Slicing State-based Models

Laurence Tratt
http://tratt.net/laurie/

Middlesex University

With thanks to Kelly Androutsopoulos and David Clark

2011/1/25

What is a state-based model?

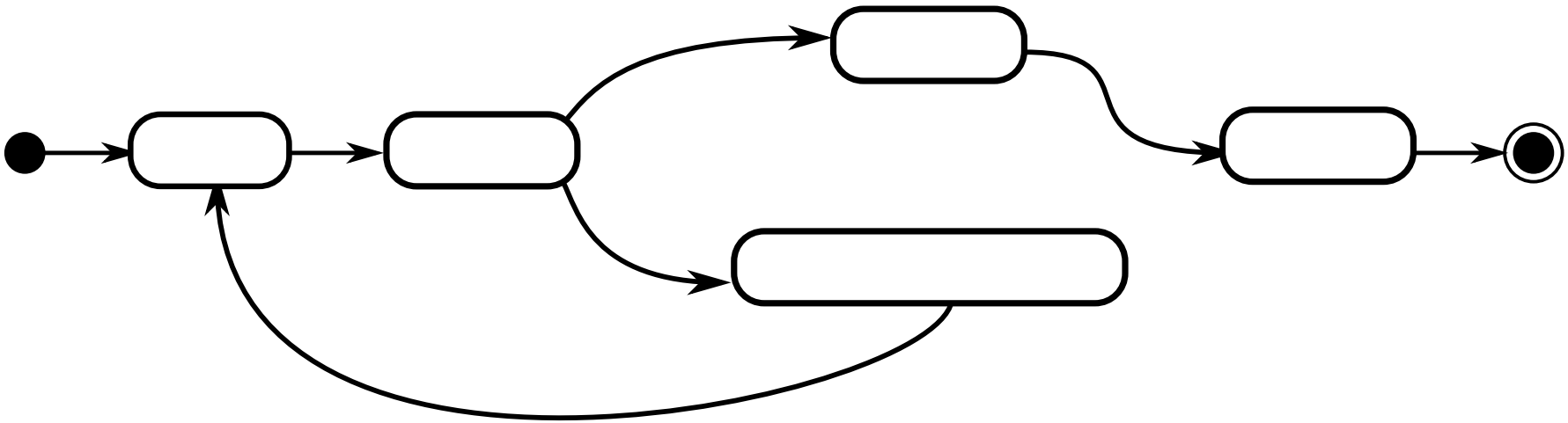
- Generic term for many languages: statecharts, statemachines, EFSMs etc.
- Fundamentally: a system is in one state until it transitions to another.

What is a state-based model?

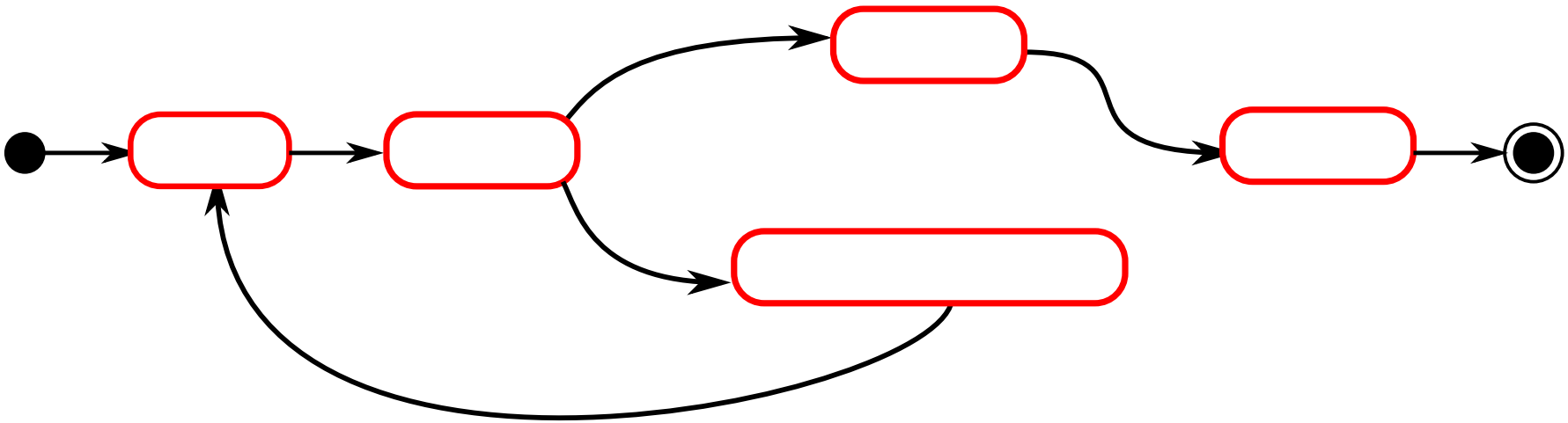
- Generic term for many languages: statecharts, statemachines, EFSMs etc.
- Fundamentally: a system is in one state until it transitions to another.
- Conventionally represented graphically.

