

Relaxations allow trading

correctness
for
performance

Relaxations allow trading

correctness
for
performance

provide the **potential**
for better-performing
implementations

Relaxing the Semantics

Relaxing the Semantics

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Relaxing the Semantics

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL13

- **Sequential specification** = set of legal sequences
- **Consistency condition** = e.g. linearizability / sequential consistency

Relaxing the Semantics

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin, S. POPL13

- **Sequential specification** = set of legal sequences
- **Consistency condition** = e.g. linearizability / sequential consistency

Local linearizability
Haas, Henzinger, Holzer, ..., S, Veith CONCUR16

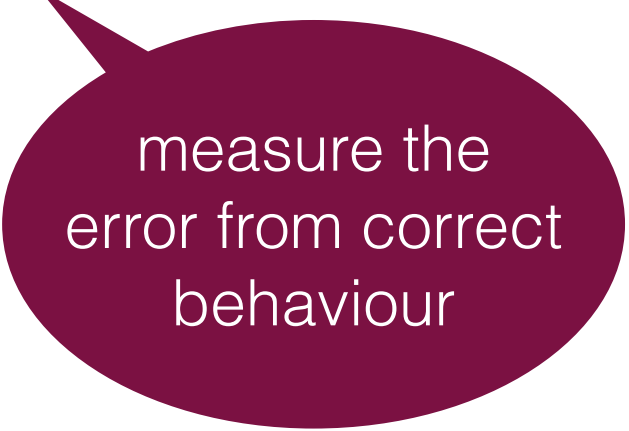
Relaxing the Sequential Specification

Relaxing the Sequential Specification

Quantitative
relaxations
(POPL13)

Goal

- trade correctness for performance
- in a controlled way with quantitative bounds



measure the
error from correct
behaviour

Goal

Stack - incorrect behavior

```
push(a)push(b)push(c)pop(a)pop(b)
```

- trade correctness for performance
- in a controlled way with quantitative bounds

measure the
error from correct
behaviour

Goal

Stack - incorrect behavior

```
push(a)push(b)push(c)pop(a)pop(b)
```

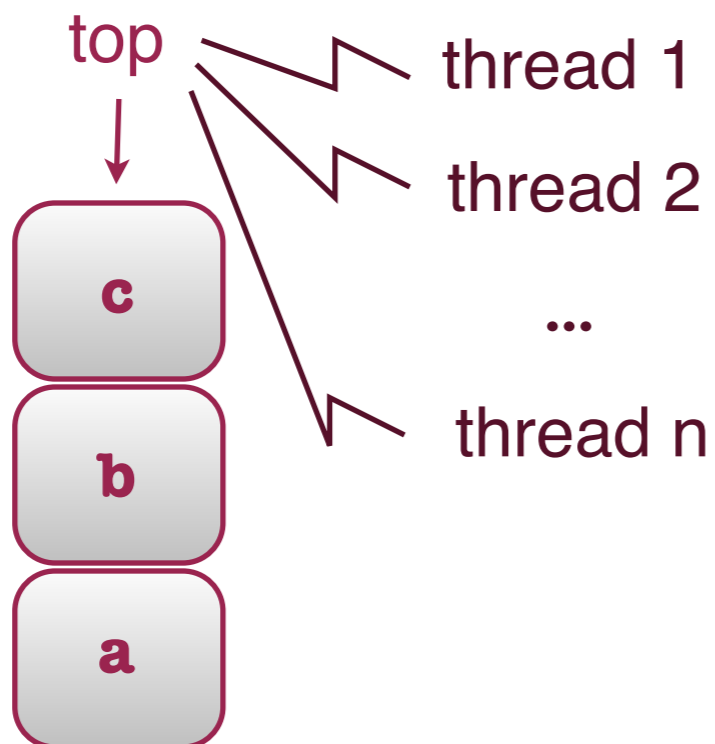
- trade correctness for performance
- in a controlled way with quantitative bounds

correct in a relaxed stack
... 2-relaxed? 3-relaxed?

