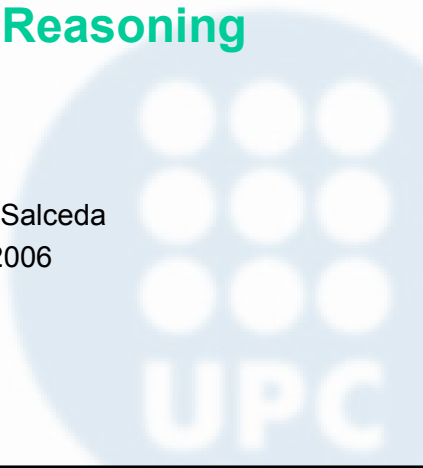


# 3. Reasoning in Agents

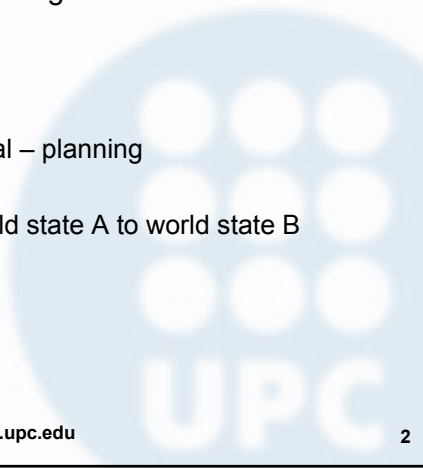
## Part 1: Introduction to Reasoning

Javier Vázquez-Salceda  
SMA-UPC 2006



### What is Reasoning?

- More than thinking
- Taking a set of facts and deriving new ones in a fixed way
- More specifically (usefully):
  - Reasoning to achieve a goal – planning
  - Problem Solving
  - Working out how to get world state A to world state B



## What is Reasoning?

### An example

- How do I achieve my dream of owning a house by the seaside?
  - Starting world state:
    - I have X amount of money
    - I have many facts about land, the city, planning permission, the housing market etc.
  - How do I achieve my goal state:
    - Where I have a house
    - (preferably one which is the BEST I could get with my money)
- The possibilities in the real world are (nearly!) infinite!

## Automated Reasoning

- Objective: carry out such inference **automatically** - without the need for human intervention
- This is very hard because:
  - The real world is complex (huge number of factors)
    - **inaccessible**
  - Resources are bounded (finite time and finite memory)
  - Things change (while I am thinking or acting the world may change)
    - **dynamic**
  - The world is uncertain (I cannot be sure that an action I take will have the expected outcome)
    - **non-deterministic**
  - There are other actors that might try to (intentionally or unintentionally) thwart my plans!
    - **non-deterministic**

## Reasoning Paradigms

- Key distinctions between paradigms
- Concrete approaches

## Key distinctions btw. Reasoning Paradigms

Monotonic vs. Non-Monotonic (I)

- **Monotonic**
  - A logical inference relation is *monotonic* if and only if, for all sets of propositions  $S$  and  $T$ , and for all propositions  $A$ , if  $S$  entails  $A$  (e.g.  $S \vdash A$ ) then  $(S \cup T) \vdash A$
  - First order logic is monotonic
  - Classical deduction - suitable for reasoning in open-ended situations
  - Absence of  $x$  implies  $x$  is unknown
  - A proposition  $A$  is false with respect to a set of propositions  $S$  when  $S \vdash \neg A$ .

## Key distinctions btw. Reasoning Paradigms

### Monotonic vs. Non-Monotonic (II)

- **Non-monotonic**
  - Logics in which the set of implications determined by a given group of premises does not necessarily grow, and can shrink, when new well-formed formulae are added to the set of premises
  - Absence of  $x$  implies  $x$  is false - closed world assumption
  - Prolog is non-monotonic
  - Reasoning to conclusions on the basis of incomplete information. Given more information, we are prepared to retract previously drawn inferences.
  - Agents are in general non-monotonic systems.

