CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Decision Analytics

Lecture 5: Boolean Satisfiability and Logic Inference

Constraint Satisfaction Problem

- A **Constraint Satisfaction Problem** (CSP) is defined by
  - A tuple of n variables
    $$X =< x_1, \dots, x_n >$$
  - a corresponding tuple of domains
    $$D =< D_1, \dots, D_n >$$
    defining the potential values each variable can assume
    $$x_i \in D_i$$
  - and a tuple of t constraints
    $$C =< C_1, \dots, C_t >$$
    each being defined itself by a tuple
    $$\mathrm{C}_i =< R_{S_i}, S_i >$$
    - comprising the scope of the constraint $S_i \subset X$, being the subset of variables the constraint operates on
    - and the relation $R_{S_i} \subset D_{s_{i_1}} \times \cdots \times D_{s_{i_{|S_i|}}}$, being the set of valid variable assignments in the scope of the constraint

Boolean Satisfiability (SAT)

- A special case of CSP are Boolean satisfiability problems

- In this case all domains are Boolean, i.e. restricted to

$$\mathrm{D_i} = \{T, F\}$$

```
model = cp_model.CpModel()
model.NewBoolVar("x")
```

- And all constraints boil down to expressions in Boolean algebra

$$R_{S_i} = \{< x_{s_{i_1}}, \dots, x_{s_{i_{|S_i|}}} > \mid T[x_{s_{i_1}}, \dots, x_{s_{i_{|S_i|}}}] = True\}$$

- The solution to this CSP is an assignment to all Boolean variables such that all terms in the constraints evaluate to true

First-order logic

- First-order logic describes the world as
  - a set of **objects** (e.g. cars, houses)
  - with individual **properties** (e.g. red, blue)

- Amongst these objects various **relations** are defined that are known to hold (e.g. $colour[car, red]$, $\exists x: colour[x, blue]$)

- (sometimes also **functions** are defined, but they can be seen as special case of a relation, e.g. $colour\mathrm{Of}[\mathrm{car}]$=red)

- One goal of logic inference is to determine facts that are implied by the constraints (e.g. $colour[house, blue]$)

First-order logic

- First-order logic describes the state of the domain using **sentences** (e.g. $colour[car, green] \vee colour[car, blue]$)

- Each sentence is either an atomic **predicate** or it is a complex sentence being constructed from other less-complex sentences (see below)

- A **predicate** describe relations between **objects** and **attributes**, which are either true or false (e.g. $colour[car, green]$)

- (sometimes also **equality** is considered in atomic sentences, but it can also be constructed from predicates, see below)

First-order logic

- A complex sentence is constructed by
  - Logical connectives ($s_1 \wedge s_2$, $s_1 \vee s_2$, $s_1 \Rightarrow s_2$, $s_1 \Leftrightarrow s_2$) combining two less complex sentences s_1 and s_2
  - Negation of another sentence s (i.e. $\neg s$) or bracketing another sentence (i.e. (s))
  - By using **quantifiers** for either universal quantification (∀) or for existential quantification (see below)
- Quantification over objects and attributes defines *first-order* logic as opposed to predicate logic (*zero-order*) and quantification over relations (*second-order*)

First-order logic

- **Universal quantification** (denoted with ∀) allow us to make statements that have to hold for all sentences they refer to

- They use **variables** as placeholders for objects and attributes in the underlying predicates to make a statement that holds for every instantiation of that variable
(e.g. $\forall x\colon colour[x, red] \Rightarrow owner[x, John]$, which means that every red car is owned by John)

- A term with no variables, i.e. a term that only contains **constants**, is called a **ground term**

First-order logic

- **Existential quantification** (denoted with ∃) allow us to make statements that have to hold for <u>at least one</u> of the sentences they refer to

- Again, they use **variables** as placeholders for objects and attributes in the underlying predicates to make a statement that holds at least one instantiation of that variable
(e.g. $\exists x\!: colour[x, green]$, which means that somewhere there is a green object)

- The domain in this case is implicit in the predicates used

First-order logic

- As sentences are made of less complex sentences, quantifiers can be nested (e.g. $\forall x: \exists y: colour[x, y]$, meaning that every object has at least one colour)

