CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Decision Analytics

Lecture 8: Strategy planning example

Strategy planning as SAT problem

- Example: **The wolf, the sheep, and the cabbage**

You find yourself on the shore of a river with a wolf, a sheep, and a cabbage. You can only take a maximum of one of these with you on the boat. Obviously you want to ferry all across, so you need to make multiple journeys back and forth. The problem is, if you leave the wolf and the sheep unattended, the wolf will eat the sheep. Also, if you leave the sheep and the cabbage alone on one shore, the sheep will have eaten the cabbage when you return.

<u>Your task:</u>
Come up with a plan how to ferry all three over the river, never leaving wolf and sheep or sheep and cabbage unattended.

The wolf, the sheep, and the cabbage

- The predicates are the Boolean sub-states the problem can be in at any given point in time

  - Wolf on this side of the river
  - Sheep on this side of the river
  - Cabbage on this side of the river
  - Ferryman on this side of the river

  - Wolf on the opposite side of the river
  - Sheep on the opposite side of the river
  - Cabbage on the opposite side of the river
  - Ferryman on the opposite side of the river

The wolf, the sheep, and the cabbage

- For each predicate and each time-step we create a Boolean variable

```
for t in range(maxT):
    WolfOnThisSide[t] = model.NewBoolVar("WolfOnThisSide"+str(t))
    SheepOnThisSide[t] = model.NewBoolVar("SheepOnThisSide"+str(t))
    CabbageOnThisSide[t] = model.NewBoolVar("CabbageOnThisSide"+str(t))
    FerrymanOnThisSide[t] = model.NewBoolVar("FerrymanOnThisSide"+str(t))
    WolfOnOppositeSide[t] = model.NewBoolVar("WolfOnThisSide"+str(t))
    SheepOnOppositeSide[t] = model.NewBoolVar("SheepOnThisSide"+str(t))
    CabbageOnOppositeSide[t] = model.NewBoolVar("CabbageOnThisSide"+str(t))
    FerrymanOnOppositeSide[t] = model.NewBoolVar("FerrymanOnOppositeSide"+str(t))
```

The wolf, the sheep, and the cabbage

- We define the **initial state** as
  - Wolf on this side of the river at first time-step
  - Sheep on this side of the river at first time-step
  - Cabbage on this side of the river at first time-step
  - Ferryman on this side of the river at first time-step

  - NOT (Wolf on the opposite side of the river at first time-step)
  - NOT (Sheep on the opposite side of the river at first time-step)
  - NOT (Cabbage on the opposite side of the river at first time-step)
  - NOT (Ferryman on the opposite side of the river at first time-step)

- The initial state also contains the negative statements of nothing being on the opposite side at the beginning

The wolf, the sheep, and the cabbage

- The initial state is a conjunction of all initial literals

$$l_{i_1 1} \wedge \cdots \wedge l_{i_{|I|} 1} \wedge \neg l_{j_1 1} \wedge \cdots \wedge \neg l_{j_{|J|} 1}$$

```
model.AddBoolAnd(
                [FerrymanOnThisSide[0],
                 WolfOnThisSide[0],
                 SheepOnThisSide[0],
                 CabbageOnThisSide[0] ] )
model.AddBoolAnd(
                [FerrymanOnOppositeSide[0].Not(),
                 WolfOnOppositeSide[0].Not(),
                 SheepOnOppositeSide[0].Not(),
                 CabbageOnOppositeSide[0].Not() ] )
```

The wolf, the sheep, and the cabbage

- The **goal state** for the problem is
  - Wolf on the opposite side of the river at last time-step
  - Sheep on the opposite side of the river at last time-step
  - Cabbage on the opposite side of the river at last time-step

```
model.AddBoolAnd(
              [WolfOnOppositeSide[maxT-1],
               SheepOnOppositeSide[maxT-1],
               CabbageOnOppositeSide[maxT-1]])
```

The wolf, the sheep, and the cabbage

- The **operators** that can alter the state of the problem in each time-step are
  - Move the wolf across
  - Move the sheep across
  - Move the cabbage across
  - Move the wolf back
  - Move the sheep back
  - Move the cabbage back
  - Move across without anything
  - Move back without anything

