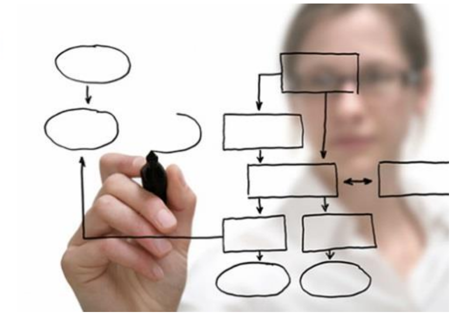
CIT CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT



Decision Analytics

Lecture 7: Boolean Satisfiability and Planning

Constraint Satisfaction Problem

- A **Constraint Satisfaction Problem** (CSP) is defined by
  - A tuple of n variables
    $$X =< x_1, \ldots, x_n >$$
  - a corresponding tuple of domains
    $$D =< D_1, \ldots, D_n >$$
    defining the potential values each variable can assume
    $$x_i \in D_i$$
  - and a tuple of t constraints
    $$C =< C_1, \ldots, C_t >$$
    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

Boolean Operators

- There are 3 basic Boolean operators

AND:	$x \wedge y$	(Evaluates to True, iff both x and y are True)
OR:	$x \vee y$	(Evaluates to True, iff any one of x or y are True)
NOT:	$\neg x$	(Evaluates to true, iff x is False)

- These operations are completely defined by the following table

x	y	$x \wedge y$	$x \vee y$	$\neg x$
False	False	False	False	True
True	False	False	True	False
False	True	False	True	
True	True	True	True	

Boolean Algebra

- De Morgan's laws:
  - A conjunction can be transformed into a disjunction

$$x \land y = \neg(\neg x \lor \neg y)$$

  - A disjunction can be transformed into a conjunction

$$x \lor y = \neg(\neg x \land \neg y)$$

Boolean Algebra

- Conjunctive normal form:
  Every Boolean term can be transformed into an equivalent term being a conjunction of disjunctions of literals, i.e. look like the following example

$$(x_1 \lor \neg x_2 \lor x_3) \land (\neg y_1 \lor y_2) \land (z_1 \lor z_2 \lor z_3 \lor z_4)$$

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

Boolean Algebra

- Useful secondary operators

  - The implication operator

$$x \Rightarrow y = \neg x \vee y$$

  - The XOR operator

$$x \otimes y = (x \wedge \neg y) \vee (\neg x \wedge y)$$

  - The Equivalence operator

$$x \equiv y = (x \wedge y) \vee (\neg x \wedge \neg y)$$

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. $colour[car, red]$, $\exists x\colon colour[x, blue]$)

- (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 ($s_1 \land s_2$, $s_1 \lor s_2$, $s_1 \Rightarrow s_2$, $s_1 \Leftrightarrow s_2$) 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. $\forall x: colour[x, red] \Rightarrow owner[x, John]$, 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 at least one 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. $\exists x\colon colour[x, green]$, 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. $colour$: $\{\text{car}, \text{house}\} \times \{red, green, blue\}$,
$owner$: $\{\text{car}, \text{house}\} \times \{Alan, Bob, Dave\}$, etc.)

- For each predicate P_n: $D_{O_n} \times D_{A_n}$, each element in the associated object domain $i \in D_{O_n}$ and each element in the associated attribute domain $j \in D_{A_n}$ 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

- A universal quantification over a predicate

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

- translates into a conjunction of the corresponding variables

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

- Similar the existential quantification over a predicate

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

- translates into a disjunction of the corresponding variables

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

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

Strategy planning as SAT problem

- Strategy planning algorithms are designed to find an **optimal plan of operations** that transform a given **initial configuration** into a desired **goal configuration**

- The state of a problem domain is represented by **predicates**, which indicate atomic sub-states of the overall problem being either true or false at any given point in time (e.g. “light is on”, “room occupied”, etc.)
- Further to these state descriptions we define **operators**, which can be applied to transform states. Operators are defined together with
  - A **pre-condition**, which must hold for the operation to be carried out (e.g. the operation “switch on light” can only happen if NOT “light is on” AND “room is occupied”)
  - A **post-condition**, which tells us how the states transform through the operation (e.g. after “switch on light” the predicate “light is on” holds)

Strategy planning as SAT problem

- To model a planning problem as SAT problem we first decide on a fixed number of time-steps T needed to complete the task
  (We can start with a smaller number, and if no strategy is found we increase the number of steps gradually until a solution is found)

- To create the corresponding CSP we generate one Boolean variables l per predicate per time-step, which indicates the state of the predicate at that point in time
- We then also generate one Boolean variable o per operator per time-step, which indicates if that operation is happening at that point in time

- So in case the problem domain has N predicates and M operators we create a **Boolean variable tuple**
- $$X = < l_{11}, \dots, l_{1T}, \dots, l_{N1}, \dots, l_{NT}, o_{11}, \dots, o_{1T}, \dots, o_{M1}, \dots, o_{MT} >$$

Strategy planning as SAT problem

- Next we add a constraint for the **initial state**, which is a conjunction of all predicates at time-step $t = 1$ indicating if they are true or false at the beginning

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

- Similar to the initial state a **goal state** is set, which defines what the targeted state of domain should be after the plan has been executed

$$l_{q_1 T} \wedge \cdots \wedge l_{q_{|Q|}} \wedge \neg l_{r_1} \wedge \cdots \wedge \neg l_{r_{|R|}}$$

(Note, that while the initial state needs to be complete, i.e. $|I| + |J| = N$, the goal state can be defined on only a subset of predicates, i.e. $|Q| + |R| \leq N$.)

Strategy planning as SAT problem

- The next step is to include constraints of all operators (**operator encoding**)

- Let the operator o_{mt} at time t have pre-conditions $p_{1t}, \dots, p_{ut}$ and post-conditions $e_{1(t+1)}, \dots, e_{v(t+1)}$, each defined by Boolean terms on the literals $l_{\cdot t}$ and $l_{\cdot(t+1)}$ respecivly, then we add an implication constraint

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

- As we have seen, this is equivalent to

$$\neg o_{kt} \vee ((p_{1t} \wedge \cdots \wedge p_{ut}) \wedge (e_{1(t+1)} \wedge \cdots \wedge e_{v(t+1)}))$$

- which means the operation has either not been executed, or both pre-condition and post-condition must hold

Strategy planning as SAT problem

- The next step is to encode the implicit assumption that predicates only change between time-steps due to the application of an operator, not ever by themselves (**frame axioms**)

- For all time-steps t and literals l_{nt} that are potentially affected by operators $o_{k_1 t}, \ldots, o_{k_{|K|} t}$ to change into $l_{n(t+1)}$ we add a constraint

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

- Which means that a if a predicate is true in the next time-step, it was either true already or some operator was active in this time-step that had an effect on it

Strategy planning as SAT problem

- Finally we need to ensure that only one operation per time-step is permissible (**complete exclusion axiom**)

- To do that we enforce for every pair of operations occurring at the same time

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

- As this is equivalent to $\neg(o_{kt} \wedge \neg o_{lt})$ it basically means "not both" at the same time.

Strategy planning as SAT problem

- To solve the strategy planning problem we construct a CSP with
  - the Boolean variables encoding predicates and operators
  - the initial state constraints
  - the goal state constraints
  - the operator encoding constraints
  - the frame axiom constraints
  - and the complete exclusion axiom constraints

- The solution contains both the evolution of states as well as the operators for each time-step that lead from the initial state to the goal state
- The final plan is the sequence of operators that evaluate to true

Thank you for your attention!