A state-based model

A state-based model

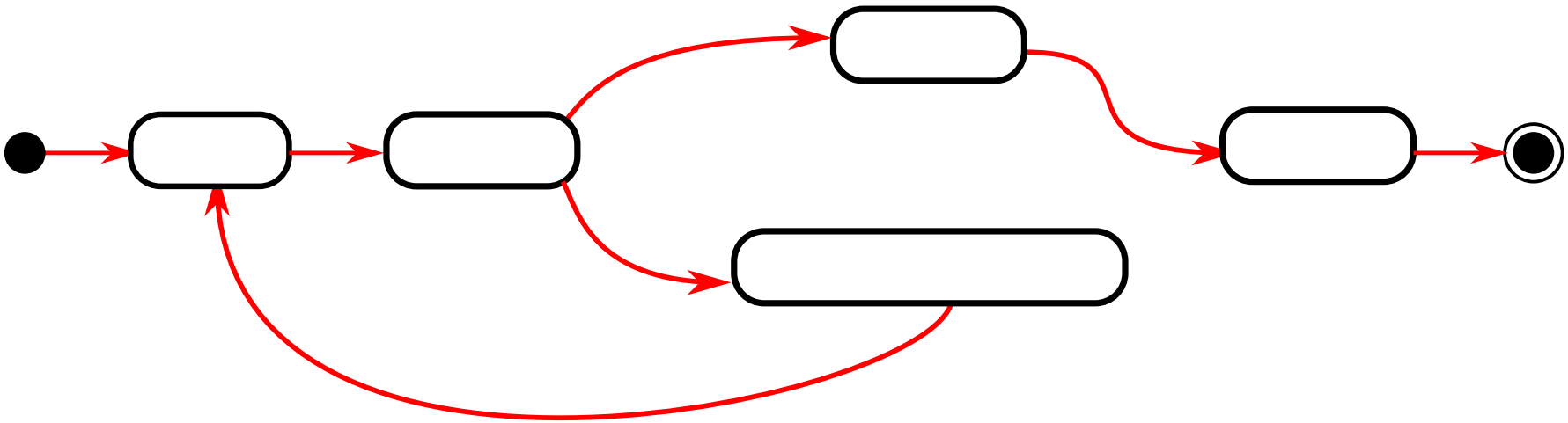


A state-based model



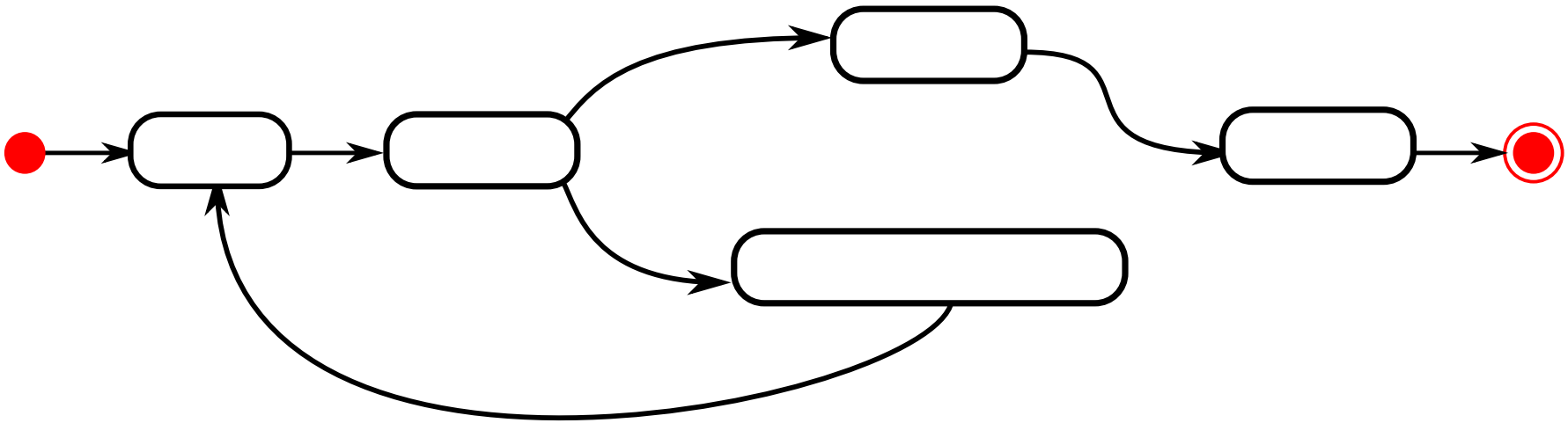
States (every language)

A state-based model



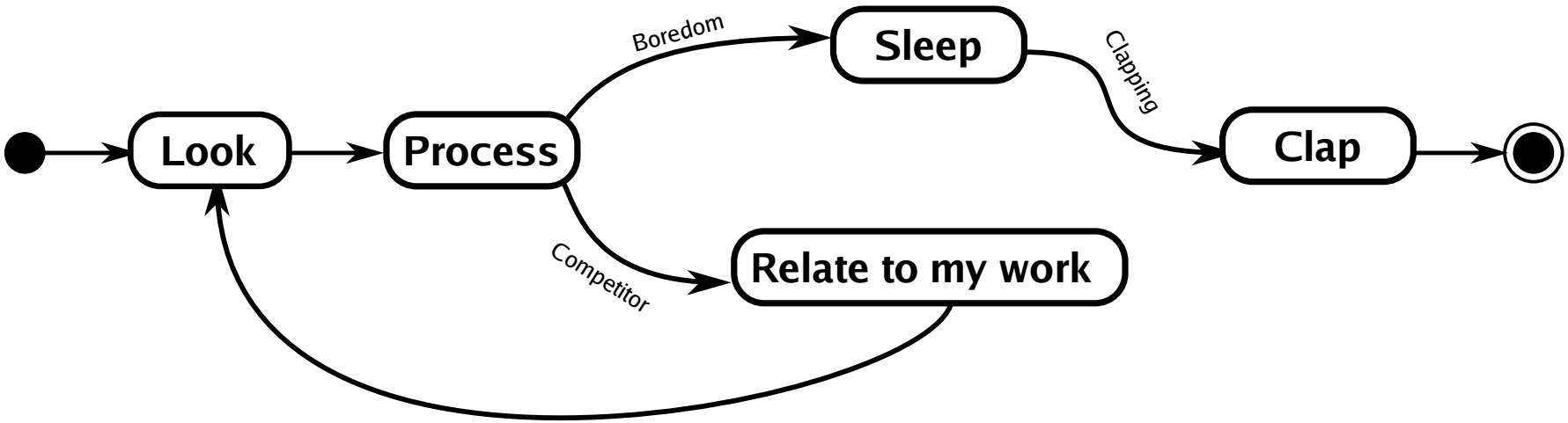
Transitions (every language): *event*[*guard*]*/action* (most languages)

A state-based model



Other bits: start / end states (varies language to language)

A state-based model



Where are state-based models used?

- Generally not an end in themselves.
- Typically used for modelling systems i.e. used as an abstraction.
- Used for computing and non-computing systems (e.g. business processes).
- Most common form probably UML statemachines.
- Short version: lots of them in the wild.

Why slice state-based models?

