Deductive Reasoning Agents

Agent-Oriented Software Engineering

A.A. 2019-2020

Prof. Paolo Giorgini

UNIVERSITY OF TRENTO - Italy

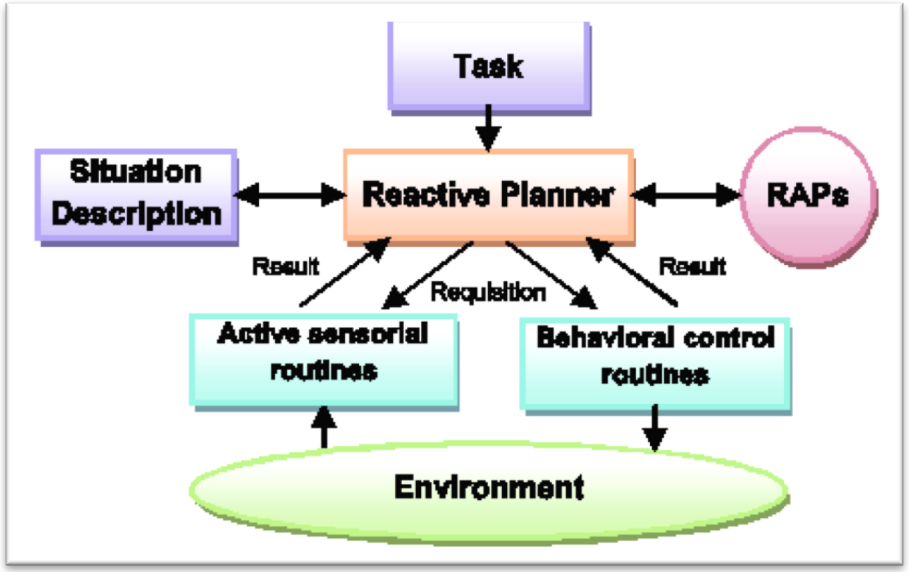
Department of Information
and Communication Technology

Agent Architectures

- We want to build agents, that enjoy the properties of autonomy, reactiveness, pro-activeness, and social ability that we talked about earlier
- This is the area of *agent architectures*
- Maes defines an agent architecture as:
  - A particular methodology for building agents
  - It specifies how… the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact
  - The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions… and future internal state of the agent
  - An architecture encompasses techniques and algorithms that support this methodology

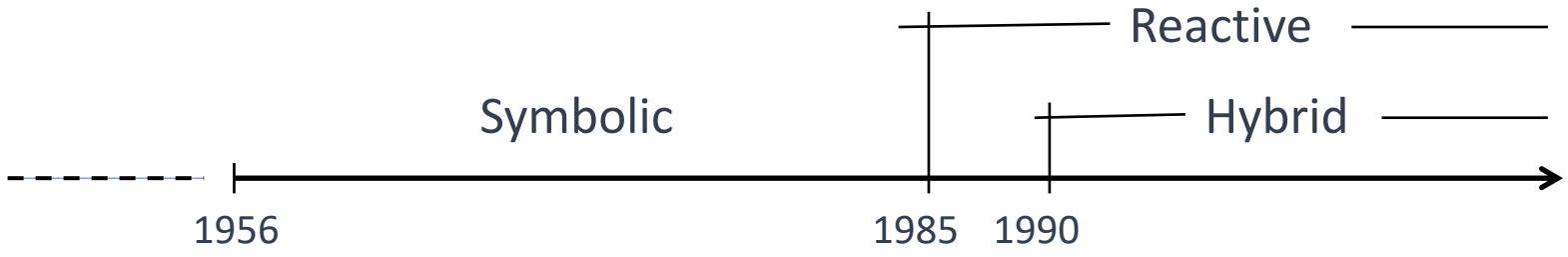
Agent Architectures

- Kaelbling considers an agent architecture to be:
  - A specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules.
  - A more abstract view of an architecture is as a general methodology for designing particular modular decompositions for particular tasks.