- Similar to conjunction and disjunction in Boolean logic, the universal and existential quantification in first-order logic are related as follows
  - $\forall x: \neg P \equiv \neg \exists x: P$
    (if P does not hold for every x, then there is not one x for which P holds)
  - $\neg \forall x: P \equiv \exists x: \neg P$
    (if it is not the case that for every x P holds, then there is one x for which P does not hold)
  - $\forall x: P \equiv \neg \exists x: \neg P$
    (if x holds for every P, then there is no x for which P does not hold)
  - $\exists x: P \equiv \neg \forall x: \neg P$
    (if there is an x for which P holds, then it is not the case that for all x P does not hold)

First-order logic

- Sometime we use the notation $x = y$ and $x \neq y$ to indicate to objects are equal or unequal

- Inequality can be handled by noting that $x \neq y \equiv \neg(x = y)$

- Then, equality can be constructed as dedicated predicate over all objects and attributes asserting

$$
\begin{aligned}
&equal[car, car] \wedge \\
&equal[house, house] \wedge \\
&equal[red, red] \wedge \\
&equal[blue, blue] \wedge \\
&\ldots
\end{aligned}
$$

First-order logic as SAT problem

- First, we identify the **predicates** $P_1, \dots, P_N$ of the problem domain (e.g. $colour$, $owner$, etc.)

- For each **predicate** P_n we determine the **object** domain D_{O_n} and **attribute** domain D_{A_n} the predicate covers
(e.g. $colour\colon \{\text{car}, \text{house}\} \times \{red, green, blue\}$,
$owner\colon \{\text{car}, \text{house}\} \times \{Alan, Bob, Dave\}$, etc.)

- For each predicate $P_n\colon D_{o_n} \times D_{A_n}$, each element in the associated object domain $j \in D_{O_n}$ and each element in the associated attribute domain $k \in D_{O_n}$ we create a Boolean variables x_{nij} indicating weather the predicate holds or not

- Finally, for each **sentence** we add a constraint over the predicate variables the sentence refers to

First-order logic as SAT problem

- Conjunctive normal form:
  We have already seen that we can transform every zero-order Boolean term into an equivalent term being a conjunction of disjunctions of literals, i.e. look like the following example

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg y_1 \vee y_2) \wedge (z_1 \vee z_2 \vee z_3 \vee z_4)$$

```
model.AddBoolOr([x1,x2.Not(),x3])
model.AddBoolOr([y1.Not(),y2])
model.AddBoolOr([z1,z2,z3,z4])
```

First-order logic as SAT problem

- A universal quantification over a predicate

$$\forall i: P_n[i, j]$$

- translates into a conjunction of the corresponding variables

$$x_{n1j} \land \cdots \land x_{n|D_{O_n}|j}$$

- Similar the existential quantification over a predicate

$$\exists i: P_n[i, j]$$

- translates into a disjunction of the corresponding variables

$$x_{n1j} \lor \cdots \lor x_{n|D_{O_n}|j}$$

- If the quantification is over an attribute, then the conjunction or disjunction is over the variables corresponding to the second index

The Zebra puzzle

- There are five houses.
- The Englishman lives in the red house.
- The Spaniard owns the dog.
- Coffee is drunk in the green house.
- The Ukrainian drinks tea.
- The green house is immediately to the right of the ivory house.
- The Old Gold smoker owns snails.
- Kools are smoked in the yellow house.
- Milk is drunk in the middle house.
- The Norwegian lives in the first house.
- The man who smokes Chesterfields lives in the house next to the man with the fox.
- Kools are smoked in the house next to the house where the horse is kept.
- The Lucky Strike smoker drinks orange juice.
- The Japanese smokes Parliaments.
- The Norwegian lives next to the blue house.
- **Now, who drinks water? Who owns the zebra?**

The Zebra puzzle

- First, we identify the predicates of the problem domain
  - $colour$
  - $nationality$
  - pet
  - $drink$
  - $cigarette$

