Demo of k-tails

The input for the tool is in resources/demo_ktails.txt The hash sign # is used to denote the start of a comment. Running the tool involves executing the following command from a command-line (expecting Java JDK and Ant to be installed from the statechum directory with build.xml):

```
ant demo_ktails
```

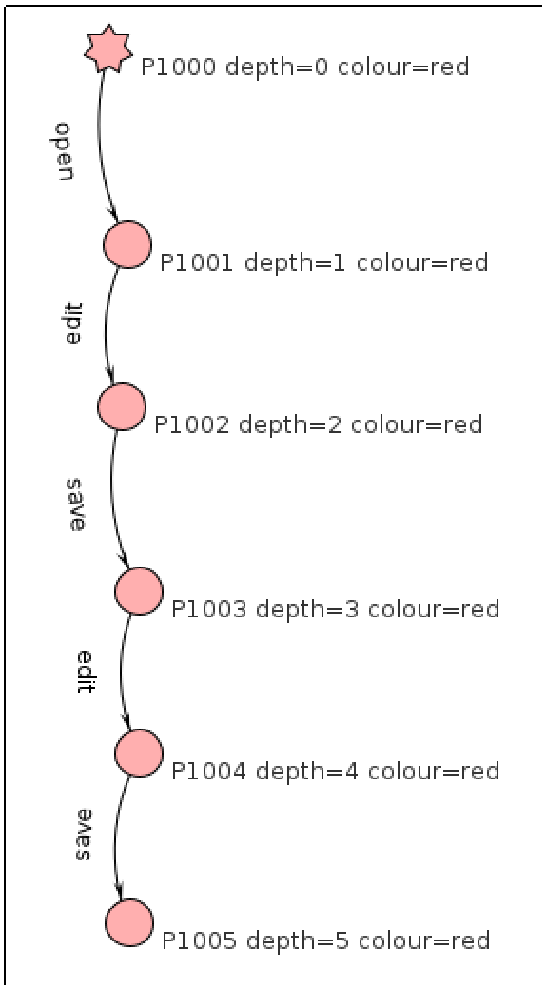
This pops up a window with a learnt automaton, to close Statechum press Esc (the tool will not terminate if you close the window with a cross). The star-state is the initial state in the automaton. States can be dragged with a mouse to re-arrange a graph.

Step 0

This is where a trace open-edit-save-edit-save is given to a k-tails learner and tail length is set to 3. This means that such a trace is turned into a very simple automaton and no merging can take place because there are no states with a suffix of length 3 that match. The longest matching suffix is edit-save which is length 2.

The input to the tool for step 0 is

```
passive
k 3
+ [[open, edit, save, edit, save]]
config learnerScoreMode KTAILS
config visualiseOutput true
```

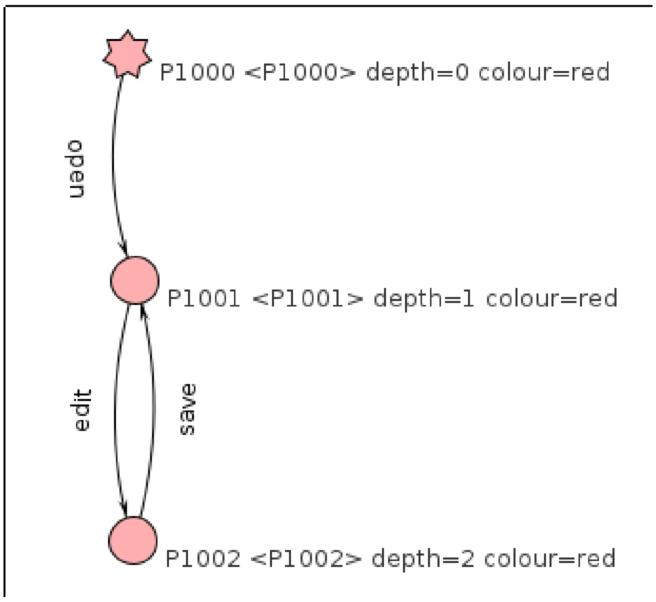


Step 1: merging with k=2

Here we change the line k 3 to read k 2 which means that any pair of states with a suffix of length 2 will be merged. This causes states P1001 and P1003 to be merged into P1001.

The input to the tool is

```
passive
k 2
+ [[open, edit, save, edit, save]]
config learnerScoreMode KTAILS
config visualiseOutput true
```