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



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

Symbolic Reasoning Agents

To build an agent in this way, there are two key problems:

- 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
- 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

Underlying problem lies with the complexity of symbol manipulation algorithms in general

- many (most) search-based symbol manipulation algorithms of interest are highly intractable

Deductive Reasoning Agents

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

Deductive Reasoning Agents: action selection

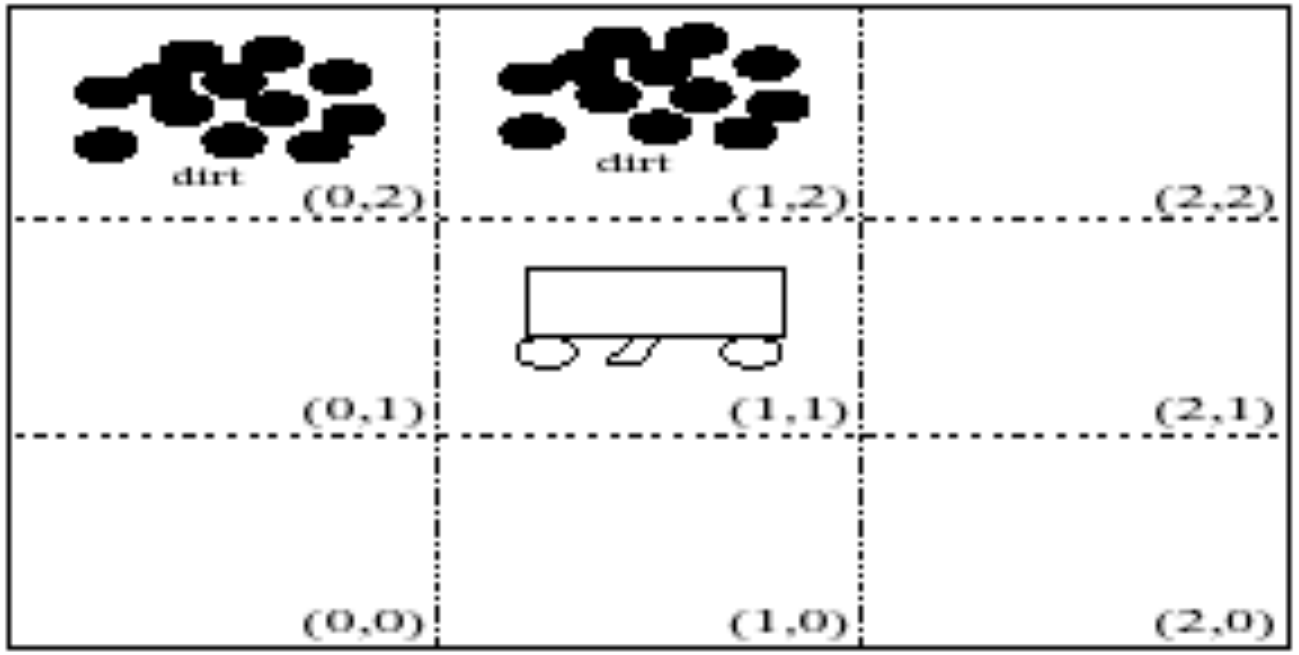
```
/* try to find an action explicitly prescribed */
for each a ∈ Ac do
        if Δ |-ρ Do(a) then-
                return a
        end-if
end-for
```

```
/* try to find an action not excluded */
for each a ∈ Ac do
        if Δ |-ρ ¬Do(a) then
                return a
        end-if
end-for
```

return *null* /* *no action found* */

Deductive Reasoning Agents

- An example: The Vacuum World
  - Goal is for the robot to clear up all dirt



Deductive Reasoning Agents

- 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}

  … *turn* means “turn right”

Deductive Reasoning Agents

- Rules ρ that govern the agent's behavior:
  - $\varphi(\ldots) \rightarrow \psi(\ldots)$
  - φ and ψ are predicates over some arbitrary list of constants and variables
  - If φ matches against the agent's database, then ψ can be concluded, with any variables in ψ instantiated
- Rules
  - Cleaning action rule: `In(x,y)^Dirt(x,y)`→`Do(suck)`
  - This action will take priority over all other possible behaviors of the agent (such as navigation)
  - In case the rule does not match against the database, the basic action of the agent will be to traverse the world

Deductive Reasoning Agents

- We can hardwire the basic navigation , so the robot will always move from (0,0) to (0,1) to (0,2) and then to (1,2), (1,1) and so on. When it reaches (2,2), it must head back to (0,0):
- Rules dealing with the traversal up to (0,2) are very simple

$$
\begin{aligned}
In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) &\longrightarrow Do(forward) \\
In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) &\longrightarrow Do(forward) \\
In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) &\longrightarrow Do(turn) \\
In(0,2) \wedge Facing(east) &\longrightarrow Do(forward)
\end{aligned}
$$

- Using these rules (+ other obvious ones), starting at $(0, 0)$ the robot will clear up dirt ?????
- Notice that in each rule, we must explicitly check whether the antecedent of the cleaning rule fires.