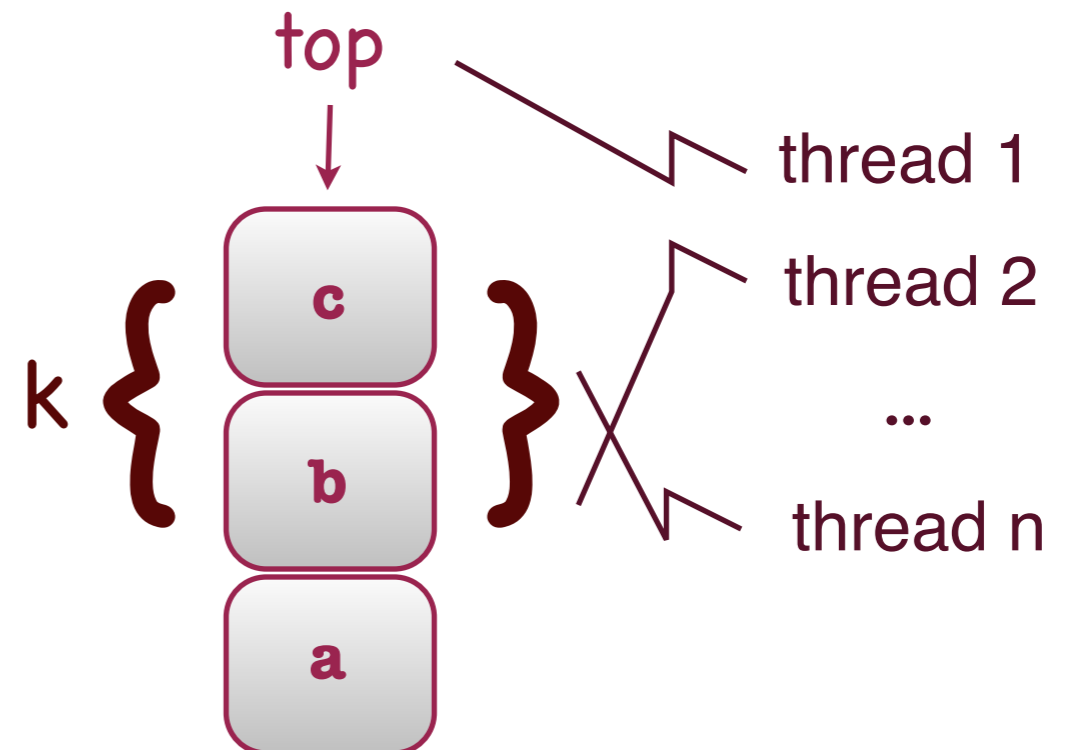
measure the
error from correct
behaviour

How can relaxing help?