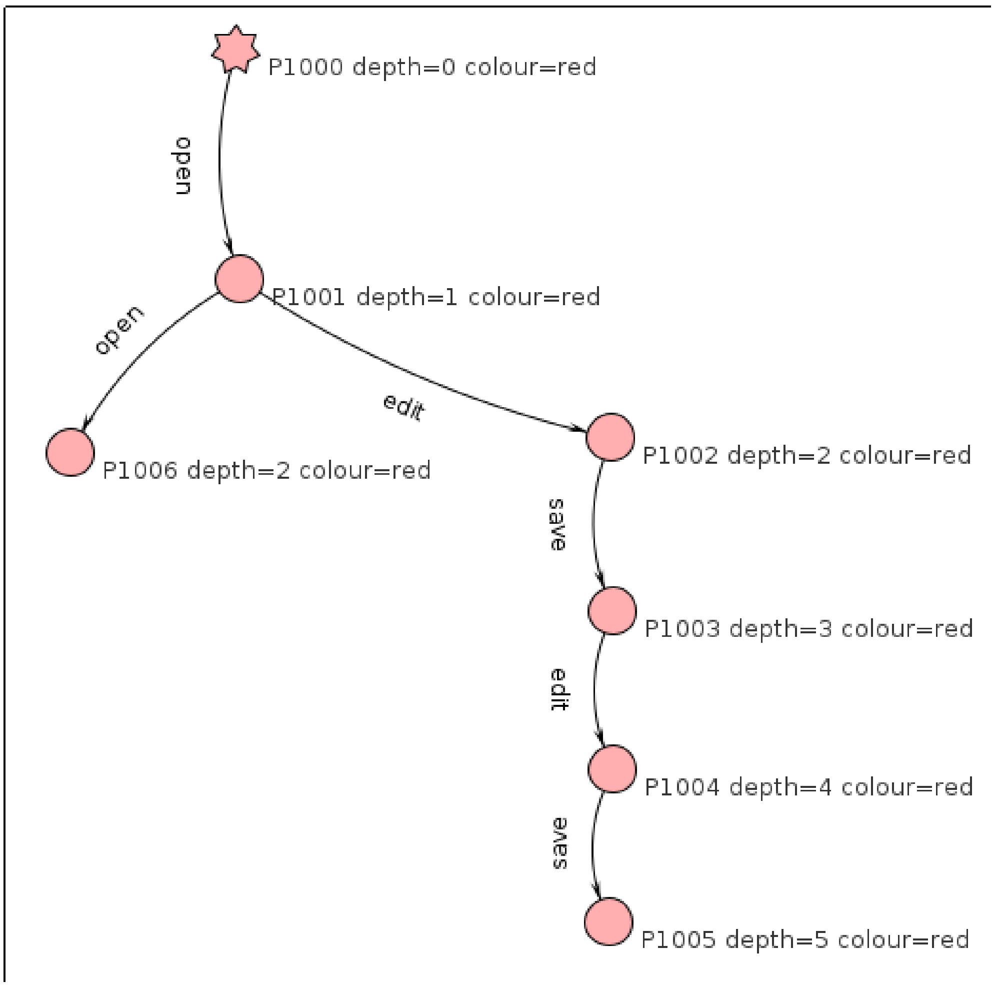

P1000 <P1000> depth=0 colour=red
open
P1001 <P1001> depth=1 colour=red
edit
save
P1002 <P1002> depth=2 colour=red

Step 2: adding another trace

Now we try to add another trace, open-open. This leads to a tree being constructed where the target states of 'open' in both traces are merged together and no additional mergers taking place.

Target states of open are being merged because this is a common prefix for these traces and we are building a deterministic automaton hence any common path from the initial state will lead to the same state.

No mergers occur because although there is a common suffix from P1001 and P1003, 'open' from P1001 is not matched in P1003.

