checking

Model Checking

- Proposed as a way of checking concurrent reactive systems*,**
- Found to be useful in checking a broad class of system properties or specifications

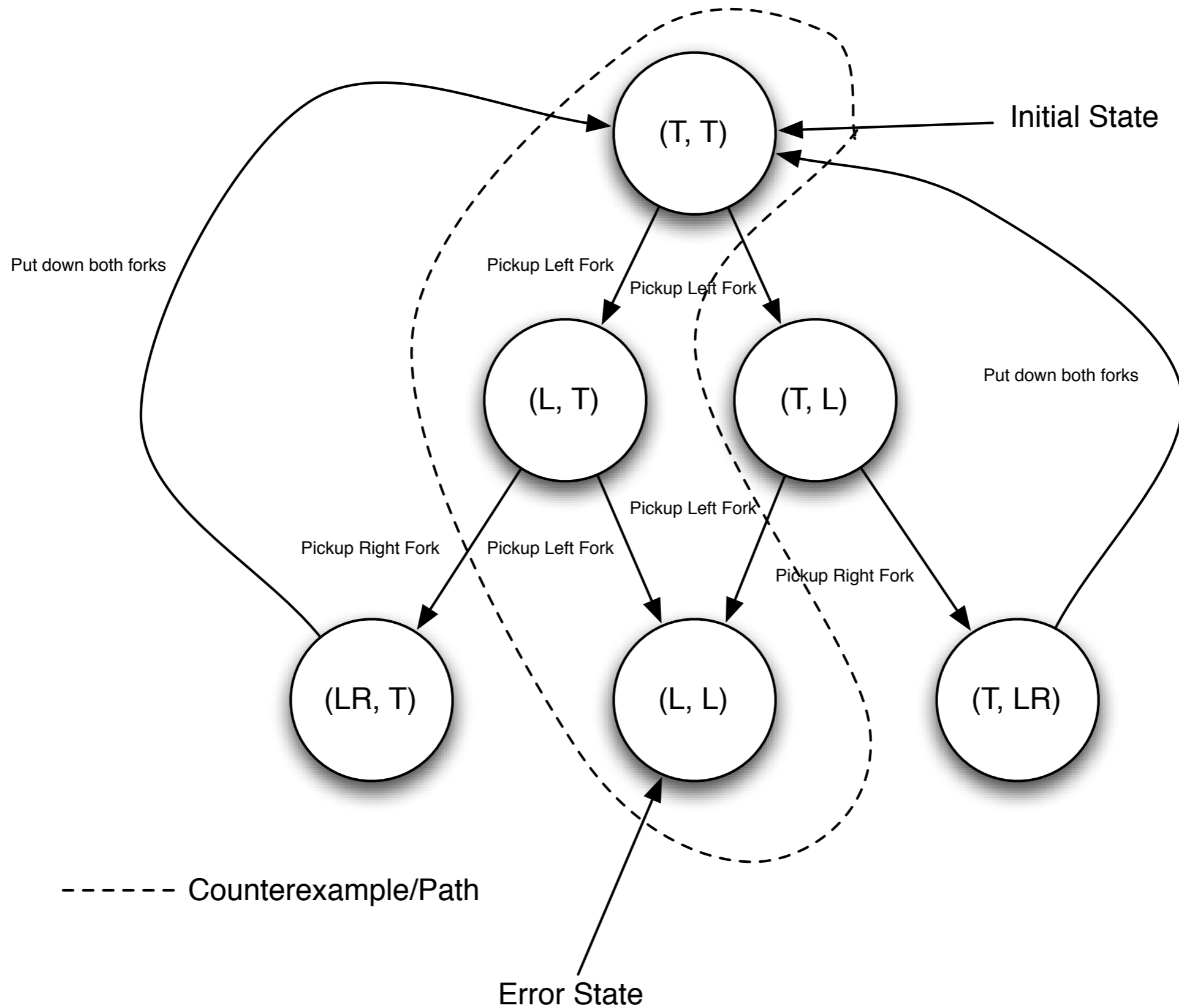
* E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Workshop on Logics of Programs, pages 52–71. Springer, 1981.

** J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In Proceedings of the 5th Colloquium on International Symposium on Programming, pages 337–351. Springer, 1982.