Stack



k-Relaxed stack



What we have

- Framework

for semantic relaxations

- Generic examples

out-of-order /
stuttering

- Concrete relaxation examples

stacks, queues,
priority queues,.. /
CAS, shared counter

- Efficient concurrent implementations

of relaxation instances

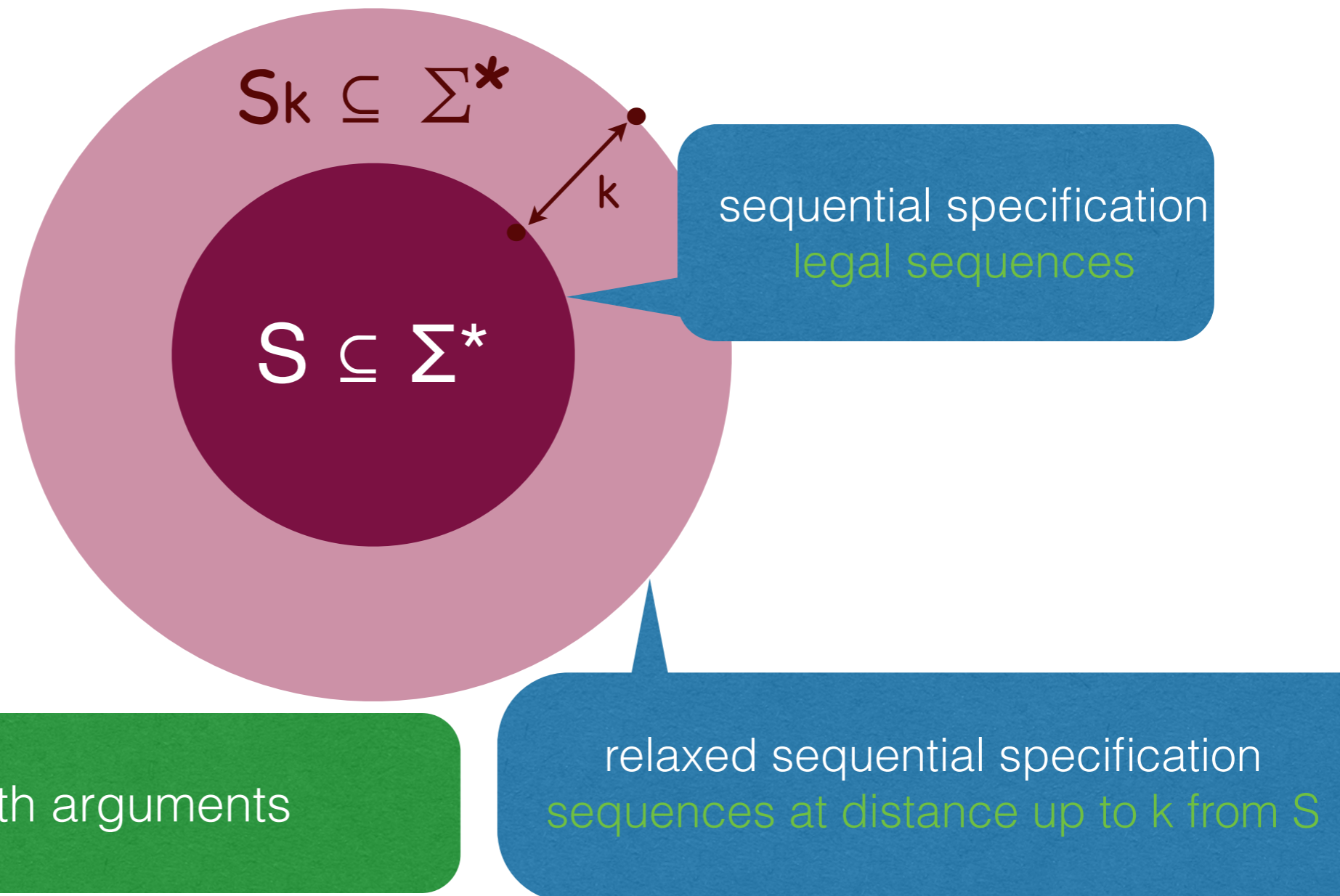
The big picture

$$S \subseteq \Sigma^*$$

sequential specification
legal sequences

Σ - methods with arguments

The big picture



Σ - methods with arguments

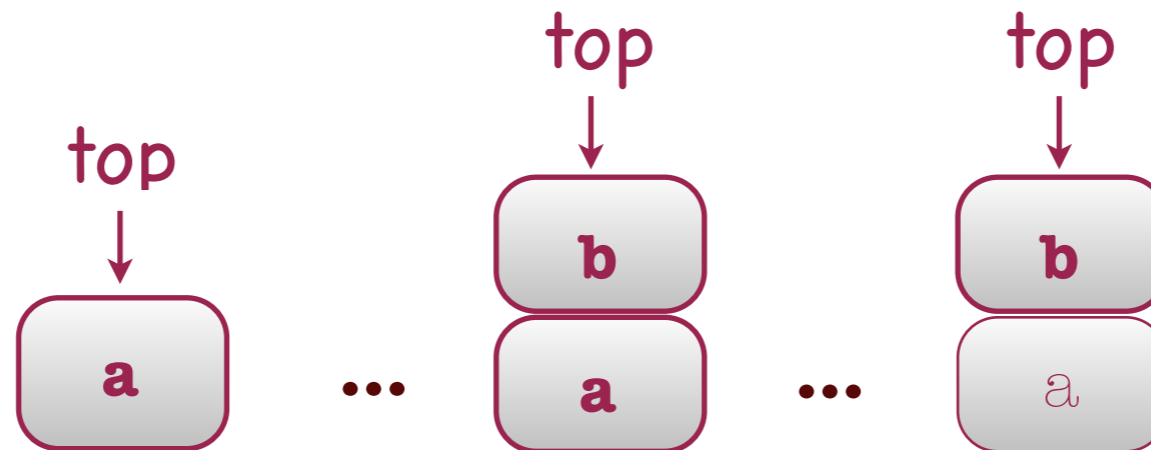
Syntactic distances do not help

$\text{push}(a)[\text{push}(i)\text{pop}(i)]^n\text{push}(b)[\text{push}(j)\text{pop}(j)]^m\text{pop}(a)$