The wolf, the sheep, and the cabbage

- For each operator and each time-step we create another Boolean variable

```
for t in range(maxT-1):
        moveWolfAcross[t] = model.NewBoolVar("MoveWolfAcross"+str(t))
        moveSheepAcross[t] = model.NewBoolVar("MoveSheepAcross"+str(t))
        moveCabbageAcross[t] = model.NewBoolVar("MoveCabbageAcross"+str(t))
        moveWolfBack[t] = model.NewBoolVar("MoveWolfBack"+str(t))
        moveSheepBack[t] = model.NewBoolVar("MoveSheepBack"+str(t))
        moveCabbageBack[t] = model.NewBoolVar("MoveCabbageBack"+str(t))
        moveAcross[t] = model.NewBoolVar("MoveAcross"+str(t))
        moveBack[t] = model.NewBoolVar("MoveBack"+str(t))
```

The wolf, the sheep, and the cabbage

- Operator: Move the wolf/sheep/cabbage across
  - Pre-conditions:
    - wolf/sheep/cabbage on this side of the river
    - Ferryman on this side of the river
  - Post-conditions:
    - wolf/sheep/cabbage on opposite side of the river
    - Ferryman on opposite side of the river
    - NOT (wolf/sheep/cabbage on this side of the river)
    - NOT (Ferryman on this side of river)

The wolf, the sheep, and the cabbage

- Operator encoding

$$o_{kt} \Rightarrow (p_{1t} \land \cdots \land p_{ut} \land e_{1t} \land \cdots \land e_{vt})$$

- Example: Move wolf across the river at time-step t

```
model.AddBoolAnd([WolfOnThisSide[t],FerrymanOnThisSide[t],
                  WolfOnOppositeSide[t+1], FerrymanOnOppositeSide[t+1],
                  WolfOnThisSide[t+1].Not(), FerrymanOnThisSide[t+1].Not()
                  ]).OnlyEnforceIf(moveWolfAcross[t])
```

The wolf, the sheep, and the cabbage

- Operator: Move the wolf/sheep/cabbage back
  - Pre-conditions:
    - wolf/sheep/cabbage on opposite side of the river
    - Ferryman on opposite side of the river
  - Post-conditions:
    - wolf/sheep/cabbage on this side of the river
    - Ferryman on this side of the river
    - NOT (wolf/sheep/cabbage on opposite side of the river)
    - NOT (Ferryman on opposite side of river)

The wolf, the sheep, and the cabbage

- Operator encoding

$$o_{kt} \Rightarrow (p_{1t} \wedge \cdots \wedge p_{ut} \wedge e_{1t} \wedge \cdots \wedge e_{vt})$$

- Example: Move wolf back at time-step t

```
model.AddBoolAnd([WolfOnOppositeSide[t], FerrymanOnOppositeSide[t],
                  WolfOnThisSide[t+1], FerrymanOnThisSide[t+1],
                  WolfOnOppositeSide[t+1].Not(), FerrymanOnOppositeSide[t+1].Not()
                ]).OnlyEnforceIf(moveWolfBack[t])
```

The wolf, the sheep, and the cabbage

- Operator: Move Ferryman across
  - Pre-conditions:
    - Ferryman on this side of the river
  - Post-conditions:
    - Ferryman on opposite side of the river
    - NOT (Ferryman on this side of river)

- Operator encoding: $o_{kt} \Rightarrow (p_{1t} \wedge \cdots \wedge p_{ut} \wedge e_{1t} \wedge \cdots \wedge e_{vt})$

```
model.AddBoolAnd([FerrymanOnThisSide[t],
                  FerrymanOnOppositeSide[t+1], FerrymanOnThisSide[t+1].Not()
                  ]).OnlyEnforceIf(moveAcross[t])
```

The wolf, the sheep, and the cabbage

- Operator: Move Ferryman back
  - Pre-conditions:
    - Ferryman on opposite side of the river
  - Post-conditions:
    - Ferryman on this side of the river
    - NOT (Ferryman on opposite side of river)

- Operator encoding: $o_{kt} \Rightarrow (p_{1t} \wedge \cdots \wedge p_{ut} \wedge e_{1t} \wedge \cdots \wedge e_{vt})$

```
model.AddBoolAnd([FerrymanOnOppositeSide[t],
                  FerrymanOnThisSide[t+1], FerrymanOnOppositeSide[t+1].Not()
                  ]).OnlyEnforceIf(moveBack[t])
```

The wolf, the sheep, and the cabbage

- The **frame axioms** are the implicit assumptions that only operations can alter the state of the system. They are:
  - The wolf/sheep/cabbage is not on this side of the river, unless it has been there before or has been moved back
  - The wolf/sheep/cabbage is not on the opposite side of the river, unless it has been there before or has been moved across
  - The Ferryman is not on this side of the river, unless he has been there before or has moved back a wolf, a sheep or a cabbage, or if he moved back on his own
  - The Ferryman is not on the opposite side of the river, unless he has been there before or has moved across a wolf, a sheep or a cabbage, or if he moved over there on his own