- All of these operate on the same object domain (houses)
- The first clue reads “There are five houses.”, therefore
  - $D_O = \{House\ \#1, House\ \#2, House\ \#3, House\ \#4, House\ \#5\}$

```
houses = ["House #1", "House #2", "House #3", "House #4", "House #5"]
```

The Zebra puzzle

- The attribute domain for $colour$ is
$$D_A = \{red, green, ivory, yellow, blue\}$$

```
colours = ["red", "green", "ivory", "yellow", "blue" ]
```

- We add the 5x5 Boolean variables corresponding to all combinations of objects and attributes

```
house_colour = {}
for house in houses:
    variables = {}
    for colour in colours:
        variables[colour] = model.NewBoolVar(house+colour)
    house_colour[house] = variables
```

The Zebra puzzle

- The attribute domain for $nationality$ is
  $$D_A = \{English, Spanish, Ukranian, Norwegian, Japanese\}$$

```
nationalities = ["English",
                 "Spanish",
                 "Ukrainian",
                 "Norwegian",
                 "Japanese"]
```

- Again, we add the 5x5 Boolean variables corresponding to all combinations of objects and attributes

```
house_nationality = {}
for house in houses:
    variables = {}
    for nationality in nationalities:
        variables[nationality] = model.NewBoolVar(house+nationality)
    house_nationality[house] = variables
```

The Zebra puzzle

- The attribute domain for pet is
$$D_A = \{dog, snails, fox, horse, zebra\}$$

```
pets = ["dog", "snails", "fox", "horse", "zebra"]
```

- We add the 5x5 Boolean variables corresponding to all combinations of objects and attributes

```
house_pet = {}
for house in houses:
    variables = {}
    for pet in pets:
        variables[pet] = model.NewBoolVar(house+pet)
    house_pet[house] = variables
```

The Zebra puzzle

- The attribute domain for $drink$ is
$$D_A = \{coffee, tea, milk, juice, water\}$$

```
drinks = ["coffee", "tea", "milk", "juice", "water"]
```

- We add the 5x5 Boolean variables corresponding to all combinations of objects and attributes

```
house_drink = {}
for house in houses:
    variables = {}
    for drink in drinks:
        variables[drink] = model.NewBoolVar(house+drink)
    house_drink[house] = variables
```

The Zebra puzzle

- The attribute domain for $cigarette$ is $D_A = \{OldGold, Chesterfields, Kools, LuckeyStrike, Parliaments\}$

```
cigarettes = ["Old Gold",
              "Chesterfields",
              "Kools",
              "Lucky Strike",
              "Parliaments"]
```

- Again, we add the 5x5 Boolean variables corresponding to all combinations of objects and attributes

```
house_cigarette = {}
for house in houses:
    variables = {}
    for cigarette in cigarettes:
        variables[cigarette] = model.NewBoolVar(house+cigarette)
    house_cigarette[house] = variables
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- "The Englishman lives in the red house."

$$\forall x\colon nationality(x, English) \Rightarrow colour(x, red)$$

- We translate this into a conjunction over all houses

$$nationality(House\ \#1, English) \Rightarrow colour(House\ \#1, red)$$
$$nationality(House\ \#2, English) \Rightarrow colour(House\ \#2, red)$$
$$nationality(House\ \#3, English) \Rightarrow colour(House\ \#3, red)$$
$$nationality(House\ \#4, English) \Rightarrow colour(House\ \#4, red)$$
$$nationality(House\ \#5, English) \Rightarrow colour(House\ \#5, red)$$

```
for house in houses:
    model.AddBoolAnd([house_colour[house]["red"]]).OnlyEnforceIf(house_nationality[house]["English"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- "The Spaniard owns the dog."