Syntactic distances do not help

$\text{push}(a)[\text{push}(i)\text{pop}(i)]^n\text{push}(b)[\text{push}(j)\text{pop}(j)]^m\text{pop}(a)$

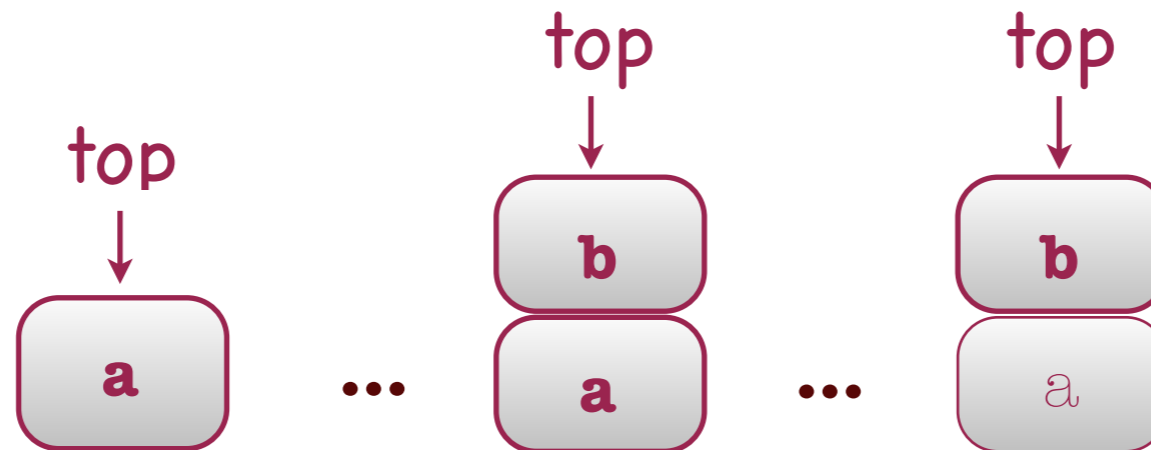
is a 1-out-of-order stack sequence



Syntactic distances do not help

$\text{push}(a)[\text{push}(i)\text{pop}(i)]^n\text{push}(b)[\text{push}(j)\text{pop}(j)]^m\text{pop}(a)$

is a 1-out-of-order stack sequence



its permutation distance is $\min(2n, 2m)$

Semantic distances need a notion of state

- States are equivalence classes of sequences in S
- Two sequences in S are equivalent iff they have an indistinguishable future