Deductive Reasoning Agents

In an agent's database we have facts like

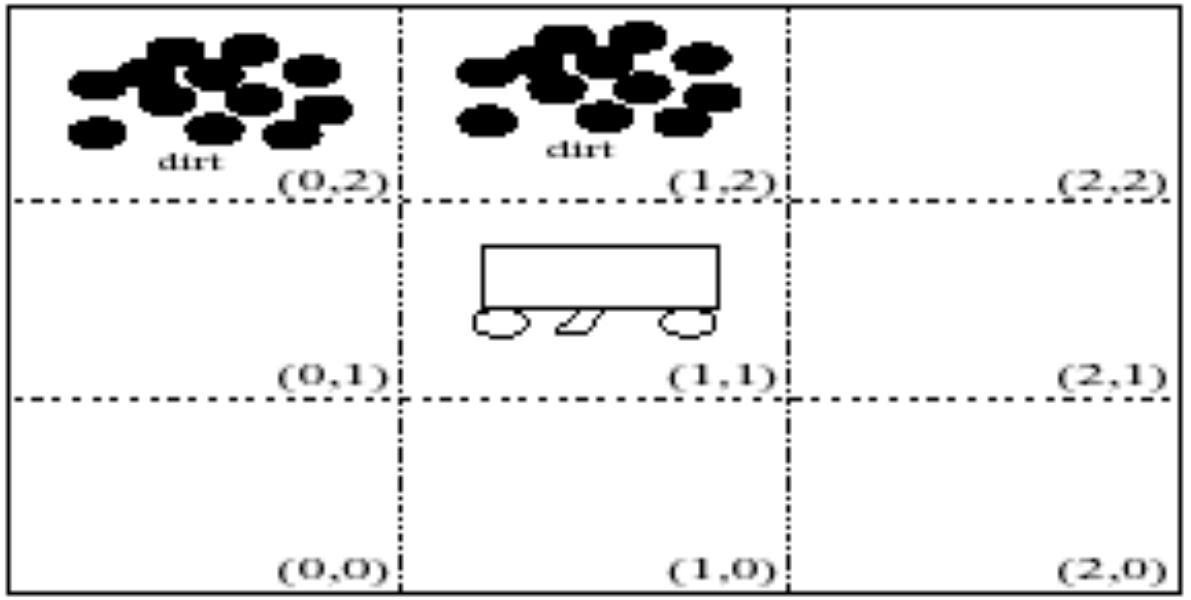
$$\texttt{In(1,1) Dirt(0,2) Dirt(1,2) Facing(West)}$$

while formulas like

$$\texttt{In(x,y)} \wedge \texttt{Dirt(x,y)} \rightarrow \texttt{Do(suck)}$$

are part of the reasoning framework and not in the database (compare: program vs. interpreter)