Basic concepts

- System under test is modelled as a state transition system
- Specifications are expressed in a propositional temporal logic
- Transition system is checked exhaustively to determine whether it is a model of the specification

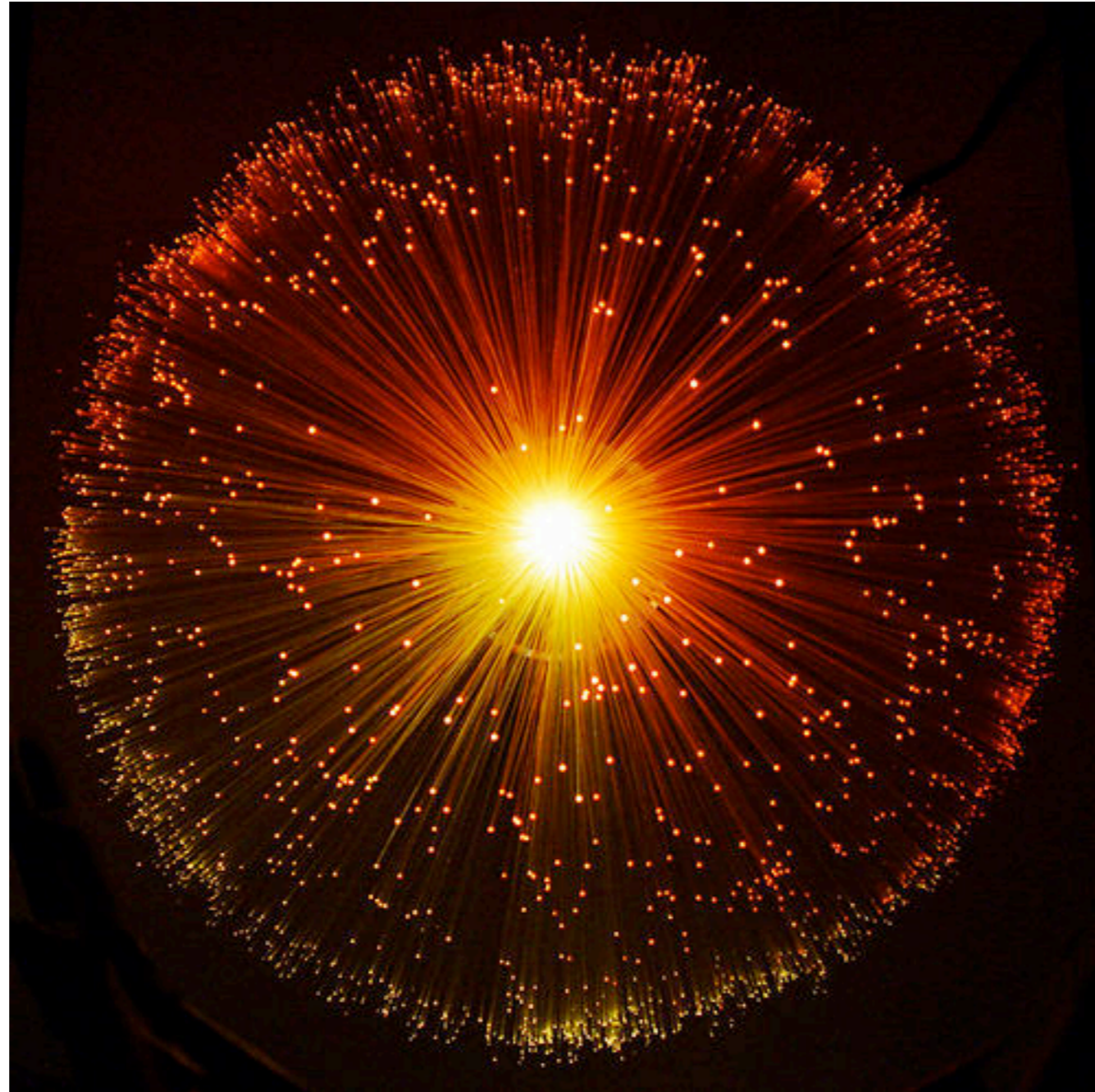
Example



It has one major
drawback... (as well as
other minor ones)

sad face

State-space explosion!