- Program slice: the subset of a program relevant to a slicing criterion.
- In English: ‘the minimalish parts of the program relevant to a specific point of interest.’

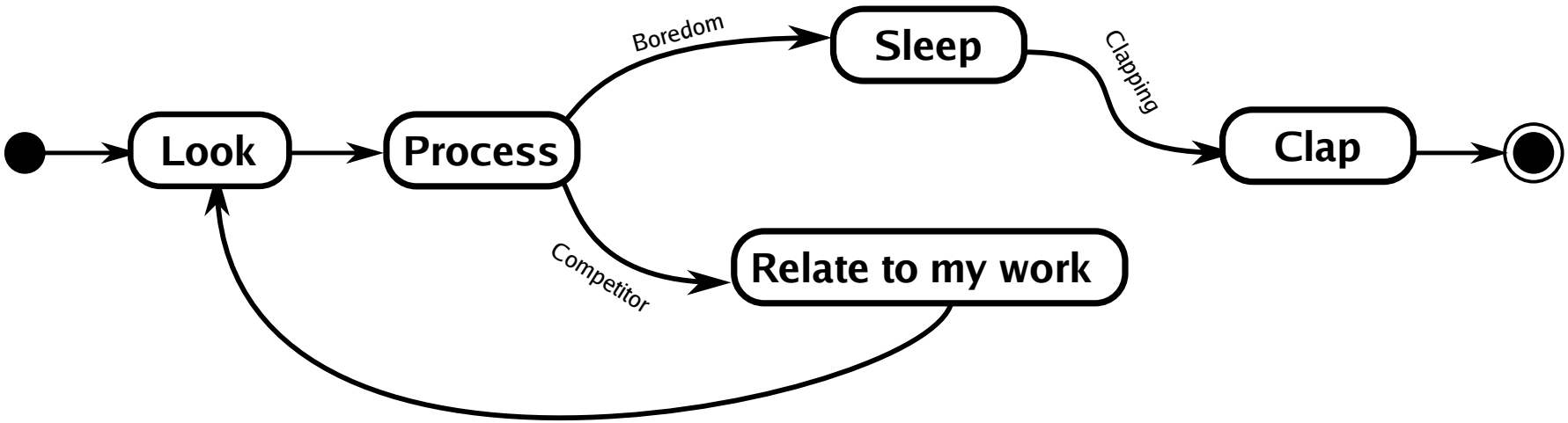
Why slice state-based models?

- Program slice: the subset of a program relevant to a slicing criterion.
- In English: ‘the minimalish parts of the program relevant to a specific point of interest.’
- Motivation? Programs are big; slice them to a manageable size.

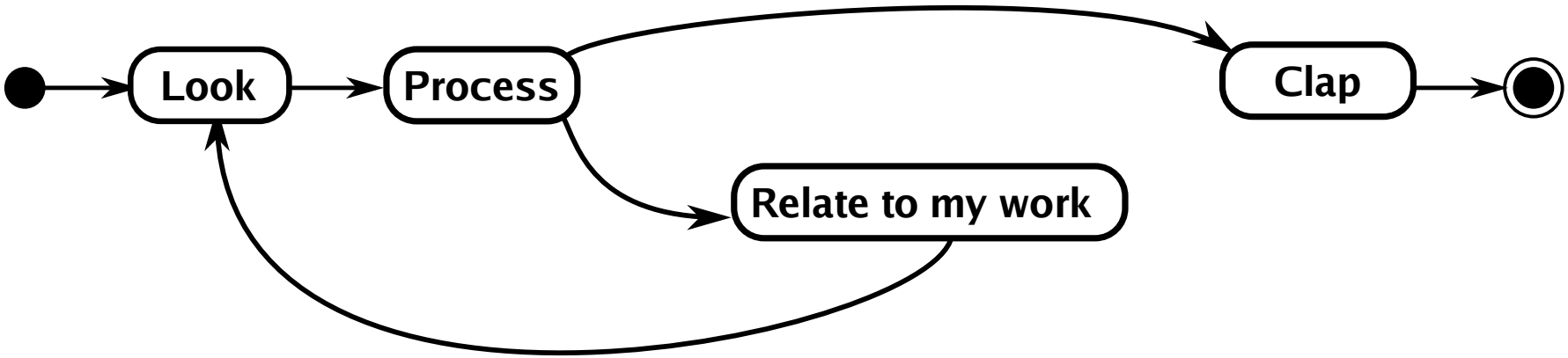
Why slice state-based models?

- Program slice: the subset of a program relevant to a slicing criterion.
- In English: ‘the minimalish parts of the program relevant to a specific point of interest.’
- Motivation? Programs are big; slice them to a manageable size.
- Same motivation for state-based models.
- State-based models can also be huge (e.g. mobile phone models).

Why slice state-based models?



Why slice state-based models?



Why not use program slicing?

- Assertion: SBMs can be encoded as programs and vice versa.

Why not use program slicing?

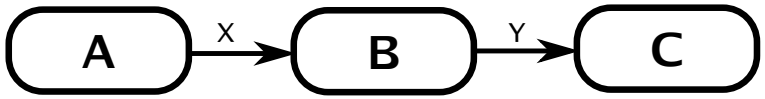
- Assertion: SBMs can be encoded as programs and vice versa.
- Intuition: transitions are just `goto`s.

Why not use program slicing?

- Assertion: SBMs can be encoded as programs and vice versa.
- Intuition: transitions are just `goto`s.
- Question: why not translate SBMs to programs and slice those?