## Key distinctions btw. Reasoning Paradigms

### Abductive vs. Deductive

- **Abductive**
  - A form of inference that works forward to the best explanation
  - Example:
    - $D$  is a collection of data (facts, observations, givens),
    - $H$  explains  $D$  (or would, if true, explain  $D$ ),
    - No other hypothesis explains  $D$  as well as  $H$  does.
    - Therefore,  $H$  is probably correct.
  - Good for diagnosis, plan recognition, natural language understanding, vision
  - Explanation is not necessarily true
- **Deductive**
  - Predictive
  - Works from premises to conclusion
  - Inference rules drive the process
  - Uses the existence of facts to infer (via rules) the existence of new facts
  - Conclusion is proven with respect to available facts

## Key distinctions btw. Reasoning Paradigms

### Forward Chaining vs. Backward Chaining

- **Forward Chaining**

- An implementation of deduction
- Rules are used to deduce new facts from existing facts
- Process continues until no more rules apply

- **Backward Chaining**

- Works backwards from goal to current situation
- Rules are used to infer that a (sub)goal holds then the preconditions (left hand side of rule) also hold
- Process moves backwards down chain of reasoning until no more rules apply
- Prolog style

## Reasoning Paradigms: Concrete Approaches

### Essential elements

- A **description** of the world
- A specification of the **goal**
- A **search space** of things to do (possibly vast)
- Some way to traverse the search space
  - Need of some **algorithm/strategy/heuristic** function to guide the traversal.

## Reasoning Paradigms: Concrete Approaches

- Approaches
  - Case-Based Reasoning
  - Model-Based Reasoning
  - Qualitative Reasoning
  - Planning Systems
  - Constraint Satisfaction Reasoning
  - Rule-Based Reasoning
  - Ontological Reasoning
  - Symbolic Reasoning
  - Logic Programming
- These are not disjoint. One can have combined approaches such as Constraint Logic-Based Planning Systems.

## Reasoning Paradigms: Concrete Approaches

### Case-Based Reasoning

- *“I remember solving a problem like this some time ago ...”*
- Functions:
  - A case-base of previous problem-solution pairs
  - An indexing scheme which classifies problems and cases
- When a new problem arises:
  - Find the closest previous problem(s) and solution(s)
  - Try to adapt the solution(s) to the new problem
  - Apply the new solution
  - Optionally add the new experience to the case-base
- Challenge is how to create initial case base

## Reasoning Paradigms: Concrete Approaches

### Model-Based Reasoning

- *“I understand how this system and its components work based on their input parameters”*
  - Component models (e.g. failure modes)
  - Differential equations, logical models, ...
- Combined:
  - Brute force search algorithms
- Often used for system diagnosis:
  - Why is my washing machine not working?
  - Why is this electric circuit failing?

## Reasoning Paradigms: Concrete Approaches

### Qualitative Reasoning

- *“Gravity works downwards, if I jump out of this plane I will probably fall”*
- Approximate way of reasoning
  - Useful to reason about too complex (e.g. chaotic, fractal) problems
  - E.g. physical world properties
  - Use “naïve” (but often useful) deduction rules

## Reasoning Paradigms: Concrete Approaches

### Planning

- *“From my current world state I can apply a sequence of possible actions to get to the goal”*
- Different types:
  - **State based planning** - we search the combinations of all actions (Domain driven)
  - **Hierarchical Task Network** - we search the possible plans (Knowledge based)
- A lot of different search techniques, world models and reasoning approaches are used
  - Linear/non-linear
  - Continuous/discrete
  - Temporal issues