* <http://www.flickr.com/photos/jonlucas/4293334/>

Problem we aim to solve

- Find faults in concurrent programs by focusing search on paths more likely to contain faults
- Optimise/shorten errors once found
- This work uses the HSF-SPIN model checker as a workhorse to find errors in PROMELA specifications
- Deadlock, assertion and liveness errors are detected in this work

Other work using metaheuristics

- Directed model-checking* community use heuristics to help narrow down state space
- Alba et al.** and Godefroid*** have applied GAs in this arena with some success
- Ant Colony Optimisation has also been tested****

* S. Edelkamp, A.L. Lafuente, and S. Leue. Directed explicit model checking with HSF-SPIN. In Proceedings of the 8th international SPIN workshop on Model checking of software, pages 57–79. Springer-Verlag New York, Inc. New York, NY, USA, 2001.

** E. Alba, F. Chicano, M. Ferreira, and J. Gomez-Pulido. Finding deadlocks in large concurrent java programs using genetic algorithms. In Proceedings of the 10th annual conference on Genetic and evolutionary computation, pages 1735–1742. ACM New York, NY, USA, 2008.

*** P. Godefroid and S. Khurshid. Exploring very large state spaces using genetic algorithms. International Journal on Software Tools for Technology Transfer (STTT), 6(2):117–127, 2004.

**** E. Alba and F. Chicano. Finding safety errors with ACO. In Proceedings of the 9th annual conference on Genetic and evolutionary computation, pages 1066–1073. ACM Press New York, NY, USA, 2007.