Semantic distances need a notion of state

- States are equivalence classes of sequences in S
- Two sequences in S are equivalent iff they have an indistinguishable future

$$x \equiv y \iff \forall u \in \Sigma^*. (xu \in S \iff yu \in S)$$

Semantic distances need a notion of state

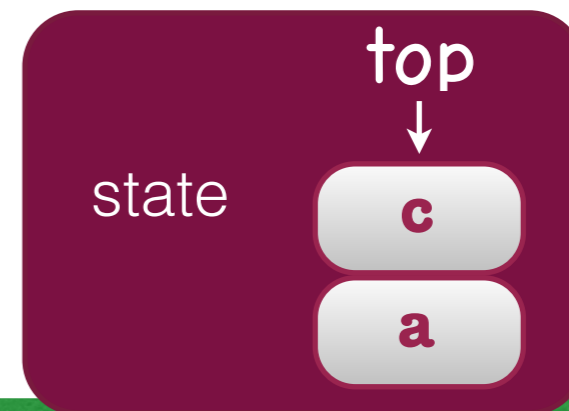
- States are equivalence classes of sequences in S

example: for stack

$\text{push}(a)\text{push}(b)\text{pop}(b)\text{push}(c) \equiv \text{push}(a)\text{push}(c)$

- Two sequences in S are equivalent iff they have an indistinguishable future

$$x \equiv y \iff \forall u \in \Sigma^*. (xu \in S \iff yu \in S)$$



Semantics goes operational

$S \subseteq \Sigma^*$ is the sequential specification

states

labels

initial state

$\text{LTS}(S) = (S/\equiv, \Sigma, \rightarrow, [\varepsilon]_{\equiv})$ with