## Reasoning Paradigms: Concrete Approaches

### Constraint Satisfaction

- *“The world is a set of interdependent choices. If I make one, it may affect another”*
- Problem:
  - A **set of variables**  $V$  (each with a possible set of values  $v_{i1}$ - $v_{in}$ )
  - A **set of constraints** linking variables  $C(v_{i1}, v_{i2}, v_{i3})$  such as “if my trousers are green my shirt should not be blue”
  - What are the legal combinations of values for each variable? Or, which choices fit together given the constraints
- Many search techniques
  - Propagating constraint effects, subdividing the constraint graph etc.
  - Related problems: dynamically changing choices/options, uncertainty, ...
  - But algorithms are typically quite expensive (complexity)
  - Domain specific SAT solvers relatively efficient
  - Good heuristics

## Reasoning Paradigms: Concrete Approaches

### Rule-Based Reasoning

- *“If the light is red STOP, if it is raining I must be wet, ...”*
- Functions by:
  - Accumulating a **set of rules** relating PRE-conditions to inferences or actions
  - A **fact base** allowing the rules to fire iteratively when the facts fit the rule preconditions
  - **Heuristics** to select one rule when several satisfy the preconditions
- Reasoning happens by traversing the facts available
- We will see more of this later

## Reasoning Paradigms: Concrete Approaches

### Ontological Reasoning

- There are two approaches
  - **Description logic reasoning** over ontological knowledge (e.g. class membership inference) - such as RACER etc.
  - Adapting the data models in each of the other schemes to **use objects in agreed ontologies** - that is, using any of the previous approaches, but the facts are represented via ontologies

## Reasoning Paradigms: Concrete Approaches

### Symbolic Reasoning

- The world or a portion of it is represented in terms of *formulae in some logic*
- Reasoning is based on inference.
- Different types of logic are used
  - description logic
  - temporal logic
  - BDI logic
  - epistemic logic
  - deontic logic
  - ...
- We will come back on this later

## Reasoning Paradigms: Concrete Approaches

### Logic Programming

- “Given this set of rules with pre and post conditions which information can I obtain from it”
- Based in model theoretic principles
- Provides possible views of solutions
- Prolog (with closed world assumption)
- **Answer Set Programming** (no closed assumption)

## Reasoning Paradigms

### Evolution in Agent Architectures

- Originally (1956-1985), pretty much all agents designed within AI were *symbolic reasoning agents*
- Its purest expression proposes that agents use *explicit logical reasoning* in order to decide what to do
- Problems with symbolic reasoning led to a reaction against this — the so-called *reactive agents* movement, 1985–present
- From 1990-present, a number of alternatives proposed: *hybrid architectures*, which attempt to combine the best of reasoning and reactive architectures

## Deductive Reasoning Agents

- **Rule-Based Systems**
- **Symbolic Reasoning Agents**
- **Deductive Reasoning Agents**
- **Agent Oriented Programming**

## Foundations: Rule-based systems

### Expert Systems and rules

- **Expert Systems** provide expert quality advice, diagnoses and recommendations on real world problems
- Designed to perform function of a human expert. E.g. Medical diagnosis - program takes place of a doctor; given a set of symptoms the system suggests a diagnosis and treatment
- The knowledge base of an expert system is often **rule based**
  - the system has a list of rules which determine what should be done in different situations
  - These rules are initially designed by human expert(s)
  - The rules are called **production rules**

## Foundations: Rule-based systems

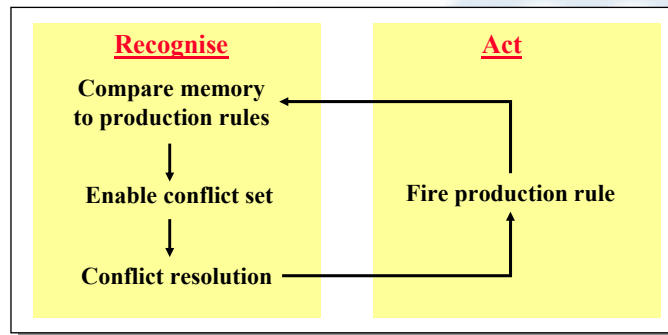
### Rules

- Each rule has two parts, the **condition-action pair**
  - **Condition** - what must be true for the rule to fire
  - **Action** - what happens when the condition is met
- Can also be thought of as IF-THEN rules
  - IF sunny(weather) AND outdoors(x)  
THEN print "Take your sunglasses x"
  - IF >30(temperature)  
THEN print "take some water"
- The contents of the **working memory** are constantly compared to the production rules
- When the contents match the condition of a rule, that rule is **fired**, and its action is executed
- More than one production rule may match the working memory - **conflict set**