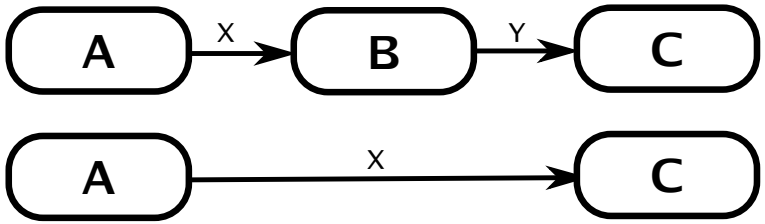
Why SBM slicing is different (1)

- Input SBM:



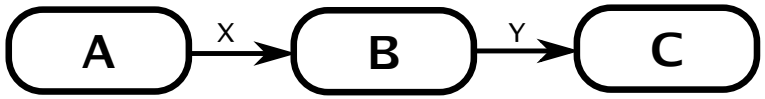
Why SBM slicing is different (1)

- Input SBM:
- What we hope for:

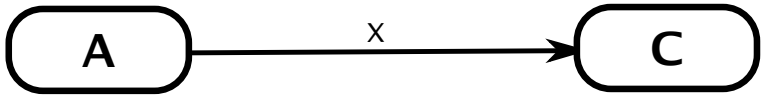


Why SBM slicing is different (1)

- Input SBM:



- What we hope for:

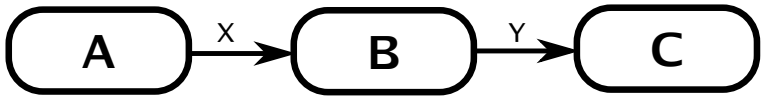


- Translate to:

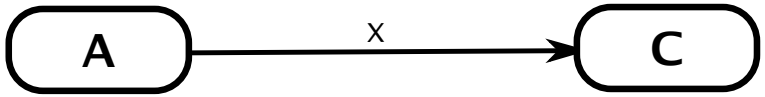
```
state := A; if (next_event() == X) goto B else ...
B: state := B; if (next_event() == Y) goto C else ...
C: state := C
```

Why SBM slicing is different (1)

- Input SBM:



- What we hope for:



- Translate to:

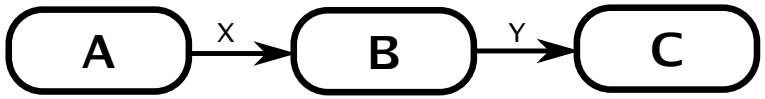
```
state := A; if (next_event() == X) goto B else ...
B: state := B; if (next_event() == Y) goto C else ...
C: state := C
```

- To work, need to rewrite to:

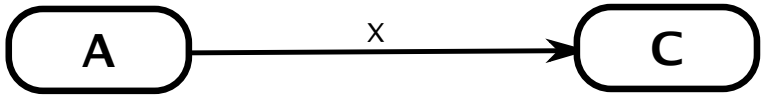
```
state := A; if (next_event() == X) goto C else ?
C: state := C
```

Why SBM slicing is different (1)

- Input SBM:



- What we hope for:



- Translate to:

```
state := A; if (next_event() == X) goto B else ...
B: state := B; if (next_event() == Y) goto C else ...
C: state := C
```

- To work, need to rewrite to:

```
state := A; if (next_event() == X) goto C else ?
C: state := C
```

- Slicing `goto` programs is hard and difficult.
- SBM slicing has to acknowledge (and make use of) graph structures.

Why SBM slicing is different (2)

- Assume #1 doesn't occur.

Why SBM slicing is different (2)

- Assume #1 doesn't occur.
- The chances of program slicing respecting the granularity of the SBM are small.
- A program slicer might produce:

```
state := A
B: if (next_event() == Y) goto C;
C: state := C
```

Why SBM slicing is different (2)

- Assume #1 doesn't occur.
- The chances of program slicing respecting the granularity of the SBM are small.
- A program slicer might produce:

```
state := A
B: if (next_event() == Y) goto C;
C: state := C
```

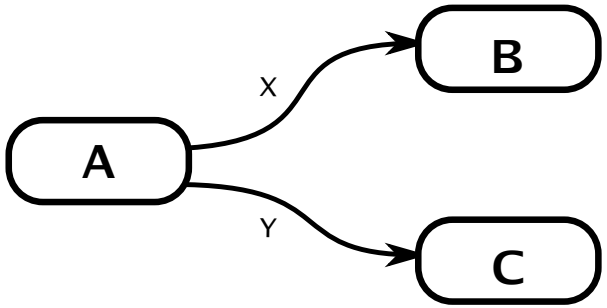
- What SBM does that map back to?
- Fundamentally: we need an isomorphism between $A \rightarrow B$ and $B \rightarrow A'$, but program slicing can't provide that.

Why SBM slicing is different (3)

- SBM features map nicely onto PL features and vice versa.

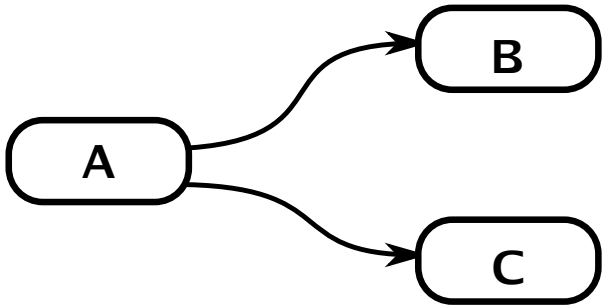
Why SBM slicing is different (3)

- SBM features map nicely onto PL features and vice versa.
- ...except one...



Why SBM slicing is different (3)

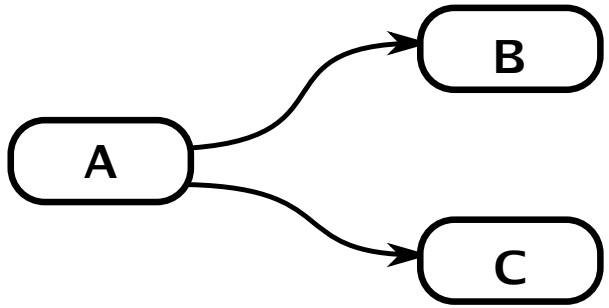
- SBM features map nicely onto PL features and vice versa.
- ...except one...



- Not many programming languages with non-determinism...

Why SBM slicing is different (3)

- SBM features map nicely onto PL features and vice versa.
- ...except one...



- Not many programming languages with non-determinism...
- So we have to encode that. Two problems:
  1. What non-determinism to encode? SBM formalisms rarely define how to resolve non-determinism; irrelevant for specifications.
  2. Translating back has problems (#2 in disguise).

Applications

- As program slicing, many potential applications.

Applications

- As program slicing, many potential applications.
- Model comprehension.

Applications

- As program slicing, many potential applications.
- Model comprehension.
- Model checking.