$$\forall x\colon nationality(x, Spanish) \Rightarrow pet(x, dog)$$

- We translate this into a conjunction over all houses

$$\begin{aligned}
&nationality(House\ \#1, Spanish) \Rightarrow pet(House\ \#1, dog)\\
&nationality(House\ \#2, Spanish) \Rightarrow pet(House\ \#2, dog)\\
&nationality(House\ \#3, Spanish) \Rightarrow pet(House\ \#3, dog)\\
&nationality(House\ \#4, Spanish) \Rightarrow pet(House\ \#4, dog)\\
&nationality(House\ \#5, Spanish) \Rightarrow pet(House\ \#5, dog)
\end{aligned}$$

```
for house in houses:
    model.AddBoolAnd([house_pet[house]["dog"]]).OnlyEnforceIf(house_nationality[house]["Spanish"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- “Coffee is drunk in the green house.”

$$\forall x: colour(x, green) \Rightarrow drink(x, coffee)$$

- We translate this into a conjunction over all houses

$$colour(House\ \#1, green) \Rightarrow drink(House\ \#1, coffee)$$
$$colour(House\ \#2, green) \Rightarrow drink(House\ \#2, coffee)$$
$$colour(House\ \#3, green) \Rightarrow drink(House\ \#3, coffee)$$
$$colour(House\ \#4, green) \Rightarrow drink(House\ \#4, coffee)$$
$$colour(House\ \#5, green) \Rightarrow drink(House\ \#5, coffee)$$

```
for house in houses:
    model.AddBoolAnd([house_drink[house]["coffee"]]).OnlyEnforceIf(house_colour[house]["green"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- "The Ukrainian drinks tea."

$$\forall x: nationality(x, Ukrainian) \Rightarrow drink(x, tea)$$

- We translate this into a conjunction over all houses

$$nationality(House\ \#1, Ukrainian) \Rightarrow drink(House\ \#1, tea)$$
$$nationality(House\ \#2, Ukrainian) \Rightarrow drink(House\ \#2, tea)$$
$$nationality(House\ \#3, Ukrainian) \Rightarrow drink(House\ \#3, tea)$$
$$nationality(House\ \#4, Ukrainian) \Rightarrow drink(House\ \#4, tea)$$
$$nationality(House\ \#5, Ukrainian) \Rightarrow drink(House\ \#5, tea)$$

```
for house in houses:
    model.AddBoolAnd([house_drink[house]["tea"]]).OnlyEnforceIf(house_nationality[house]["Ukrainian"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- "The green house is immediately to the right of the ivory house."

$$colour(House\ \#1, ivory) \Rightarrow colour(House\ \#2, green)$$
$$colour(House\ \#2, ivory) \Rightarrow colour(House\ \#3, green)$$
$$colour(House\ \#3, ivory) \Rightarrow colour(House\ \#4, green)$$
$$colour(House\ \#5, ivory) \Rightarrow colour(House\ \#5, green)$$
$$\neg colour(House\ \#5, ivory)$$

```
for i in range(4) in houses:
    model.AddBoolAnd([house_colour[houses[i+1]]["green"]]).OnlyEnforceIf(house_colour[houses[i]]["ivory"])
model.AddBoolAnd([house_colour[houses[4]]["ivory"].Not()])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- “The Old Gold smoker owns snails.”

$$\forall x\colon cigarette(x, Old\ Gold) \Rightarrow pet(x, snails)$$

- We translate this into a conjunction over all houses

$$cigarette(House\ \#1, Old\ Gold) \Rightarrow pet(House\ \#1, snails)$$
$$cigarette(House\ \#2, Old\ Gold) \Rightarrow pet(House\ \#2, snails)$$
$$cigarette(House\ \#3, Old\ Gold) \Rightarrow pet(House\ \#3, snails)$$
$$cigarette(House\ \#4, Old\ Gold) \Rightarrow pet(House\ \#4, snails)$$
$$cigarette(House\ \#5, Old\ Gold) \Rightarrow pet(House\ \#5, snails)$$

```
for house in houses:
    model.AddBoolAnd([house_pet[house]["snails"]]).OnlyEnforceIf(house_cigarette[house]["Old Gold"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- “Kools are smoked in the yellow house.”

$$\forall x : cigarette(x, Kools) \Rightarrow colour(x, yellow)$$

- We translate this into a conjunction over all houses

$$cigarette(House\ \#1, Kools) \Rightarrow colour(House\ \#1, yellow)$$
$$cigarette(House\ \#1, Kools) \Rightarrow colour(House\ \#1, yellow)$$
$$cigarette(House\ \#1, Kools) \Rightarrow colour(House\ \#1, yellow)$$
$$cigarette(House\ \#1, Kools) \Rightarrow colour(House\ \#1, yellow)$$
$$cigarette(House\ \#1, Kools) \Rightarrow colour(House\ \#1, yellow)$$