transition relation

$$[s]_{\equiv} \xrightarrow{m} [sm]_{\equiv} \iff sm \in S$$

Semantics goes operational

$S \subseteq \Sigma^*$ is the sequential specification

states

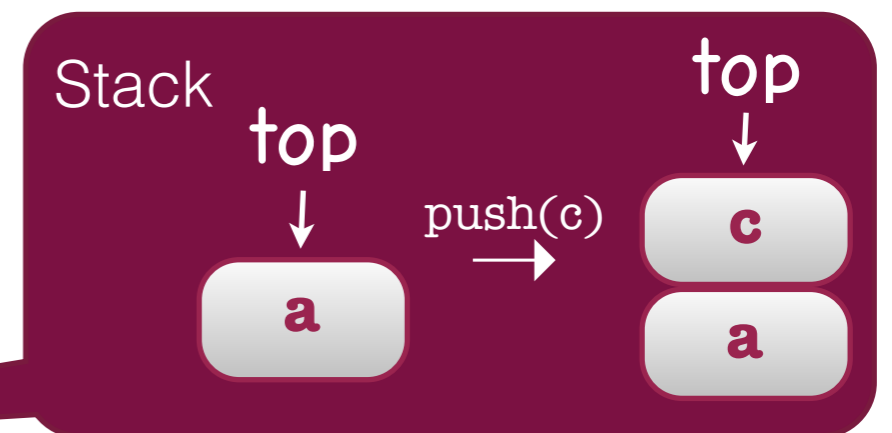
labels

initial state

$LTS(S) = (S/\equiv, \Sigma, \rightarrow, [\varepsilon]_{\equiv})$ with

transition relation

$$[s]_{\equiv} \xrightarrow{m} [sm]_{\equiv} \iff sm \in S$$

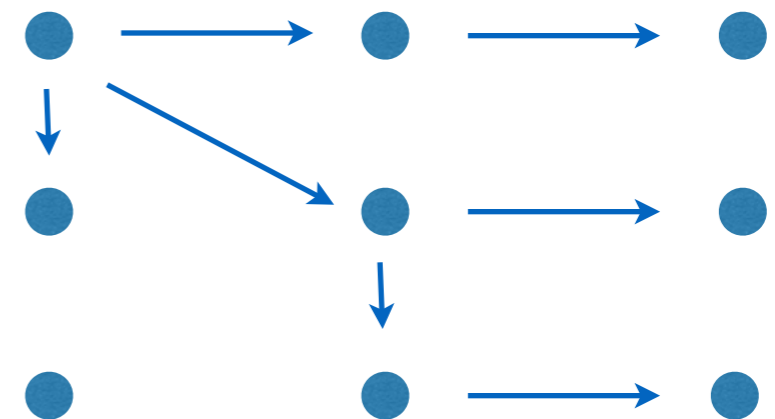


The relaxation framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function

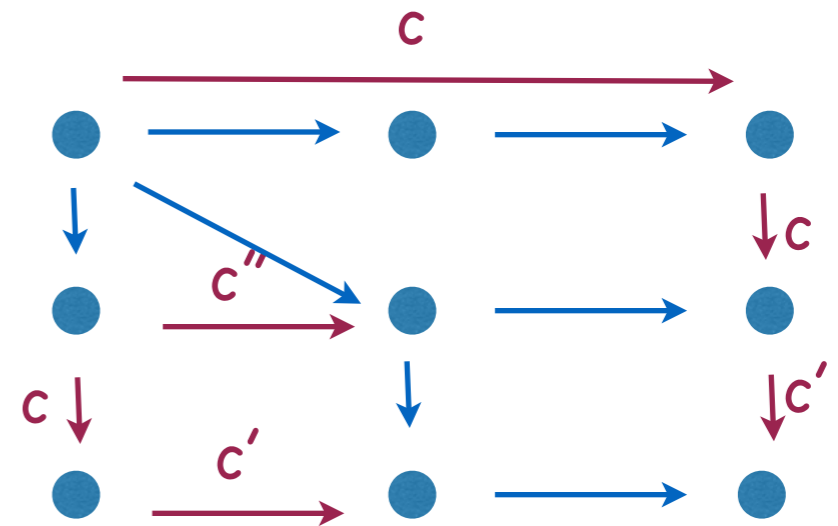
The relaxation framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function



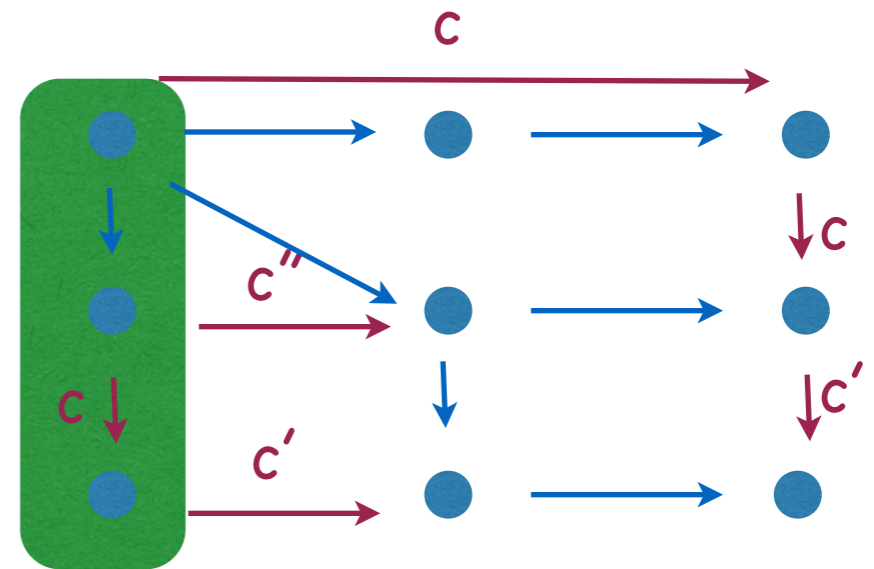
The relaxation framework

- Start from $LTS(S)$
- Add transitions with transition costs
- Fix a path cost function