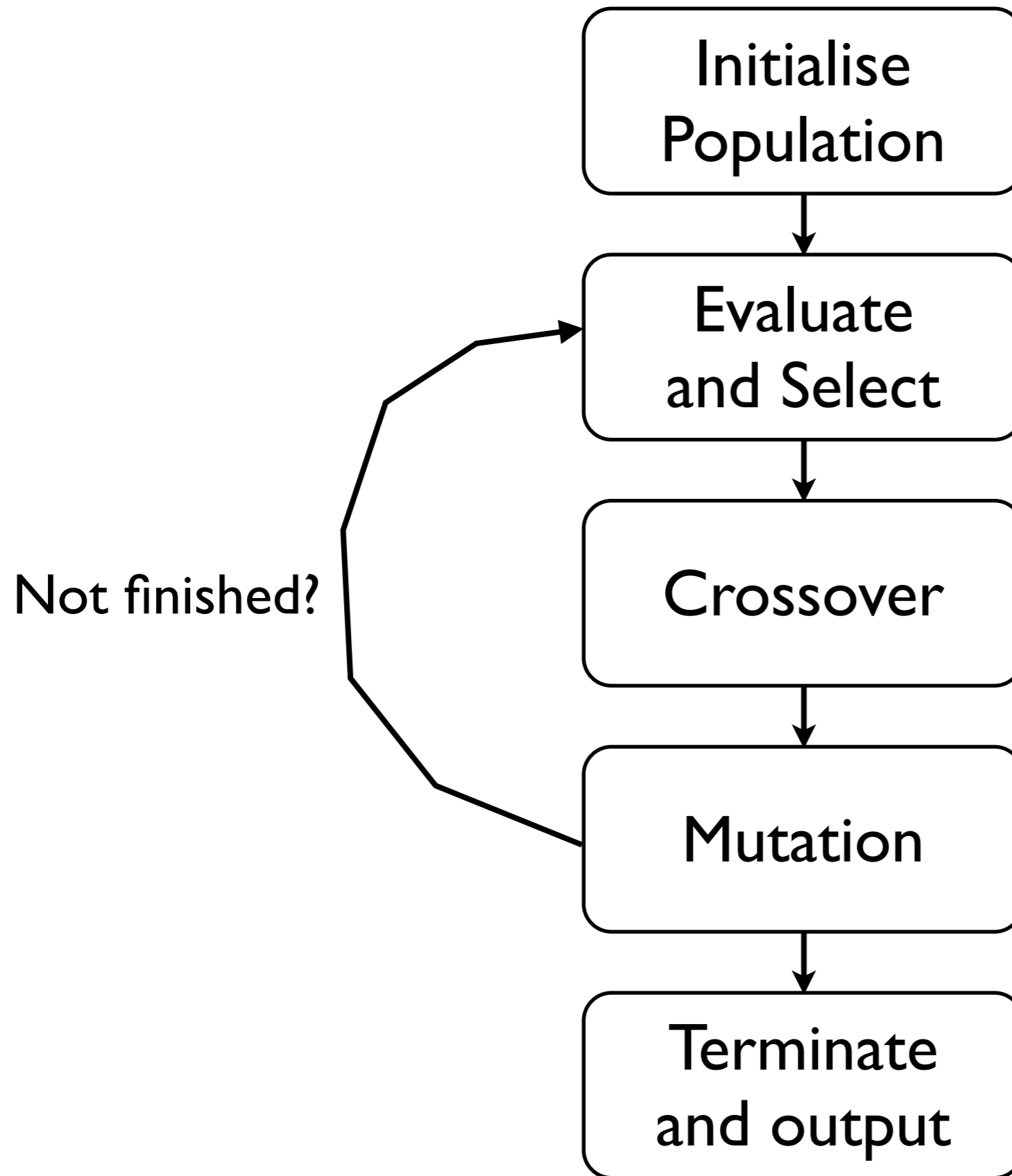
EDA-based Model Checking

Quick EDA Introduction

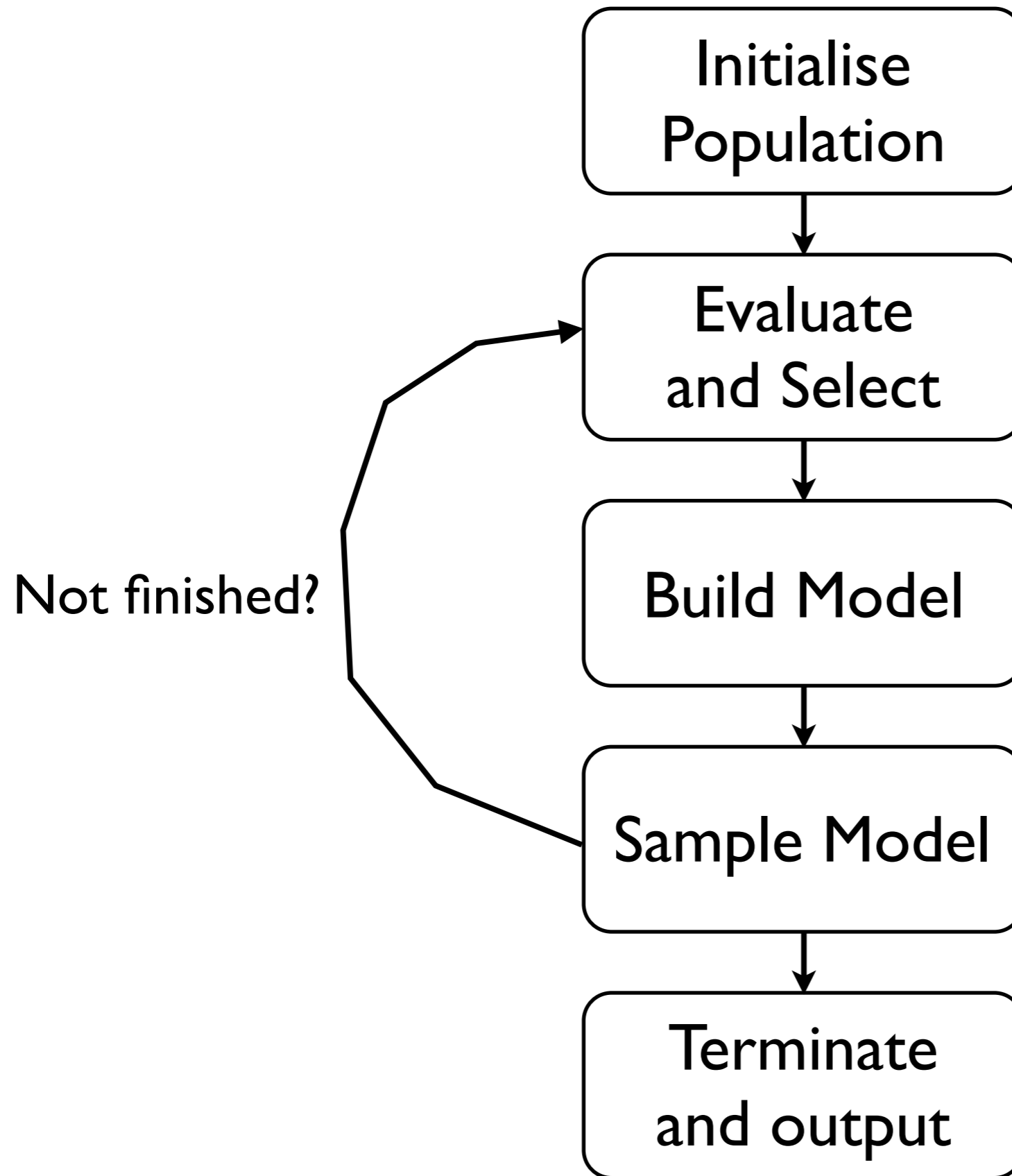
- Estimation of Distribution Algorithms (EDAs) are an effective metaheuristic search paradigm
- Similar to Genetic Algorithms (GAs)
- Replace the crossover and mutation stages with modelling and sampling stages
- Also known as Probabilistic Model Building Genetic Algorithms (PMBGAs)