```
for house in houses:
    model.AddBoolAnd([house_colour[house]["yellow"]]).OnlyEnforceIf(house_cigarette[house]["Kools"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- “Milk is drunk in the middle house.”

$$drink(House\ \#3, milk)$$

```
model.AddBoolAnd([house_drink["House #3"]["milk"]])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- "The Norwegian lives in the first house."

$$nationality(House\ \#1, Norwegian)$$

```
model.AddBoolAnd([house_nationality["House #1"]["Norwegian"]])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- “The man who smokes Chesterfields lives in the house next to the man with the fox.”

$$cigarette[House\ \#1, \text{Chesterfields}] \Rightarrow pet(House\ \#2, fox)$$
$$cigarette[House\ \#2, \text{Chesterfields}] \Rightarrow pet(House\ \#1, \text{fox}) \vee pet(House\ \#3, fox)$$
$$cigarette[House\ \#3, \text{Chesterfields}] \Rightarrow pet(House\ \#2, \text{fox}) \vee pet(House\ \#4, fox)$$
$$cigarette[House\ \#4, \text{Chesterfields}] \Rightarrow pet(House\ \#3, \text{fox}) \vee pet(House\ \#5\ fox)$$
$$cigarette[House\ \#5, \text{Chesterfields}] \Rightarrow pet(House\ \#4, fox)$$

```
for i in range(1,4):
    model.AddBoolOr([
            house_pet[houses[i+1]]["fox"],
            house_pet[houses[i-1]]["fox"]]).OnlyEnforceIf(house_cigarette[houses[i]]["Chesterfields"])
model.AddBoolOr([house_pet["House #2"]["fox"]]).OnlyEnforceIf(house_cigarette["House #1"]["Chesterfields"])
model.AddBoolOr([house_pet["House #4"]["fox"]]).OnlyEnforceIf(house_cigarette["House #5"]["Chesterfields"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- “Kools are smoked in the house next to the house where the horse is kept.”

$$cigarette[House\ \#1, \text{Kools}] \Rightarrow pet(House\ \#2, horse)$$
$$cigarette[House\ \#2, \text{Kools}] \Rightarrow pet(House\ \#1, \text{horse}) \vee pet(House\ \#3, horse)$$
$$cigarette[House\ \#3, \text{Kools}] \Rightarrow pet(House\ \#2, \text{horse}) \vee pet(House\ \#4, horse)$$
$$cigarette[House\ \#4, \text{Kools}] \Rightarrow pet(House\ \#3, \text{horse}) \vee pet(House\ \#5\ horse)$$
$$cigarette[House\ \#5, \text{Kools}] \Rightarrow pet(House\ \#4, horse)$$

```
for i in range(1,4):
    model.AddBoolOr([
            house_pet[houses[i+1]]["horse"],
            house_pet[houses[i-1]]["horse"]]).OnlyEnforceIf(house_cigarette[houses[i]]["Kools"])
model.AddBoolOr([house_pet["House #2"]["horse"]]).OnlyEnforceIf(house_cigarette["House #1"]["Kools"])
model.AddBoolOr([house_pet["House #4"]["horse"]]).OnlyEnforceIf(house_cigarette["House #5"]["Kools"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- "The Lucky Strike smoker drinks orange juice."

