The ferryman

The ferryman can cross the river with at most one passenger in his boat. There is a behavioural conflict between:

1. the goat and the cabbage; and

2. the goat and the wolf; if they are on the same river bank but the ferryman crosses the river or stays on the other bank.

Can the ferryman transport all goods to the other side, without any conflicts occurring?

This is a planning problem, but it can be solved by model checking. We describe a transition system in which the states represent which goods are at which side of the river. Then we ask if the goal state is reachable from the initial state: Is there a path from the initial state such that it has a state along it at which all the goods are on the other side, and during the transitions to that state the goods are never left in an unsafe, conflicting situation?

The location of each agent is modelled as a boolean variable: 0 denotes that the agent is on the initial bank, and 1 the destination bank. Thus, ferryman = 0 means that the ferryman is on the initial bank, ferryman = 1 that he is on the destination bank, and similarly for the variables goat, cabbage and wolf. The variable carry takes a value indicating whether the goat, cabbage, wolf or nothing is carried by the ferryman. The definition of next(carry) works as follows. It is non-deterministic, but the set from which a value is non-deterministically chosen is determined by the values of ferryman, goat etc., and always includes 0. If ferryman = goat (i.e., they are on the same side) then g is a member of the set from which next(carry) is chosen. The situation for cabbage and wolf is similar. Thus, if ferryman = goat = wolf = cabbage then that set is {g, w, 0}. The next value assigned to ferryman is non-deterministic: he can choose to cross or not to cross the river. But the next values of goat, cabbage and wolf are deterministic, since whether they are carried or not is determined by the ferryman's choice, represented by the non-deterministic assignment to carry; these values follow the same pattern. Note how the boolean guards refer to state bits at the next state. The SMV compiler does a dependency analysis and rejects circular dependencies on next values. (The dependency analysis is rather pessimistic: sometimes NuSMV complains of circularity even in situations when it could be resolved. The original CMU-SMV is more liberal in this respect.)

Syntax of CTL

The language of well-formed formulas for CTL is generated by the following grammar:

$$\begin{aligned}\phi ::= {} & \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid (\phi \Leftrightarrow \phi) \\ \mid \quad & \mathrm{AX}\,\phi \mid \mathrm{EX}\,\phi \mid \mathrm{AF}\,\phi \mid \mathrm{EF}\,\phi \mid \mathrm{AG}\,\phi \mid \mathrm{EG}\,\phi \mid \mathrm{A}\,[\phi\,\mathrm{U}\,\phi] \mid \mathrm{E}\,[\phi\,\mathrm{U}\,\phi]\end{aligned}$$

where p ranges over a set of atomic formulas. It is not necessary to use all connectives – for example, $\{\neg, \wedge, \mathrm{AX}, \mathrm{AU}, \mathrm{EU}\}$ comprises a complete set of connectives, and the others can be defined using them.

- $\mathbf{A}$ means 'along All paths' *(inevitably)*
- $\mathbf{E}$ means 'along at least (there Exists) one path' *(possibly)*

For example, the following is a well-formed CTL formula:

$$\mathrm{EF}\ (\mathrm{EG}\ p \Rightarrow \mathrm{AF}\ r)$$

The following is not a well-formed CTL formula:

$$\mathrm{EF}\ \big(r\ \mathrm{U}\ q\big)$$

The problem with this string is that $\mathbf{U}$ can occur only when paired with an $\mathbf{A}$ or an $\mathbf{E}$.

CTL uses atomic propositions as its building blocks to make statements about the states of a system. These propositions are then combined into formulas using logical operators and temporal operators.