## Foundations: Rule-based systems

### Recognise-Act Cycle

- The system cycles around in the **recognise-act cycle**
- Whenever a condition is matched, it is added to the **conflict set** - all the rules which are currently matched
- The system must then decide which rule within the conflict set to fire - **conflict resolution**



jvazquez@lsi.upc.edu

25

## Symbolic Reasoning Agents

- The classical approach to building agents is to view them as a particular type of **knowledge-based system**, and bring all the associated (discredited?!) methodologies of such systems to bear
- This paradigm is known as **symbolic AI**
- We define a deliberative agent or agent architecture to be one that:
  - contains an explicitly represented, **symbolic model of the world**
  - makes decisions (for example about what actions to perform) via symbolic reasoning

jvazquez@lsi.upc.edu

26

## Symbolic Reasoning Agents

### Problems to solve

- If we aim to build an agent in this way, there are two key problems to be solved:
  1. *The transduction problem:*  
that of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful...vision, speech understanding, learning
  2. *The representation/reasoning problem:*  
that of how to symbolically represent information about complex real-world entities and processes, and how to get agents to reason with this information in time for the results to be useful...knowledge representation, automated reasoning, automatic planning

## Symbolic Reasoning Agents

### Problems to solve

- Most researchers accept that **neither problem is anywhere near solved**
- Underlying problem lies with the **complexity of symbol manipulation algorithms** in general: many (most) search-based symbol manipulation algorithms of interest are **highly intractable**
- Because of these problems, some researchers have looked to alternative techniques for building agents...

## Deductive Reasoning Agents

- How can an agent decide what to do using theorem proving?
- Basic idea is to use logic to encode a theory **stating the best action** to perform in any given situation
- Let:
  - $\rho$  be this theory (typically a set of rules)
  - $\Delta$  be a logical database that describes the current state of the world
  - $Ac$  be the set of actions the agent can perform
  - $\Delta \vdash_{\rho} \phi$  mean that  $\phi$  can be proved from  $\Delta$  using  $\rho$

## Deductive Reasoning Agents

### Action selection

```

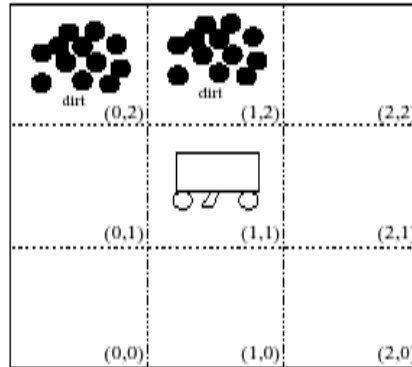
/* try to find an action explicitly prescribed */
for each  $a \in Ac$  do
  if  $\Delta \vdash_{\rho} Do(a)$  then
    return  $a$ 
  end-if
end-for
/* try to find an action not excluded */
for each  $a \in Ac$  do
  if  $\Delta \not\vdash_{\rho} \neg Do(a)$  then
    return  $a$ 
  end-if
end-for
return null /* no action found */

```

## Deductive Reasoning Agents

Example: the Vacuum World

- Goal is for the robot to clear up all dirt



jvazquez@lsi.upc.edu

31

## Deductive Reasoning Agents

Example: the Vacuum World

- Use 3 domain predicates to solve problem:

$In(x, y)$             agent is at  $(x, y)$   
 $Dirt(x, y)$           there is dirt at  $(x, y)$   
 $Facing(d)$           the agent is facing direction  $d$