Exercise



Sates

`In(x,y)`

`Facing(d): d={n,s,e,w}`

`Dirt(x,y)`

Actions

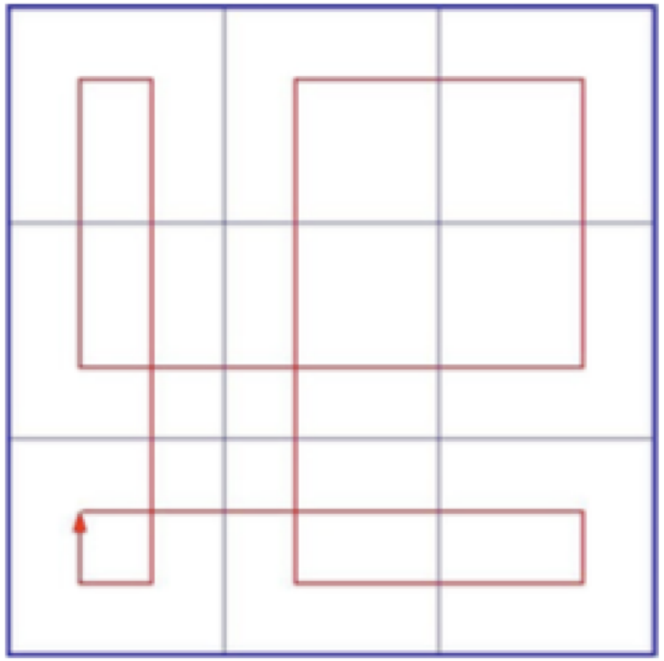
`DO(a): a={suck, rotate, forward}`

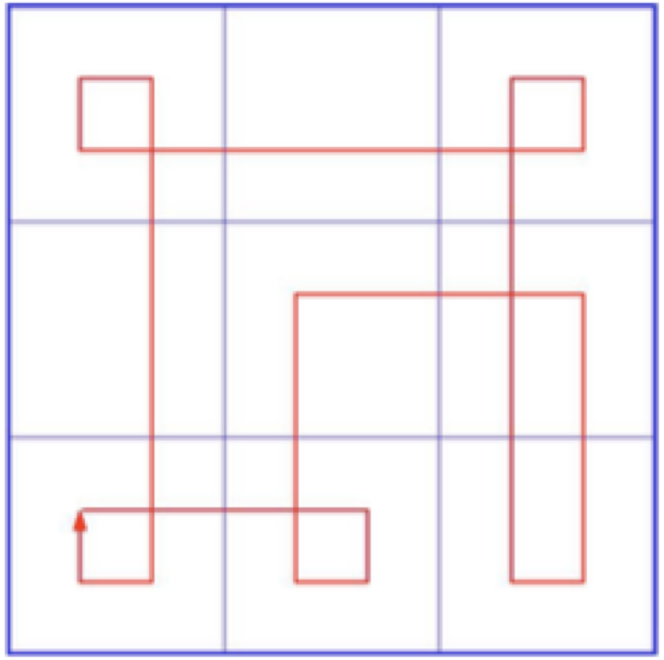
"rotate" means rotate clockwise the direction of moving

Solution in two steps

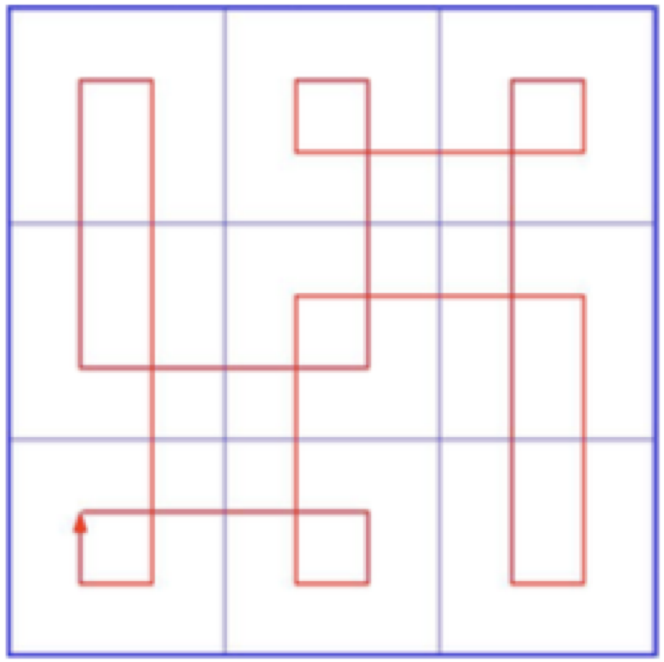
- Find a cyclic path that allows the robot to go through all positions
- Write the navigations rules
  - The order will define the priority of the rules