Applications

- As program slicing, many potential applications.
- Model comprehension.
- Model checking.
- Testing.

Applications

- As program slicing, many potential applications.
- Model comprehension.
- Model checking.
- Testing.
- etc.

The SLIM project

- SLIcing state based Models (SLIM).
- EPSRC funded project, first at King's and now at UCL (and a few hangers on like me).

The SLIM project

- SLIcing state based Models (SLIM).
- EPSRC funded project, first at King's and now at UCL (and a few hangers on like me).
- Building upon previous work (e.g. Korel et al.) and advancing the art w.r.t. SBM slicing.

Problem #1: dependency

- Data dependency is easy.
- Control dependency (CD) isn't—but it's the heart of slicing!

Problem #1: dependency

- Data dependency is easy.
- Control dependency (CD) isn't—but it's the heart of slicing!
- Underlying question: should CD be sensitive or insensitive to non-termination?

Problem #1: dependency

- Data dependency is easy.
- Control dependency (CD) isn't—but it's the heart of slicing!
- Underlying question: should CD be sensitive or insensitive to non-termination?
- Dealing with non-terminating SBMs is non-trivial.
- No existing definition ideal.

Problem #1: dependency

- Data dependency is easy.
- Control dependency (CD) isn't—but it's the heart of slicing!
- Underlying question: should CD be sensitive or insensitive to non-termination?
- Dealing with non-terminating SBMs is non-trivial.
- No existing definition ideal.
- New CD definition *UNTICD* (FASE 2009) subsumes Korel et al.'s CD definition; and has neat correspondences with NTICD and NTSCD.

Problem #1: dependency (2)

- Realisation: ‘best’ dependency depends on your application.
- e.g. NTSCD good for model-checking; UNTICD better for many other uses.

Problem #1: dependency (2)

- Realisation: ‘best’ dependency depends on your application.
- e.g. NTSCD good for model-checking; UNTICD better for many other uses.
- Adapted WOD (Amtoft et al.) for SBMs.
- Differs significantly from UNTICD when no data dependency; subtly different in other ways.

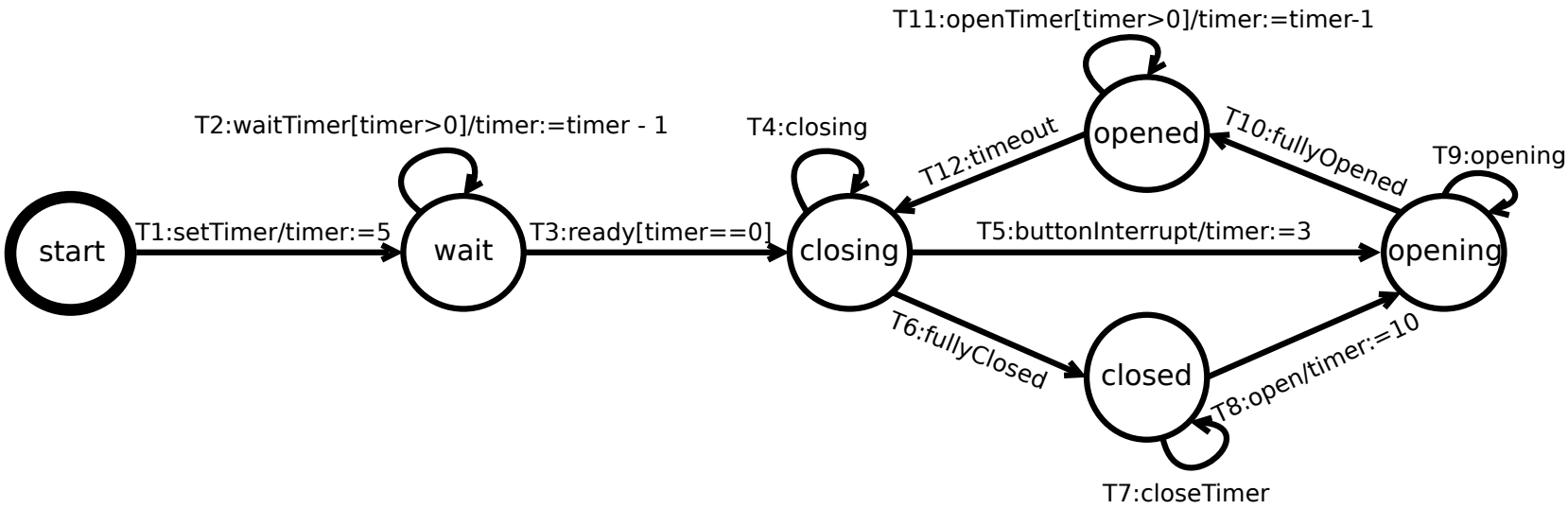
Problem #1: dependency (2)

- Realisation: ‘best’ dependency depends on your application.
- e.g. NTSCD good for model-checking; UNTICD better for many other uses.
- Adapted WOD (Amtoft et al.) for SBMs.
- Differs significantly from UNTICD when no data dependency; subtly different in other ways.
- Several empirical studies done (under review) to help determine which CD is best.
- Perhaps work to do: both UNTICD and WOD can introduce new non-determinism.