- Possible actions:

$Ac = \{turn, forward, suck\}$

P.S. *turn* means "turn right"

- Rules  $\rho$  for determining what to do:

$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \rightarrow Do(forward)$

$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \rightarrow Do(forward)$

$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \rightarrow Do(turn)$

$In(0,2) \wedge Facing(east) \rightarrow Do(forward)$

- ...and so on!
- Using these rules (+ other obvious ones), starting at  $(0, 0)$  the robot will clear up dirt

jvazquez@lsi.upc.edu

32

## Deductive Reasoning Agents

Example: the Vacuum World

- Problems:
  - How to convert video camera input to *Dirt(0, 1)*?
  - decision making assumes a *static* environment: *calculative rationality*
  - decision making using first-order logic is *undecidable!*
- Even where we use *propositional* logic, decision making in the worst case means solving *co-NP-complete* problems (PS: co-NP-complete = bad news!)
- Typical solutions:
  - weaken the logic
  - use symbolic, non-logical representations
  - shift the emphasis of reasoning from *run time* to *design time*
- We will look at some examples of these approaches

## Agent Oriented Programming

- Yoav Shoham introduced “Agent Oriented Programming” in 1990:
  - “new programming paradigm, based on a societal view of computation”.
- Key idea: directly programming agents in terms of intentional notions like belief, commitment, and intention.
- The motivation behind such a proposal is that, as we humans use the intentional stance as an abstraction mechanism for representing the properties of complex systems.
  - In the same way that we use the intentional stance to describe humans, it might be useful to use to describe the programming of machines.
- Shoham suggested that a complete AOP system will have 3 components:
  - a logic for specifying agents and describing their mental states
  - an interpreted programming language for programming agents
  - an ‘agentification’ process, for converting ‘neutral applications’ (e.g., databases) into agents
- Relationship between logic and programming language is *semantics*

## Agent Oriented Programming

### AGENT0

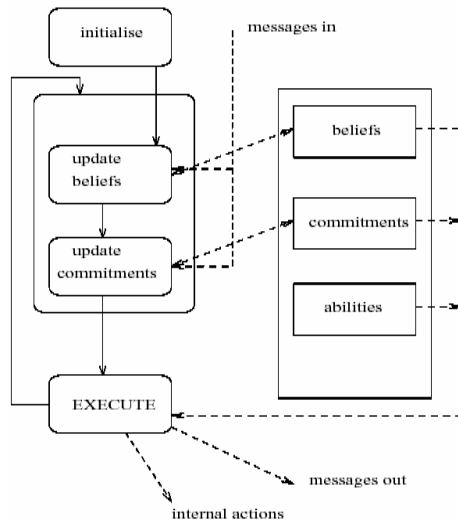
- AGENT0 is the first AOP language.
- AGENT0 is implemented as an extension to LISP
- Each agent in AGENT0 has 4 components:
  - a set of **capabilities** (things the agent can do)
  - a set of **initial beliefs**
  - a set of **initial commitments** (things the agent will do)
  - a set of **commitment rules**
- The key component, which determines how the agent acts, is the commitment rule set
- Each commitment rule contains
  - a **message condition**
  - a **mental condition**
  - an **action**

## Agent Oriented Programming

### AGENT0

- On each 'agent cycle'...
  - The message condition is matched against the messages the agent has received
  - The mental condition is matched against the beliefs of the agent
  - If the rule fires, then the agent becomes committed to the action (the action gets added to the agent's commitment set)
- Actions may be
  - **private**:  
an internally executed computation, or
  - **communicative**:  
sending messages
- Messages are constrained to be one of three types:
  - "requests" to commit to action
  - "unrequests" to refrain from actions
  - "informs" which pass on information