$$\forall x: cigarette(x, LuckyStrike) \Rightarrow drink(x, juice)$$

- We translate this into a conjunction over all houses

$$cigarette(House\ \#1, LuckyStrike) \Rightarrow drink(House\ \#1, juice)$$
$$cigarette(House\ \#1, LuckyStrike) \Rightarrow drink(House\ \#1, juice)$$
$$cigarette(House\ \#1, LuckyStrike) \Rightarrow drink(House\ \#1, juice)$$
$$cigarette(House\ \#1, LuckyStrike) \Rightarrow drink(House\ \#1, juice)$$
$$cigarette(House\ \#1, LuckyStrike) \Rightarrow drink(House\ \#1, juice)$$

```
for house in houses:
    model.AddBoolAnd([house_drink[house]["juice"]]).OnlyEnforceIf(house_cigarette[house]["Lucky Strike"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- "The Japanese smokes Parliaments."

$$\forall x: nationality(x, Japanese) \Rightarrow cigarette(x, Parliaments)$$

- We translate this into a conjunction over all houses

$$nationality(House\ \#1, Japanese) \Rightarrow cigarette(House\ \#1, Parliaments)$$
$$nationality(House\ \#2, Japanese) \Rightarrow cigarette(House\ \#2, Parliaments)$$
$$nationality(House\ \#3, Japanese) \Rightarrow cigarette(House\ \#3, Parliaments)$$
$$nationality(House\ \#4, Japanese) \Rightarrow cigarette(House\ \#4, Parliaments)$$
$$nationality(House\ \#5, Japanese) \Rightarrow cigarette(House\ \#5, Parliaments)$$

```
for house in houses:
    model.AddBoolAnd([house_cigarette[house]["Parliaments"]]).OnlyEnforceIf(house_nationality[house]["Japanese"])
```

The Zebra puzzle

- Now we go sentence by sentence and re-formulate in first-order logic
- “The Norwegian lives next to the blue house.

$$colour[House\ \#1, \text{blue}] \Rightarrow nationality(House\ \#2, Norwegian)$$
$$colour[House\ \#2, blue] \Rightarrow nationality(House\ \#1, \text{Norwegian}) \vee nationality(House\ \#3, Norwegian)$$
$$colour[House\ \#3, blue] \Rightarrow nationality(House\ \#2, \text{Norwegian}) \vee nationality(House\ \#4\ Norwegian)$$
$$colour[House\ \#4, blue] \Rightarrow nationality(House\ \#3, \text{Norwegian}) \vee nationality(House\ \#4, Norwegian)$$
$$colour[House\ \#5, \text{blue}] \Rightarrow nationality(House\ \#4, Norwegian)$$

```
for i in range(1,4):
    model.AddBoolOr([
                 house_nationality[houses[i+1]]["Norwegian"],
                 house_nationality[houses[i-1]]["Norwegian"]]).OnlyEnforceIf(house_colour[houses[i]]["blue"]
model.AddBoolOr([house_nationality[houses[1]]["Norwegian"]]).OnlyEnforceIf(house_colour[houses[0]]["blue"])
model.AddBoolOr([house_nationality[houses[3]]["Norwegian"]]).OnlyEnforceIf(house_colour[houses[4]]["blue"])
```

The Zebra puzzle

- That is all sentences of the zebra puzzle encoded, but is this all?

- There are some **implicit assumptions** that are not explicitly stated in the text
  - Every house has a colour/nationality/pet/drink/cigarette
  - Every house has no more than one colour/nationality/pet/drink/cigarette
  - All houses have a different colour/nationality/pet/drink/cigarette

The Zebra puzzle

- Every house has a colour

$$\forall x \exists y: colour(x, y)$$

- That translates into

$$\begin{aligned}&(colour(House\ \#1, red) \vee colour(House\ \#1, green) \vee colour(House\ \#1, ivory) \vee colour(House\ \#1, yellow) \vee colour(House\ \#1, blue))\\ &\wedge (colour(House\ \#2, red) \vee colour(House\ \#2, green) \vee colour(House\ \#2, ivory) \vee colour(House\ \#2, yellow) \vee colour(House\ \#2, blue))\\ &\wedge (colour(House\ \#3, red) \vee colour(House\ \#3, green) \vee colour(House\ \#3, ivory) \vee colour(House\ \#3, yellow) \vee colour(House\ \#3, blue))\\ &\wedge (colour(House\ \#4, red) \vee colour(House\ \#4, green) \vee colour(House\ \#4, ivory) \vee colour(House\ \#4, yellow) \vee colour(House\ \#4, blue))\\ &\wedge (colour(House\ \#5, red) \vee colour(House\ \#5, green) \vee colour(House\ \#5, ivory) \vee colour(House\ \#5, yellow) \vee colour(House\ \#5, blue))\end{aligned}$$