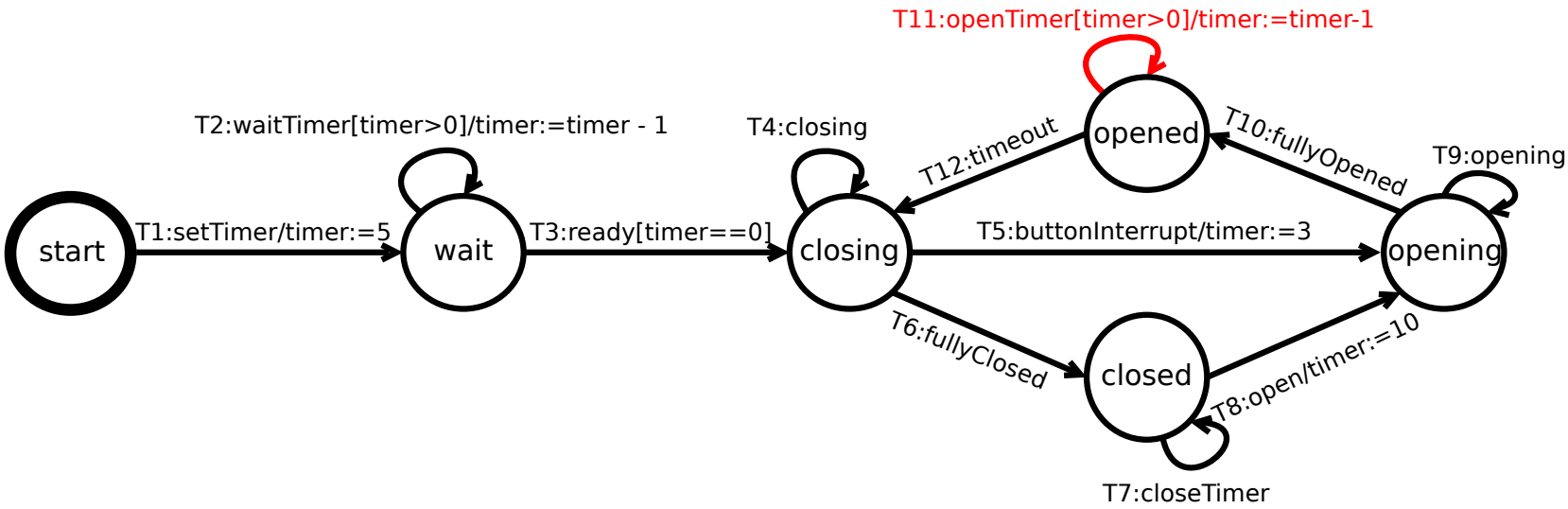
Problem #2: slicing

- Korel et al.'s algorithms ([1] and [2]) the only amorphous SBM slicers.
- Tends to produce large slices, but not easy to improve upon.

Lift example



Lift example

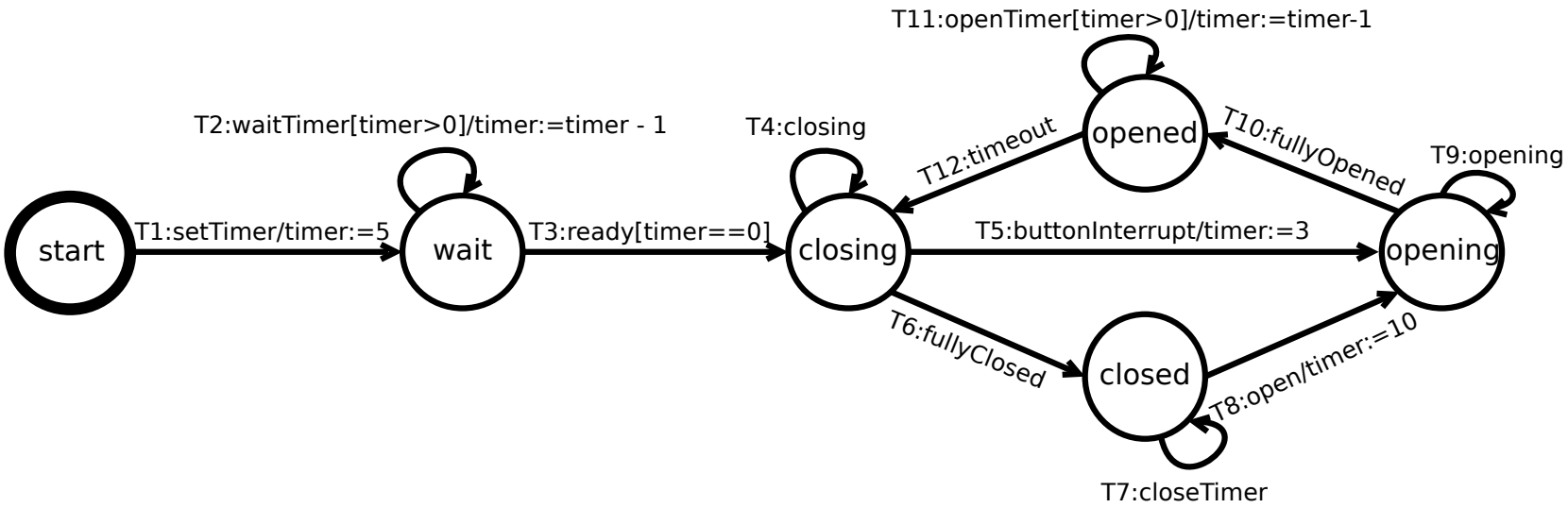


Amorphous slicing algorithm

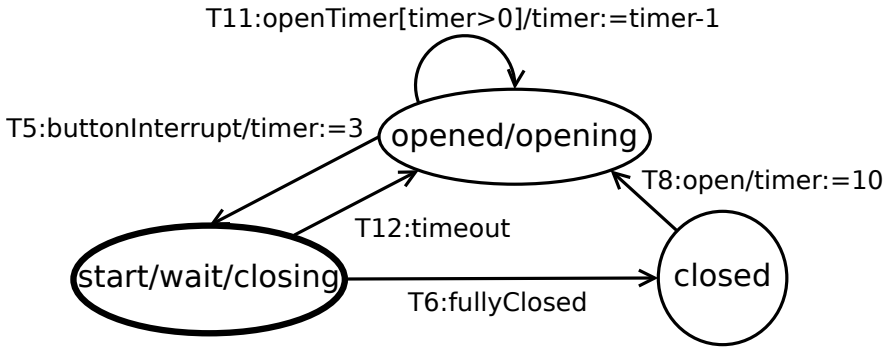
Essentials:

1. $M_{post} \leftarrow M_{pre}$
2. DG $\leftarrow$ compute_dependence_graph(CD_{def}, M_{pre})
3. $M_{post} \leftarrow$ traverse_backwards_marking(t_{sc}, V_{sc}, DG)
4. $M_{post} \leftarrow$ anonymise_unmarked_transitions(M_{post})
5. **while** apply_rule1(M_{post}) **or** apply_rule2(M_{post}) **or** apply_rule3(M_{post}) **do**
6: **end while**
7: apply_epsilon_elimination(M_{post})
8: garbage_collect(M_{post})
9. $S_{toMerge} \leftarrow$ right_invariant_equivalence(M_{post})
10. merge_states($S_{toMerge}, M_{post}$)
11. $S_{toMerge} \leftarrow$ left_invariant_equivalence(M_{post})
12. merge_states($S_{toMerge}, M_{post}$)
13. **return** M_{post}