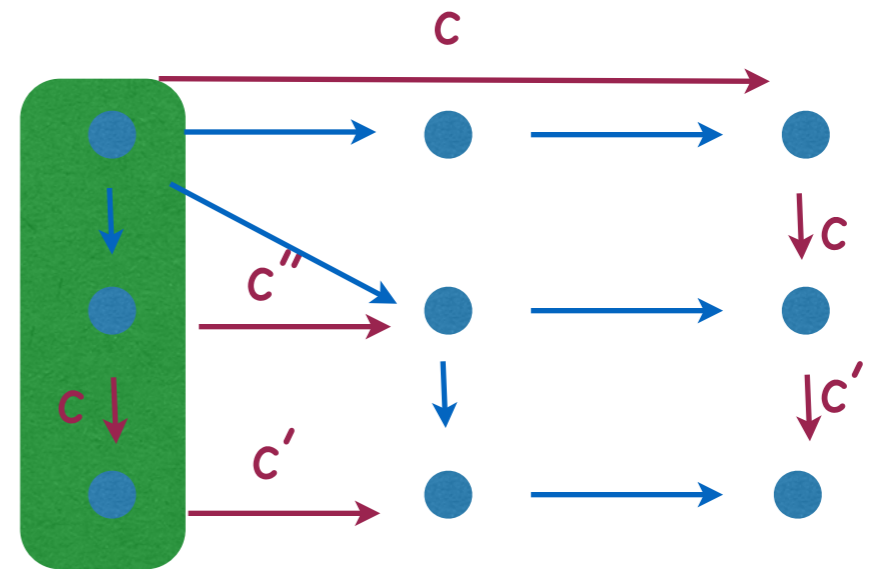
The relaxation framework

- Start from $LTS(S)$
- Add transitions with transition costs
- Fix a path cost function



The relaxation framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function



distance = minimal cost on all paths labelled by the sequence

Generic out-of-order

$$\text{segment_cost}(q \xrightarrow{m} q') = |\mathbf{v}|$$

transition cost

Where \mathbf{v} is a sequence of minimal length s.t.

removing \mathbf{v} enables a transition

or

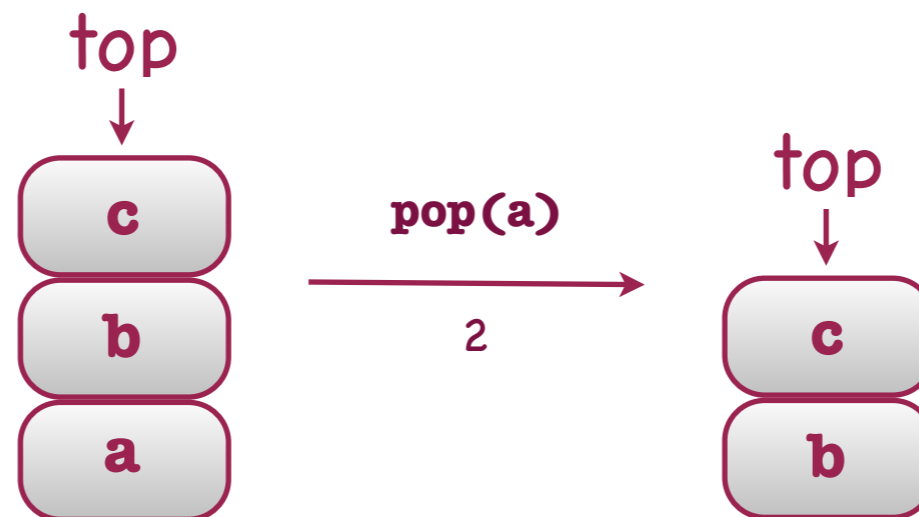
inserting \mathbf{v} enables a transition

goes with different path costs

Out-of-order stack

Sequence of push's with no matching pop

- Canonical representative of a state
- Add incorrect transitions with **segment-costs**



- Possible path cost functions **max**, **sum**,...

also more advanced