## Agent Oriented Programming AGENT0



jvazquez@lsi.upc.edu

37

## Agent Oriented Programming AGENT0

- A commitment rule:

```

COMMIT (
  (agent, REQUEST, DO(time, action)), ;;; msg condition
  (B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyaction)]
  ), ;;; mental condition
  self,
  DO(time, action)
)

```

- This rule may be paraphrased as follows:  
if I receive a message from *agent* which requests me to do *action* at *time*, and I believe that:
  - *agent* is currently a friend
  - I can do the action
  - At *time*, I am not committed to doing any other action
 then commit to doing *action* at *time*

jvazquez@lsi.upc.edu

38

## Agent Oriented Programming

### AGENT0 and PLACA

- AGENT0 provides support for multiple agents to cooperate and communicate, and provides basic provision for debugging...
- ...it is, however, a *prototype*, that was designed to illustrate some principles, rather than be a production language
- A more refined implementation was developed by Thomas, for her 1993 doctoral thesis
- Her Planning Communicating Agents (PLACA) language was intended to address 2 severe drawbacks to AGENT0:
  - the inability of agents to plan,
  - the inability of agents to communicate requests for action via high-level goals
- Agents in PLACA are programmed in much the same way as in AGENT0, in terms of *mental change* rules

## Agent Oriented Programming

### AGENT0 and PLACA

- An example mental change rule:
 

```
((self ?agent REQUEST (?t (xeroxed ?x)))
  (AND (CAN-ACHIEVE (?t xeroxed ?x))
    (NOT (BEL (*now* shelving)))
    (NOT (BEL (*now* (vip ?agent)))))
  ((ADOPT (INTEND (5pm (xeroxed ?x)))))
  ((?agent self INFORM
    (*now* (INTEND (5pm (xeroxed ?x)))))))
```
- This can be paraphrased as follows:
 

if someone asks you to xerox something, and you can, and you don't believe that they're a VIP, or that you're supposed to be shelving books, then

  - adopt the intention to xerox it by 5pm, and
  - inform them of your newly adopted intention

## MetateM

- MetateM is a multi-agent language in which each agent is programmed by giving it a **temporal logic specification** of the behavior it should exhibit
- These specifications are executed directly in order to generate the behavior of the agent
- Temporal logic is classical logic augmented by *modal operators* for describing how the truth of propositions changes over time

## MetateM

### Temporal Logic operators

- For example. . .
  - $\square important(agents)$   
means “it is now, and will always be true that agents are important”
  - $\diamond important(ConcurrentMetateM)$   
means “sometime in the future, ConcurrentMetateM will be important”
  - $\blacklozenge important(Prolog)$   
means “sometime in the past it was true that Prolog was important”
  - $(\neg friends(us)) \mathcal{U} apologize(you)$   
means “we are not friends until you apologize”
  - $\bigcirc apologize(you)$   
means “tomorrow (in the next state), you apologize”.
  - $\odot prepareSlides(me)$   
means “yesterday (previous run) I prepared my slides”.
  - $post-doc(me) \mathcal{S} year(2003)$   
means “I am a posdoc researcher since 2003”

## MetateM

### Execution rules

- MetateM is a framework for *directly executing* temporal logic specifications
- The root of the MetateM concept is Gabbay's *separation theorem*: Any arbitrary temporal logic formula can be rewritten in a logically equivalent *past*  $\Rightarrow$  *future* form.
- This *past*  $\Rightarrow$  *future* form can be used as **execution rules**
- A MetateM program is a set of such rules
- Execution proceeds by a process of continually matching rules against a "history", and *firing* those rules whose antecedents are satisfied
  - Execution is thus a process of iteratively generating a model for the formula made up of the program rules
- The instantiated future-time consequents become *commitments* which must subsequently be satisfied

jvazquez@lsi.upc.edu

43

## MetateM

### Example

- An example MetateM program: the resource controller...