Lift example

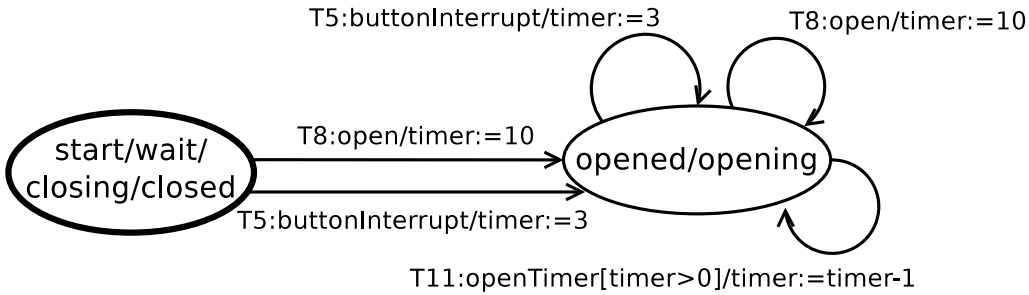


Lift example



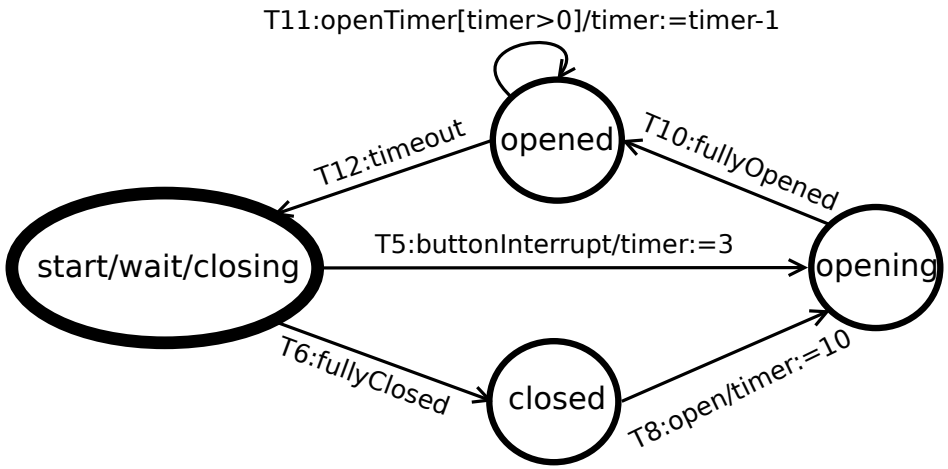
Korel et al.'s slicing algorithm[2] with NTICD

Lift example



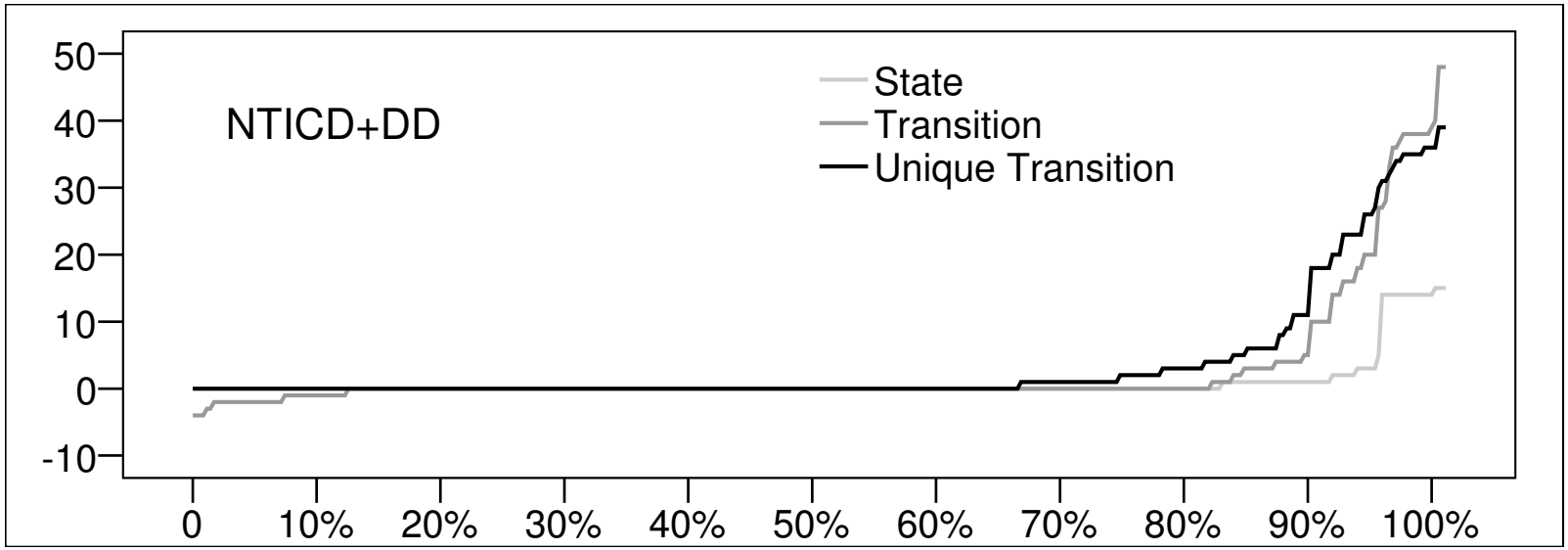
SLIM(NTICD)