The wolf, the sheep, and the cabbage

- Frame axiom: The wolf is not on this side of the river, unless it has been there before or has been moved back

$$(l_{nt} \vee \neg l_{n(t+1)}) \vee (o_{k_1 t} \vee \cdots \vee o_{k_{|K|} t})$$

```
for t in range(maxT-1):
        model.AddBoolOr([WolfOnThisSide[t+1].Not(),
                         WolfOnThisSide[t],
                         moveWolfBack[t]])
```

The wolf, the sheep, and the cabbage

- Frame axiom: The wolf is not on the opposite side of the river, unless it has been there before or has been moved across

$$(l_{nt} \lor \neg l_{n(t+1)}) \lor (o_{k_1 t} \lor \cdots \lor o_{k_{|K|} t})$$

```
for t in range(maxT-1):
    model.AddBoolOr([WolfOnOppositeSide[t+1].Not(),
                     WolfOnOppositeSide[t],
                     moveWolfAcross[t]])
```

The wolf, the sheep, and the cabbage

- Frame axiom: The Ferryman is not on this side of the river, unless he has been there before or has moved back a wolf, a sheep or a cabbage, or if he moved back on his own

$$(l_{nt} \vee \neg l_{n(t+1)}) \vee (o_{k_1 t} \vee \cdots \vee o_{k_{|K|} t})$$

```
for t in range(maxT-1):
    model.AddBoolOr([FerrymanOnThisSide[t+1].Not(),
                     FerrymanOnThisSide[t],
                     moveWolfBack[t],
                     moveSheepBack[t],
                     moveCabbageBack[t],
                     moveBack[t]])
```

The wolf, the sheep, and the cabbage

- Frame axiom: The Ferryman is not on the opposite side of the river, unless he has been there before or has moved across a wolf, a sheep or a cabbage, or if he moved over there on his own

$$(l_{nt} \vee \neg l_{n(t+1)}) \vee (o_{k_1 t} \vee \cdots \vee o_{k_{|K|} t})$$

```
for t in range(maxT-1):
    model.AddBoolOr([FerrymanOnOppositeSide[t+1].Not(),
                     FerrymanOnOppositeSide[t],
                     moveWolfAcross[t],
                     moveSheepAcross[t],
                     moveCabbageAcross[t],
                     moveAcross[t]])
```

The wolf, the sheep, and the cabbage

- The **complete exclusion axiom** states that no two operations can occur concurrently
    - We cannot simultaneously move the wolf and sheep across
    - We cannot simultaneously move the wolf and cabbage across
    - …
    - We cannot simultaneously move the wolf across and the sheep back
    - …
    - We cannot simultaneously move across the river and back

(there are a total of 8 operations and therefore $\frac{8!}{2!6!} = 28$ pairs of operations to consider for each time-step)

The wolf, the sheep, and the cabbage

- Example exclusion axiom: we cannot simultaneously move the wolf and sheep across

$$\neg o_{kt} \vee \neg o_{lt}$$

```
    for t in range(maxT-1):
        model.AddBoolOr([moveWolfAcross[t].Not(), moveSheepAcross[t].Not()])
```

(There need to be 28 more of these to cover all pairs of operations)

The wolf, the sheep, and the cabbage

- So that’s it, or is there anything left to add?

- Up until this point we created a satisfiability problem that calculates a viable plan to achieve the goal using the operations provided

- However, we haven’t specified any additional constraints, yet.
- In particular, nothing is preventing the plan from leaving the wolf and sheep or he sheep and cabbage on the same shore

