Automated Reasoning in Artificial Intelligence:

Introduction to Description Logic

Szymon Klarman

(*part of the content based on the tutorial by* **Stefan Schlobach**)

szymon.klarman@gmail.com

VU University Amsterdam, 2009-2012

Assignment

Tasks:

1. implement tableau algorithm for the DL $\mathcal{ALC}$ in LoTREC,
2. solve specified reasoning problems using your implementation,
3. elaborate on certain implementation issues,
4. propose an extension of the algorithm to cover certain constructs beyond $\mathcal{ALC}$.

To be delivered:

1. final presentation,
2. implementation + report.

LoTREC

LoTREC is a Generic Tableau Prover — a platform for prototyping *tableau algorithms* for a variety of modal logics.

`http://www.irit.fr/Lotrec/`

Good thing: a very handy and universal toolkit. Gives a quick and clean way of declaring:

- the syntax of your logic,
- the rules of your tableau algorithm,
- complex strategies of using those rules,
- sample formulas on which you can test the algorithms.

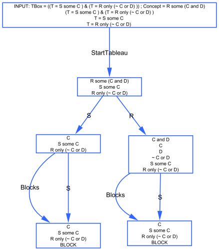
However: it has been developed in the academia:

- quite a few bugs and implementation problems,
- not always stable (save your work often!),
- not much documentation and user support.

Basic notions

LoTREC manipulates over graph structures called *pre-models*. A pre-model corresponds to a branch of a tableau.

From the DL perspective the *nodes* of a graph represent individuals, *links* between the nodes are roles, and *elements* of the nodes are concepts.



Implementing a tableau algorithm in LoTREC

We will implement a tableau algorithm for a fragment of modal logic **K** consisting of:

- *atomic propositions*: $p, q, r,$
- *atomic negation*: $\neg p,$
- *disjunction*: $p \vee q,$
- *possibility operator*: $\Diamond p.$

The tableau rules for this fragment are:

$\Rightarrow_\vee$ **IF** $(x : p \vee q) \in S$ **THEN** $S' := S \cup \{x : p\}$ **or** $S' := S \cup \{x : q\}$

$\Rightarrow_\Diamond$ **IF** $(x : \Diamond p) \in S$ **THEN** $S' := S \cup \{(x, y) : R,\ y : p\}$
where y is a 'fresh' variable in S

$\Rightarrow_\times$ **IF** $\{x : p, x : \neg p\} \subseteq S$ **THEN** mark the branch as CLOSED

We check whether *a set of formulas is satisfiable*.

Connectors

In the *connectors* tab you define the syntax of your logic:

- *Name*: name of the connector as used in the input formulas,
- *Arity*: the number of arguments taken by the connector,
- *Display*: the way the connector is displayed in the tableau,
- *Priority*, *Associative*: standard notions, but not relevant here.

Example:

Name: `or` | Arity: 2 | Display $\_ \vee \_$

The symbol $\_$ is used to mark the positions of the arguments w.r.t. the connector. Note that while defining the input formulas you can only use the *prefix notation*. Therefore:

input: `or P Q` | display: `P` $\vee$ `Q`

Rules

In the *rules* tab you specify the condition-action rules to be used in your algorithm.

Variables:

- *node variables*: x, y, node, node'...
- *expression variables* (formulas, relations): _x, _y, `and` _x _y...
- *expression constants*: CLASH, MARK...

Conditions:

- *hasElement*: a node x has element _y
- *hasNotElement*: a node x does not have element _y
- *isLinked*: a node x1 is related to a node x2 via relation _y
- *isAncestor*: a node x1 is an ancestor of node x2 (opposite to being a successor)

Rules

Conditions cont.:

- *isNewNode*: a node x1 is a node in the graph
  does not have a specific meaning but sometimes is necessary for creating complex patterns, e.g.:
  ```
  isAncestor node1 node2
  isNewNode node2
  ```
- *isAtomic*: the expression _x is atomic (is not a complex expression),
- *areNotIdentical*: node x1 is not the same node as x2,
- *contains*: node x1 contains all elements of node x2.

Rules

Actions:

- *add*: add expression _x to node y,
- *createNewNode*: create new node x,
- *link*: link node x1 to node x2 with relation _y,
- *stop*: stops the pre-model containing node x from developing further,
- *duplicate*: duplicates the current pre-model, e.g.

```
condition:  hasElement node (or _x _y)
action:     duplicate copy
            add node _x
            add copy.node _y
```

Strategies

In the *strategies* tab you write the pseudo-code of your algorithm based on the use of in-built *routines*, defined *rules* and other *strategies*.

- *no routine*:

  ```
  rule1
  rule2
  ```

  Meaning: take the pre-model, apply **rule1** as long as applicable, apply **rule2** as long as applicable, return the resulting pre-model.

- *repeat – end*:

  ```
  repeat
     rule1
     rule2
  end
  ```

  Meaning: As above, but after each run update the pre-model and repeat until saturation.

Strategies

- *firstRule – end*:

```
firstRule
   rule1
   rule2
end
```

  Meaning: take the pre-model, apply the first rule as long as applicable, return the resulting pre-model.

- *allRules – end*:

```
firstRule
   rule1
   allRules
     rule2
     rule3
   end
end
```

  Meaning: a block with no routine inside the firstRule block.

Strategies

- *applyOnce*:

```
applyOnce rule
```

  Meaning: apply the rule only once and then move on.

GOOD LUCK!