Lift example



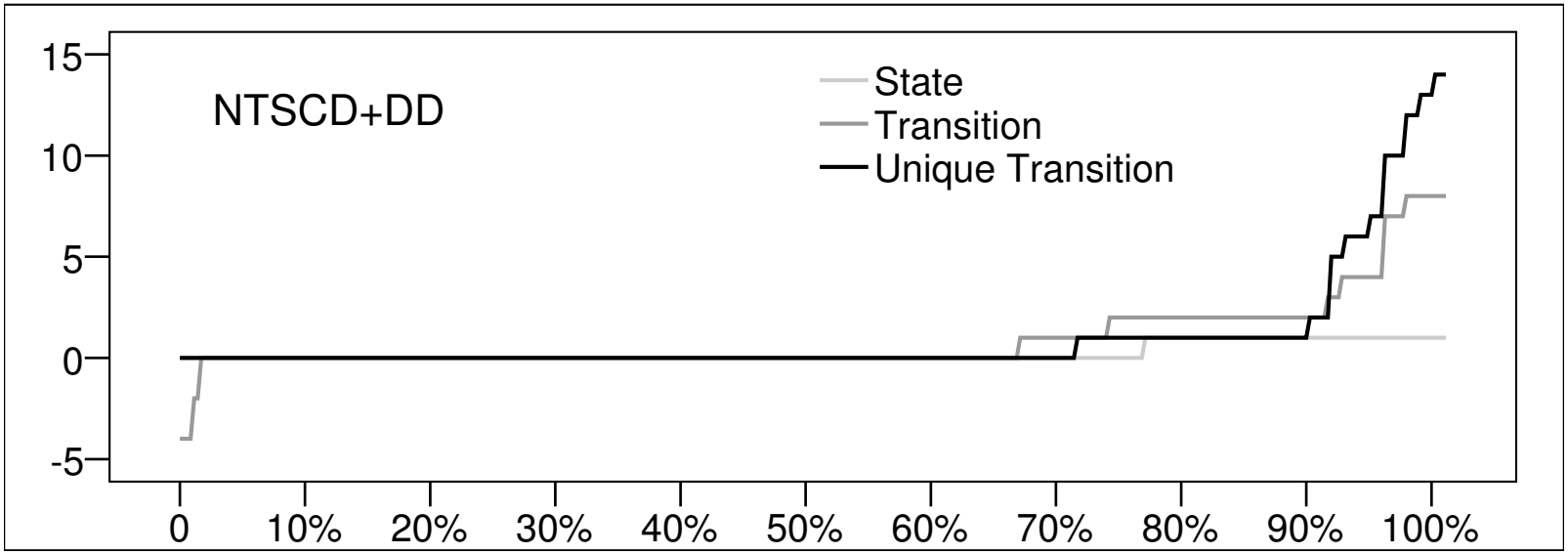
SLIM(UNTICD)

Relative slice sizes



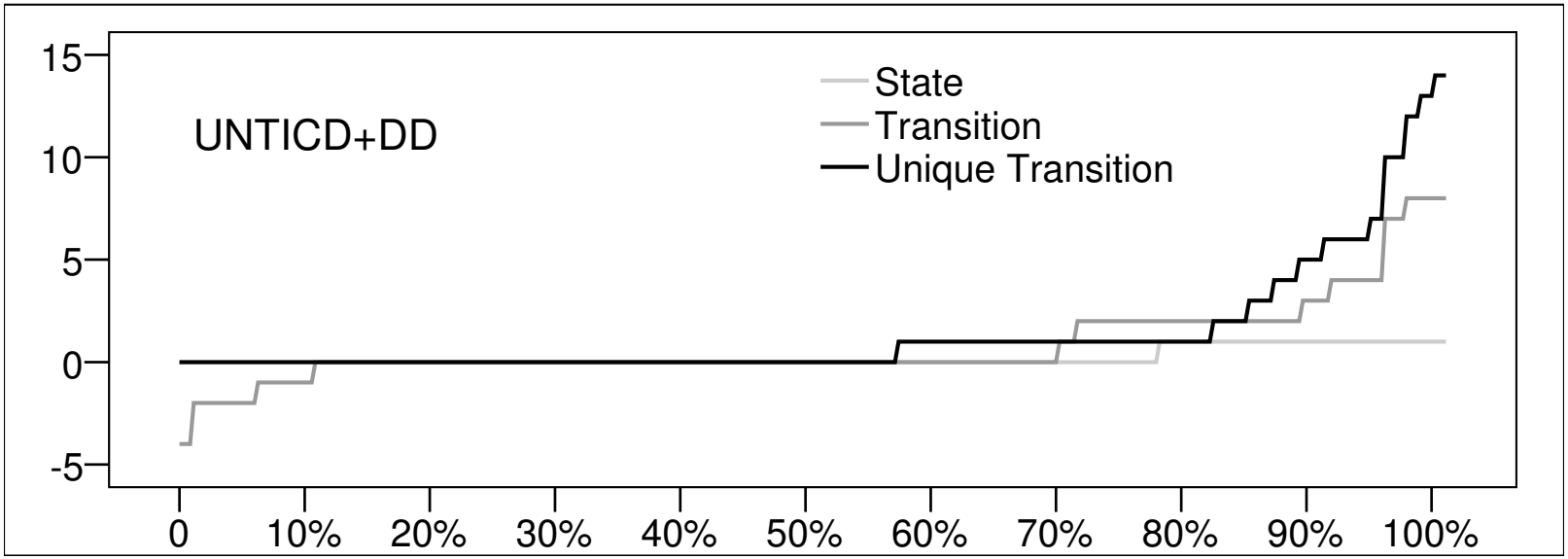
SLIM & Korel et. al's algorithm[2] with NTICD

Relative slice sizes



SLIM & Korel et. al's algorithm[2] with NTSCD

Relative slice sizes



SLIM & Korel et. al's algorithm[2] with UNTICD

Software

- The CREST slicing tool.
- Python tool which can slice SBMs in several different ways.
- Pluggable control dependency etc.
- Able to visualize huge SBMs with Graphviz.
- To appear soonish.

Summary

- Slicing SBMs is useful and not easy.
- SLIM has pushed the state of the art forward considerably.
- Ready for real-world in some parts; in others, important questions to be answered.

Summary

- Slicing SBMs is useful and not easy.
- SLIM has pushed the state of the art forward considerably.
- Ready for real-world in some parts; in others, important questions to be answered.

Thanks for listening