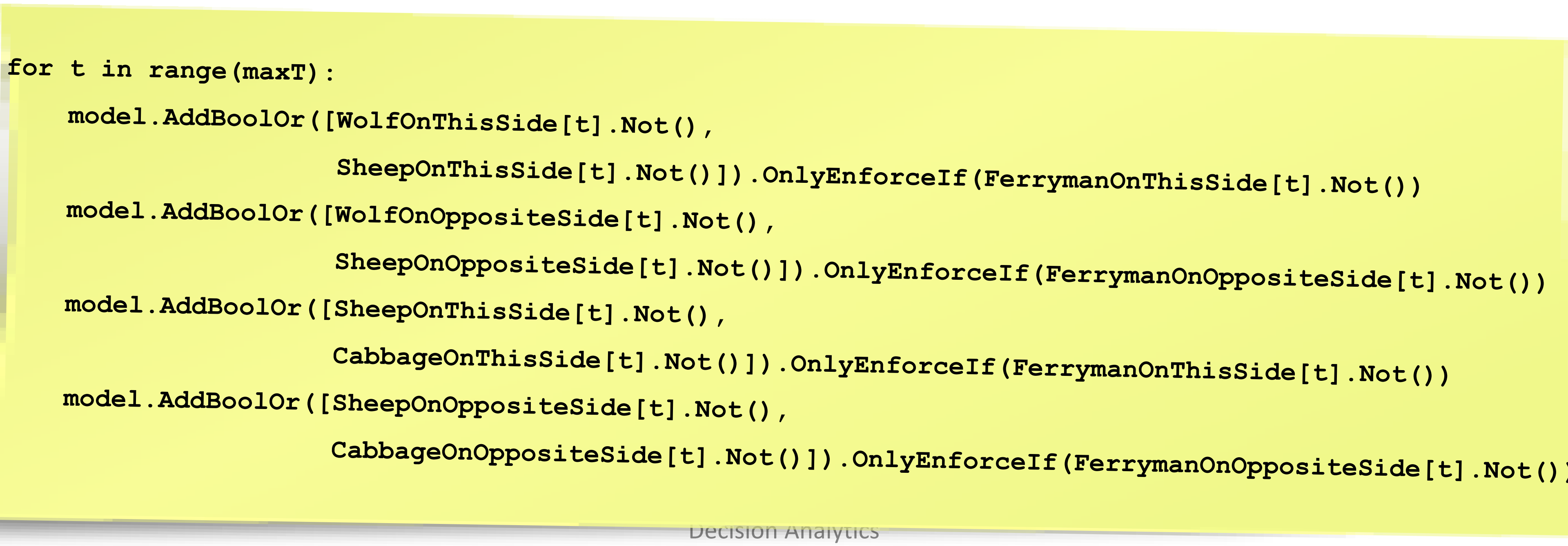
The wolf, the sheep, and the cabbage

- The additional constraints are:
  - The wolf and the sheep cannot be on this side of the river, unless the Ferryman is as well
  - The sheep and the cabbage cannot be on this side of the river, unless the Ferryman is as well
  - The wolf and the sheep cannot be on the opposite side of the river, unless the Ferryman is as well
  - The sheep and the cabbage cannot be on the opposite side of the river, unless the Ferryman is as well

The wolf, the sheep, and the cabbage

- To enforce the additional constraints we add for each time-step the following implication:
  - If the Ferryman is not on this side, then the wolf is not on this side or the sheep is not on this side (and similar for the other 3 constraints)

$$\neg l_{it} \Rightarrow \neg l_{jt} \vee \neg l_{kt}$$



```
for t in range(maxT):
    model.AddBoolOr([WolfOnThisSide[t].Not(),
                     SheepOnThisSide[t].Not()]).OnlyEnforceIf(FerrymanOnThisSide[t].Not())
    model.AddBoolOr([WolfOnOppositeSide[t].Not(),
                     SheepOnOppositeSide[t].Not()]).OnlyEnforceIf(FerrymanOnOppositeSide[t].Not())
    model.AddBoolOr([SheepOnThisSide[t].Not(),
                     CabbageOnThisSide[t].Not()]).OnlyEnforceIf(FerrymanOnThisSide[t].Not())
    model.AddBoolOr([SheepOnOppositeSide[t].Not(),
                     CabbageOnOppositeSide[t].Not()]).OnlyEnforceIf(FerrymanOnOppositeSide[t].Not()
```

The wolf, the sheep, and the cabbage

- When we solve this Boolean satisfiability problem, the resulting plan for ferrying everything over the river is encoded in the Boolean variables corresponding to the operations
- Therefore we can extract the result as follows:

```
for t in range(maxT-1):
        if solver.Value(moveWolfAcross[t]): print("move wolf across")
        if solver.Value(moveWolfBack[t]): print("move wolf back")
        if solver.Value(moveSheepAcross[t]): print("move sheep across")
        if solver.Value(moveSheepBack[t]): print("move sheep back")
        if solver.Value(moveCabbageAcross[t]): print("move cabbage across")
        if solver.Value(moveCabbageBack[t]): print("move cabbage back")
        if solver.Value(moveAcross[t]): print("move across")
        if solver.Value(moveBack[t]): print("move back")
```

The wolf, the sheep, and the cabbage

- The plan for ferrying all three over the river is
  - move the sheep across
  - move back
  - move the cabbage across
  - move the sheep back
  - move the wolf across
  - move back
  - move the sheep across

Thank you for your attention!