* M. Pelikan, D.E. Goldberg, and F.G. Lobo. A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, 21(1):5–20, 2002.

Genetic Algorithm



Estimation of Distribution Algorithm



Encoding Paths

- In this work, a simple string representation is used
- Paths in a transition system can be viewed as a sequence of actions that cause transitions between states
- Sequences are represented as strings, alphabet is the set of possible actions in the transition system
- Alphabet is certainly modelling language specific, and may even be problem specific


Modelling Strings

- To model paths in the transition system, variant of N-gram GP* is used
- N-gram GP models the joint probabilities of n-grams of length n
- An n-gram is a subsequence of characters of length n from a longer sequence

* R. Poli and N.F. McPhee. A linear estimation-of-distribution GP system. Lecture Notes in Computer Science, 4971:206–217, 2008.

Learning the Model (2-grams)

A selected string → B A B A B B



The diagram shows a sequence of characters: B, A, B, A, B, B. The first 'B' is enclosed in a red rectangular box, and the second 'B' is enclosed in a blue rectangular box. An orange arrow points from the text 'A selected string' to the first 'B'.

Model:

Learning the Model (2-grams)

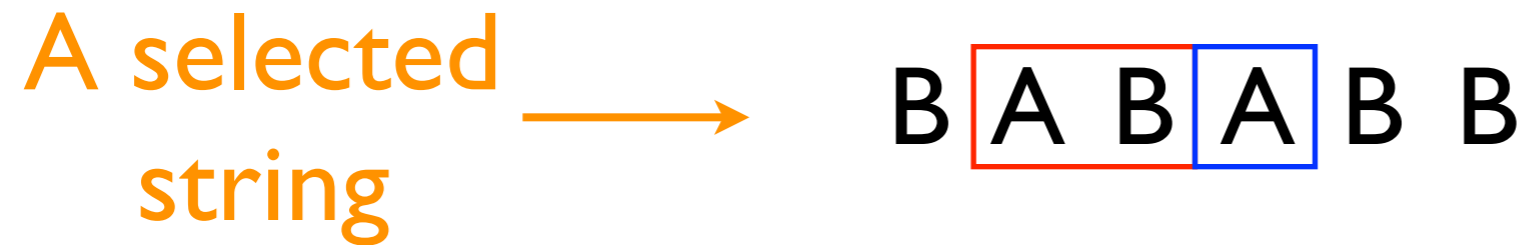
A selected string \longrightarrow **B A B** A B B

Model:

B A: B=1

Learning the Model (2-grams)

A selected string \longrightarrow B A B A B B




Model:

B A : B=1

A B : A=1

Learning the Model (2-grams)

A selected string \longrightarrow B A B A B B



Model:

B A : B=2

A B : A=1

Learning the Model (2-grams)

A selected string \longrightarrow B A B A B B

Model:

B A : B=2

A B : A=1, B=1

Learning the Model (2-grams)

A selected string \longrightarrow B A B A B B

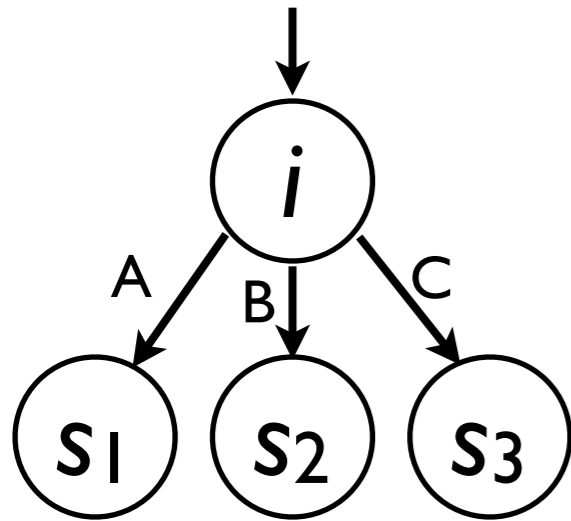
Model:

B A: B=2
A B: A=1, B=1

Normalised Model:

B A: B=1.0
A B: A=0.5, B=0.5

Sampling the Model



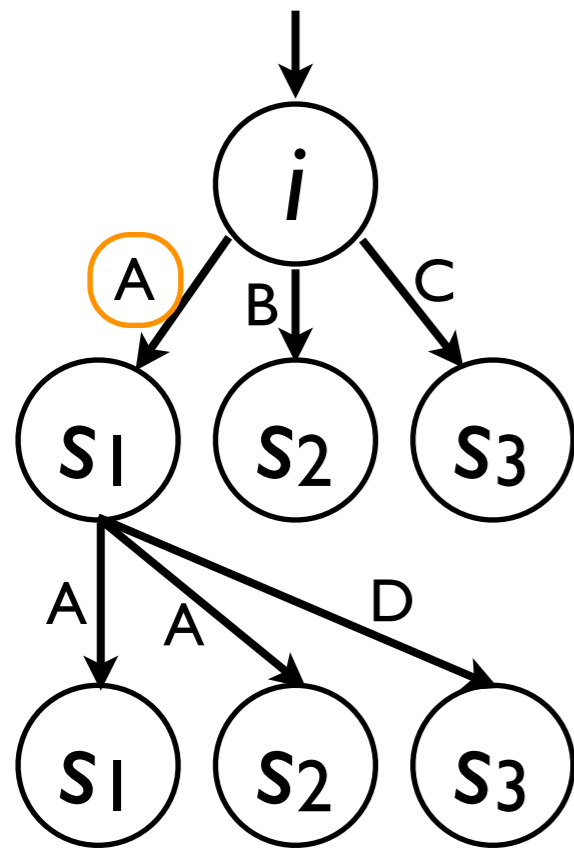
Path:

(I)

Model:

I: $A=0.8$, $B=0.15$, $C=0.05$

Sampling the Model

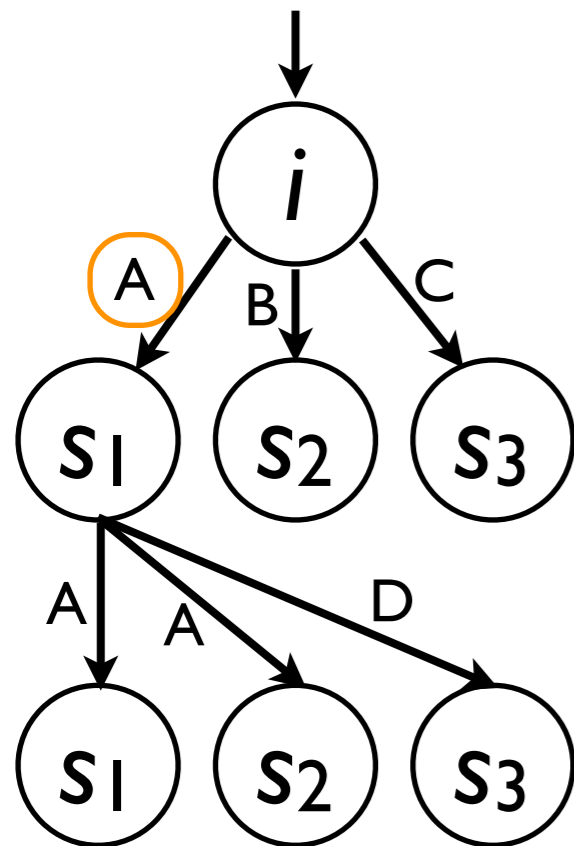


Path:
(I)