$$\begin{aligned} \forall x \quad \bigcirc \text{ask}(x) &\Rightarrow \diamond \text{give}(x) \\ \forall x,y \quad \text{give}(x) \wedge \text{give}(y) &\Rightarrow (x=y) \end{aligned}$$

- First rule ensure that an 'ask' is eventually followed by a 'give'
- Second rule ensures that only one 'give' is ever performed at any one time
- There are algorithms for executing MetateM programs that appear to give reasonable performance

jvazquez@lsi.upc.edu

44

## Concurrent MetateM

- Concurrent MetateM provides an operational framework through which societies of MetateM processes can operate and communicate
- It is based on a model for concurrency in executable logics: the notion of executing a logical specification to generate individual agent behavior
- A Concurrent MetateM system contains a number of agents (objects), each object has 3 attributes:
  - a name
  - an interface
  - a MetateM program

## Concurrent MetateM

### object interface

- An object's interface contains two sets:
  - *environment predicates* — these correspond to messages the object will accept
  - *component predicates* — correspond to messages the object may send
- For example, a 'stack' object's interface:
 

```
stack(pop, push)[popped, stackfull]
{pop, push} = environment preds
{popped, stackfull} = component preds
```
- If an agent receives a message headed by an environment predicate, it accepts it
- If an object satisfies a commitment corresponding to a component predicate, it broadcasts it

## Concurrent MetateM

### Example

- To illustrate the language Concurrent MetateM in more detail, here are some example programs...
- Snow White has some sweets (resources), which she will give to the Dwarves (resource consumers)
- She will only give to one dwarf at a time
- She will always eventually give to a dwarf that asks
- Here is Snow White, written in Concurrent MetateM:

Snow-White(ask)[give]:  
 $\odot \text{ask}(x) \Rightarrow \diamond \text{give}(x)$   
 $\text{give}(x) \wedge \text{give}(y) \Rightarrow (x = y)$

## Concurrent MetateM

### Example

- The 'eager' dwarf asks for a sweet initially, and then whenever he has just received one, asks again

eager(give)[ask]:  
 $\text{start} \Rightarrow \text{ask}(\text{eager})$   
 $\odot \text{give}(\text{eager}) \Rightarrow \text{ask}(\text{eager})$

- Some dwarves are even less polite: 'greedy' just asks every time

greedy(give)[ask]:  
 $\text{start} \Rightarrow \square \text{ask}(\text{greedy})$

## Concurrent MetateM

### Example

- Fortunately, some have better manners; 'courteous' only asks when 'eager' and 'greedy' have eaten

$$\begin{aligned} \text{courteous}(\text{give})[\text{ask}]: \\ ((\neg \text{ask}(\text{courteous}) \mathcal{S} \text{give}(\text{eager})) \wedge \\ (\neg \text{ask}(\text{courteous}) \mathcal{S} \text{give}(\text{greedy}))) \Rightarrow \\ \text{ask}(\text{courteous}) \end{aligned}$$

- And finally, 'shy' will only ask for a sweet when no-one else has just asked

$$\begin{aligned} \text{shy}(\text{give})[\text{ask}]: \\ \text{start} \Rightarrow \diamond \text{ask}(\text{shy}) \\ \text{ask}(x) \Rightarrow \neg \text{ask}(\text{shy}) \\ \text{give}(\text{shy}) \Rightarrow \diamond \text{ask}(\text{shy}) \end{aligned}$$

## Concurrent MetateM

- Summary:
  - an(other) experimental language
  - very nice underlying theory...
  - ...but unfortunately, lacks many desirable features — could not be used in current state to implement 'full' system
  - currently prototype only, full version on the way!

## References

- [1] Wooldridge, M. "Introduction to Multiagent Systems". John Wiley and Sons, 2002.
- [2] Weiss, G. "Multiagent Systems: A modern Approach to Distributed Artificial Intelligence". MIT Press. 1999. ISBN 0262-23203

These slides are based mainly in [2] and material from M. Wooldridge, J. Padget and M. de Vos