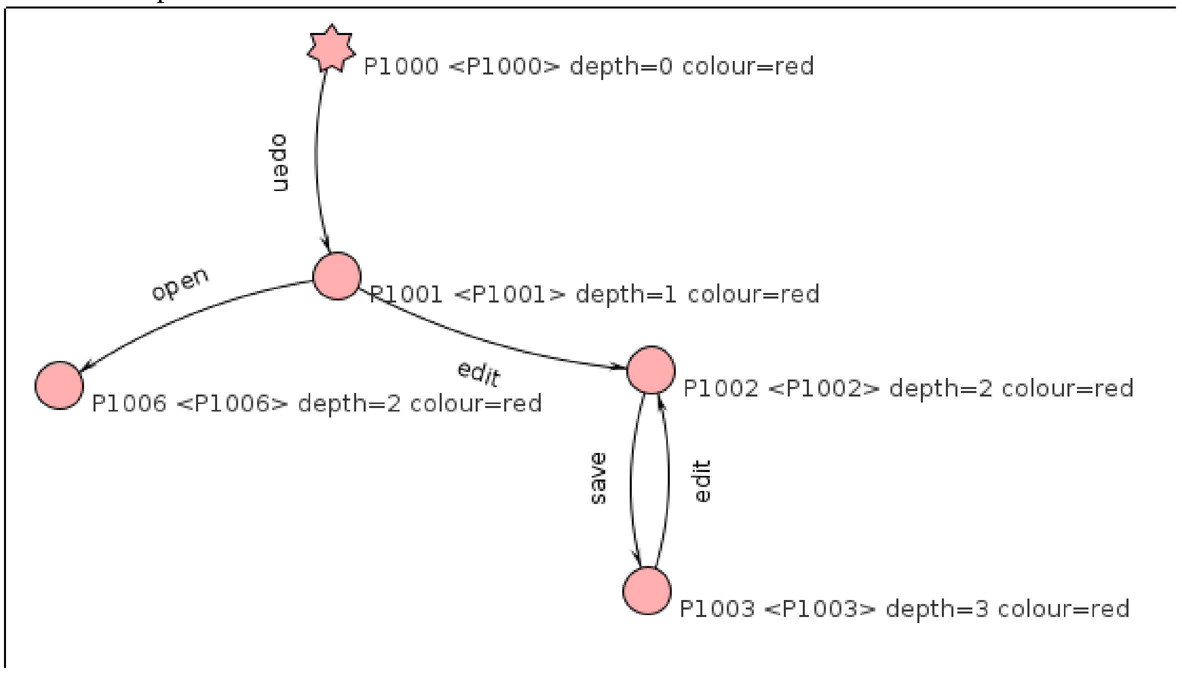


The input to the tool for step 2 is:

```
passive
k 2
+ [[open, edit, save, edit, save], [open,open]]
config learnerScoreMode KTAILS
config visualiseOutput true
```

Step 3: setting k=1

With k=1, any pairs of states with common suffixes of length 1 and more are merged. This means a merge between P1002 and P1004. P1001 still cannot be merged with any other state because P1003 does not have 'open' and P1000 does not have 'edit'.

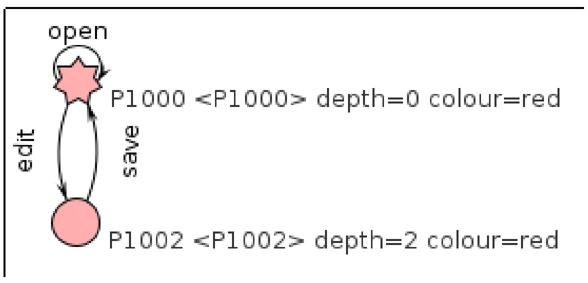

P1000 <P1000> depth=0 colour=red
open
P1001 <P1001> depth=1 colour=red
open
edit
P1006 <P1006> depth=2 colour=red
P1002 <P1002> depth=2 colour=red
save
edit
P1003 <P1003> depth=3 colour=red

The input to the tool for step 3 is:

```
passive
k 1
+ [[open, edit, save, edit, save], [open,open]]
config learnerScoreMode KTAILS
config visualiseOutput true
```

Step 4: setting merging algorithm to KTAILS_ANY

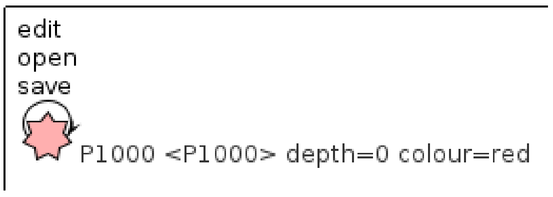
This setting is very close to how k-tails is usually defined: it permits merging for any pair of states where there is any suffix of length k rather than requiring a match for all suffixes of that length. This means that P1000 will be merged with P1001, P1003 and P1005. State P1002 will be merged with P1004 (see figure for step 2 depicting the graph with these states before the merge).


open
P1000 <P1000> depth=0 colour=red
edit
save
P1002 <P1002> depth=2 colour=red

The input to the tool for step 4 is:

```
passive
k 1
+ [[open, edit, save, edit, save], [open,open]]
config learnerScoreMode KTAILS_ANY
config visualiseOutput true
```

Step 5: setting k=0

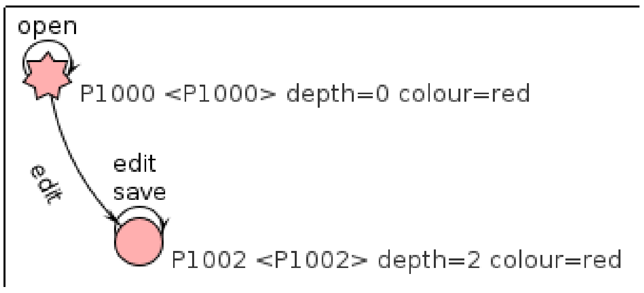


Now let's try to set k=0, this means that any pair of states can be merged, the result is a single-state automaton. Although this is not an interesting result by itself, this is what may happen if a learner makes a mistake and merges too many states.

The input to the tool for step 5 is:

```
passive
k 0
+ [[open, edit, save, edit, save], [open,open]]
config learnerScoreMode KTAILS_ANY
config visualiseOutput true
```

Step 6: adding negative traces


open
P1000 <P1000> depth=0 colour=red
edit
edit
save
P1002 <P1002> depth=2 colour=red

Where a learner makes a mistake, one could try to correct it by giving it counter-examples. These would include traces either permitted by the learnt automaton that should not be allowed or traces prohibited by the learnt automaton that should be possible. In this example, we give a negative trace open-save which means that it should not be possible to do a save immediately after open (this is actually debatable, but assume for the purpose of this demo that only modified files should be saved). The outcome is an automaton where save is no longer used on a transition from the initial state.

The input to the tool for step 6 is

```
passive
k 0
+ [[open, edit, save, edit, save], [open,open]]
- [[open,save]]
config learnerScoreMode KTAILS_ANY
config visualiseOutput true
```