Model:

I: $A=0.8$, $B=0.15$, $C=0.05$

Sampling the Model



Path:

(I, A)

Model:

$I \ A: \ A=0.3, \ D=0.7$

A few special cases

Case 1:

Excess actions in model
for given n-gram

Solution:

Cull actions from model
(for this choice only)
and normalise

Case 2:

Some actions from state
are not “described” by
model

Solution:

Set the probabilities of
“non-described” actions
to some suitably low
value

Last special case

- Last special case is when the n-gram was not observed at all in the solutions from which the model was learned
- In this case, a “blank” distribution is constructed, with each action having an equal probability of being chosen

Fitness Function

Algorithm 1 Fitness function used to rank individuals. Individuals that are “closer” to violating a property are favoured.

Require: A, B are Individuals;
 if $A.error_found \neq B.error_found$ **then**
 return IndividualWithErrorFound(A,B);
 else if $A.error_found$ **and** $B.error_found$ **then**
 return IndividualWithShortestPath(A,B);
 else
 return IndividualWithLowestHSFSPINMetric(A,B);
 end if

Algorithm 2 HSF-SPIN heuristic metric algorithm.

Require: I is an Individual;
 aggregateMetric = 0;
 for all $States\ s \in I.Path$ **do**
 aggregateMetric += s.HSFSPINMetric;
 end for
 return aggregateMetric/LengthOfPath;

Example Path and Alphabet

```
1 (NULL TRANSITION)
2 (models/deadlock.philosophers.noloop.prm:32)(break)
3 (models/deadlock.philosophers.noloop.prm:12)(left?fork)
4 (models/deadlock.philosophers.noloop.prm:12)(left?fork)
```

Other details

- Truncation selection used
- Elitism used, one individual copied from old to new population
- Mutation operator implemented

Experimentation

Experimentation

- Implemented technique using HSF-SPIN and the ECJ toolkit
- Systems under test are PROMELA specifications
- Tested on a good number of test cases, on deadlock, assertion and liveness properties
- Compared to A^* and INDFS

Parameters

- 100 independent executions of EDA and random search technique
- Population size: 150
- N-gram size: 3
- Truncation selection used, best 20% of individuals used to build model
- Mutation probability set to 0.001
- Elitism set at 1 (one, silly font)
- Entire population is replaced with newly sampled individuals
- Initial population generated randomly
- Termination criterion: Generation limit

Test cases

- Plain Old Telephony System
- Dining Philosophers (2 systems)
- CORBA Global Inter-op Protocol (GIOP)
- Leader voting distributed algorithm
- Alternating bit protocol
- Elevator system
- Nuclear power plant operator

Results

- EDA finds errors 100% of the time, on all runs. Deterministic approaches fail on the larger test cases
- Where deterministic approaches succeed, EDA finds equivalent or shorter errors
- Finds errors consistently where previous metaheuristic efforts have failed
- Works well on toy problems and those derived from industrial code

EDA Advantages

- Computationally efficient when compared to other metaheuristic techniques (it can be thought of as ACO-lite)
- Can handle arbitrary path lengths
- Potential for model analysis to yield insights into the target problem and model reuse
- Algorithm is designed with problem in mind

Summary

Summary

- Novel spin on N-gram GP
- Presented an effective, extensible and intuitive way of searching transition systems
- Very general technique, can be applied wherever model checking is applicable
- Software model checkers exist for C, .NET languages etc...
- Possibly applicable to searching any kind of graph that has suitably labelled edges

Future Plans

- Paper to appear that uses model information between runs to reduce computational effort
- Attacking the Very Large Transition System benchmark suite for industrial case study
- Finish my thesis before my fiancée murders me

Thanks!

Any questions?