```
for house in houses:
    variables = []
    for colour in colours:
        variables.append(house_colour[house][colour])
    model.AddBoolOr(variables)
```

The Zebra puzzle

- Every house has an occupant

$$\forall x \exists y : nationality(x, y)$$

```
for house in houses:
    variables = []
    for nationality in nationalities:
        variables.append(house_nationality[house][nationality])
    model.AddBoolOr(variables)
```

The Zebra puzzle

- Every house has a pet

$$\forall x \exists y : pet(x, y)$$

```
for house in houses:
    variables = []
    for pet in pets:
        variables.append(house_pet[house][pet])
    model.AddBoolOr(variables)
```

The Zebra puzzle

- Every house has a drink

$$\forall x \exists y: drink(x, y)$$

```
for house in houses:
    variables = []
    for drink in drinks:
        variables.append(house_drink[house][drink])
    model.AddBoolOr(variables)
```

The Zebra puzzle

- Every house has a cigarette

$$\forall x \exists y: cigarette(x, y)$$

```
for house in houses:
    variables = []
    for cigarette in cigarettes:
        variables.append(house_cigarette[house][cigarette])
    model.AddBoolOr(variables)
```

The Zebra puzzle

- Every house has no more than one colour

$$\forall x \forall y \forall z: y \neq z \Rightarrow \neg(colour(x,y) \wedge colour(x,z))$$

This is equivalent to

$$\forall x \forall y \forall z: y \neq z \Rightarrow (\neg colour(x,y) \vee \neg colour(x,z))$$

```
for house in houses:
    for i in range(5):
        for j in range(i+1,5):
            model.AddBoolOr([
                house_colour[house][colours[i]].Not(),
                house_colour[house][colours[j]].Not()])
```

The Zebra puzzle

- Every house has no more than one occupant

$$\forall x \forall y \forall z : y \neq z \Rightarrow \neg(nationality(x, y) \wedge nationality(x, z))$$

This is equivalent to

$$\forall x \forall y \forall z : y \neq z \Rightarrow \ (\neg nationality(x, y) \vee \neg nationality(x, z))$$

```
for house in houses:
    for i in range(5):
        for j in range(i+1,5):
            model.AddBoolOr([
                house_nationality[house][nationalities[i]].Not(),
                 house_nationality[house][nationalities[j]].Not()])
```

The Zebra puzzle

- Every house has no more than one pet

$$\forall x \forall y \forall z: y \neq z \Rightarrow \neg(pet(x,y) \wedge pet(x,z))$$

This is equivalent to

$$\forall x \forall y \forall z: y \neq z \Rightarrow (\neg pet(x,y) \vee \neg pet(x,z))$$

```
for house in houses:
    for i in range(5):
        for j in range(i+1,5):
            model.AddBoolOr([
                house_pet[house][pets[i]].Not(),
                house_pet[house][pets[j]].Not()])
```

The Zebra puzzle

- Every house has no more than one drink

$$\forall x \forall y \forall z: y \neq z \Rightarrow \neg(drink(x,y) \land drink(x,z))$$

This is equivalent to

$$\forall x \forall y \forall z: y \neq z \Rightarrow (\neg drink(x,y) \lor \neg drink(x,z))$$

```
for house in houses:
    for i in range(5):
        for j in range(i+1,5):
            model.AddBoolOr([
                house_drink[house][drinks[i]].Not(),
                house_drink[house][drinks[j]].Not()])
```

The Zebra puzzle

- Every house has no more than one cigarette

$$\forall x \forall y \forall z: y \neq z \Rightarrow \neg(cigarette(x,y) \land cigarette(x,z))$$

This is equivalent to

$$\forall x \forall y \forall z: y \neq z \Rightarrow \ (\neg cigarette(x,y) \lor \neg cigarette(x,z))$$

```
for house in houses:
    for i in range(5):
        for j in range(i+1,5):
            model.AddBoolOr([
                house_cigarette[house][cigarettes[i]].Not(),
                house_cigarette[house][cigarettes[j]].Not()])
```

The Zebra puzzle

- Every house has a different colour

$$\forall x \forall y \forall z: x \neq y \Rightarrow \neg(colour(x,z) \wedge colour(y,z))$$

This is equivalent to

$$\forall x \forall y \forall z: x \neq y \Rightarrow (\neg colour(x,z) \vee \neg colour(y,z))$$

```
for i in range(5):
    for j in range(i+1,5):
        for k in range(5):
            model.AddBoolOr([
                    house_colour[houses[i]][colours[k]].Not(),
                    house_colour[houses[j]][colours[k]].Not()])
```

The Zebra puzzle

- Every house has a different occupant

$$\forall x \forall y \forall z: x \neq y \Rightarrow \neg(nationality(x,z) \wedge nationality(y,z))$$

This is equivalent to

$$\forall x \forall y \forall z: x \neq y \Rightarrow (\neg nationality(x,z) \vee \neg nationality(y,z))$$

```
for i in range(5):
    for j in range(i+1,5):
        for k in range(5):
            model.AddBoolOr([
                house_nationality[houses[i]][nationalities[k]].Not(),
                house_nationality[houses[j]][nationalities[k]].Not()])
```

The Zebra puzzle

- Every house has a different pet

$$\forall x \forall y \forall z: x \neq y \Rightarrow \neg\left(pet(x,z) \wedge pet(y,z)\right)$$

This is equivalent to

$$\forall x \forall y \forall z: x \neq y \Rightarrow \left(\neg pet(x,z) \vee \neg pet(y,z)\right)$$

```
for i in range(5):
    for j in range(i+1,5):
        for k in range(5):
            model.AddBoolOr([
                house_pet[houses[i]][pets[k]].Not(),
                house_pet[houses[j]][pets[k]].Not()])
```

The Zebra puzzle

- Every house has a different drink

$$\forall x \forall y \forall z: x \neq y \Rightarrow \neg(drink(x,z) \wedge drink(y,z))$$

This is equivalent to

$$\forall x \forall y \forall z: x \neq y \Rightarrow (\neg drink(x,z) \vee \neg drink(y,z))$$

```
for i in range(5):
    for j in range(i+1,5):
        for k in range(5):
            model.AddBoolOr([
                house_drink[houses[i]][drinks[k]].Not(),
                house_drink[houses[j]][drinks[k]].Not()])
```

The Zebra puzzle

- Every house has a different cigarette

$$\forall x \forall y \forall z: x \neq y \Rightarrow \neg(cigarette(x,z) \wedge cigarette(y,z))$$

This is equivalent to

$$\forall x \forall y \forall z: x \neq y \Rightarrow (\neg cigarette(x,z) \vee \neg cigarette(y,z))$$

```
for i in range(5):
    for j in range(i+1,5):
        for k in range(5):
            model.AddBoolOr([
                    house_cigarette[houses[i]][cigarettes[k]].Not(),
                    house_cigarette[houses[j]][cigarettes[k]].Not()])
```

The Zebra puzzle

- That is all constraints, both explicit and implicit, encoded

- Who drinks water? The Norwegian.
- Who owns the zebra? The Japanese.

```
for house in houses:
        if solver.Value(house_drink[house]["water"]):
            for nationality in nationalities:
                if solver.Value(house_nationality[house][nationality]):
                    print("The "+nationality+" drinks water.")
        if solver.Value(house_pet[house]["zebra"]):
            for nationality in nationalities:
                if solver.Value(house_nationality[house][nationality]):
                    print("The "+nationality+" owns the zebra.")
```

First-order logic as SAT problem

- While we can solve a first-order logic problem using the SAT solver, this is probably not the most efficient solution

- Dedicated solvers or programming languages (e.g. Prolog) are more efficient and more versatile for this task

- In particular they solve the problems by variable substitution avoiding the need to model all combinations of assignments as variables

- The main benefit of the presented approach is that it can be integrated with the other capabilities of the CP-SAT solver, in particular if the logic inference is only a sub-problem of something else

Thank you for your attention!