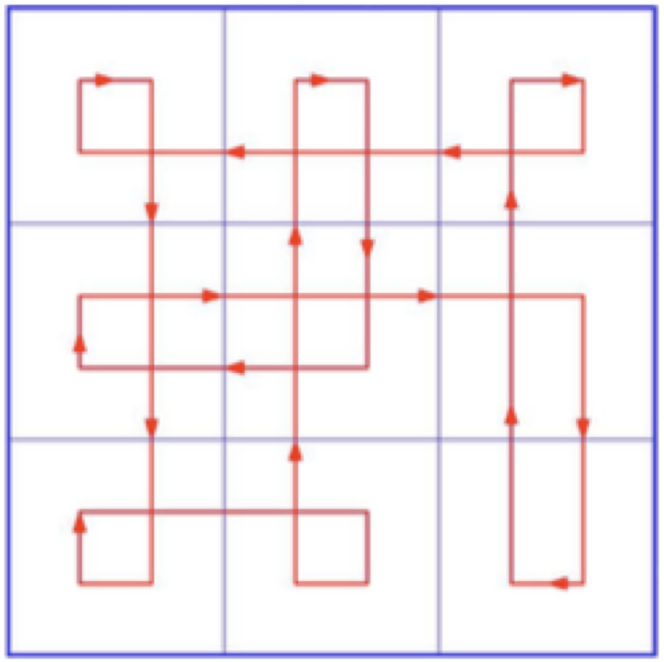
Possible solutions





Others





Rules

$$In(x,y) \wedge Dirt(x,y) \rightarrow DO(suck)$$

$$[In(0,0) \vee In(1,2)] \wedge \neg Facing(E) \rightarrow DO(turn)$$

$$[In(2,0) \vee In(2,1)] \wedge \neg Facing(W) \rightarrow DO(turn)$$

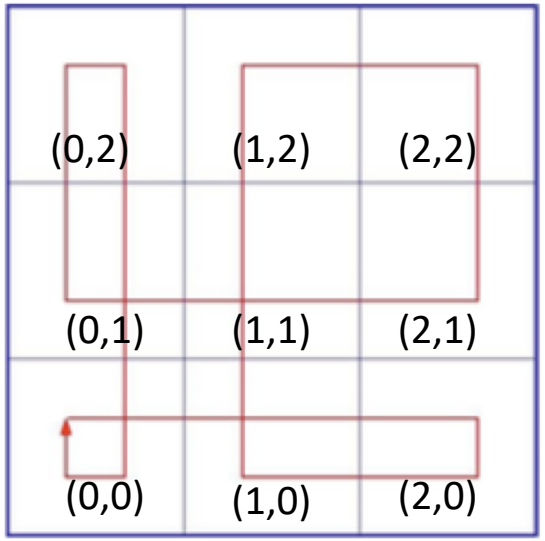
$$[In(0,2) \vee In(2,2)] \wedge \neg Facing(S) \rightarrow DO(turn)$$

$$In(1,0) \wedge [Facing(W) \vee Facing(S)] \rightarrow DO(turn)$$

$$In(1,1) \wedge [Facing(E) \vee Facing(S)] \rightarrow DO(turn)$$

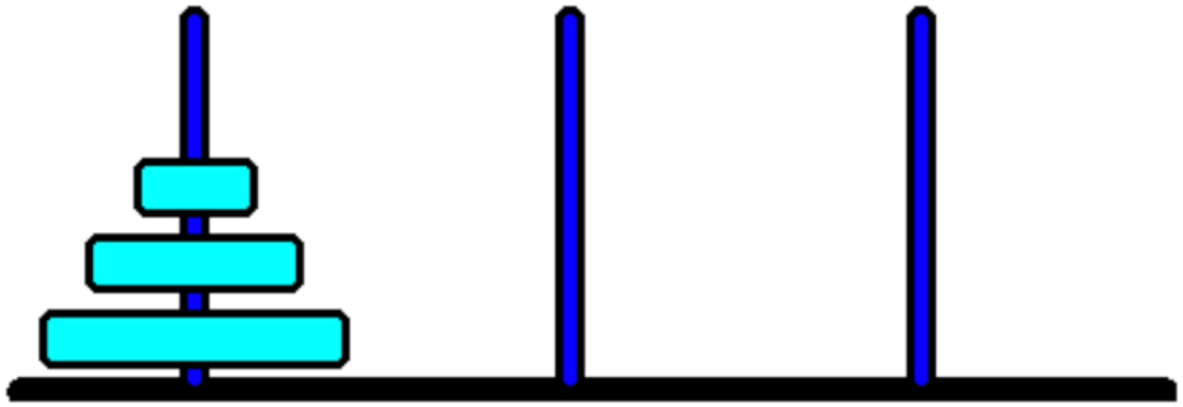
$$In(0,1) \wedge [Facing(W) \vee Facing(E)] \rightarrow DO(turn)$$

$$DO(forward)$$



The Tower of Hannoi (Prolog)

```
move(1,X,Y,_) :-
      write('Move top disk from '),
      write(X),
      write(' to '),
      write(Y),
      nl.
move(N,X,Y,Z) :-
      N>1,
      M is N-1,
      move(M,X,Z,Y),
      move(1,X,Y,_),
      move(M,Z,Y,X).
```



Executing the Prolog program

```
?- move(3,left,right,center).
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
true
```

Problems with DRA

- If we can prove `Do(a)`, then `a` is an optimal action
  - namely, the best action that could be performed when the environment is as described in the database
- Suppose at time t_1 the agent's database is Δ_1 and `a` is the optimal action to be performed. At time t_2, the agent deduces `a` (it took some time to do it) so the agent start to do `a`
- What happened between t_1 and t_2?
  - If $t_2 - t_1$ is infinitesimal, no problem (decision making was instantaneous)
  - Logic-based reasoning is not instantaneous !

Calculative rationality

- An agent is said to enjoy the property of calculative rationality if and only if its decision –making apparatus will suggest an action that was optimal when the agent decision-making process began.
- Calculative rationality is clearly not acceptable in environment that change faster than the agent make decisions
- Moving away from strictly logical representation languages and complete sets of deduction rules, one can build agents that enjoy respectable performance
  - Losing obviously advantages like simplicity, elegance, and logical semantics

Other problems

The problem of “translating” raw data provided by the agent’s sensors into an internal symbolic form

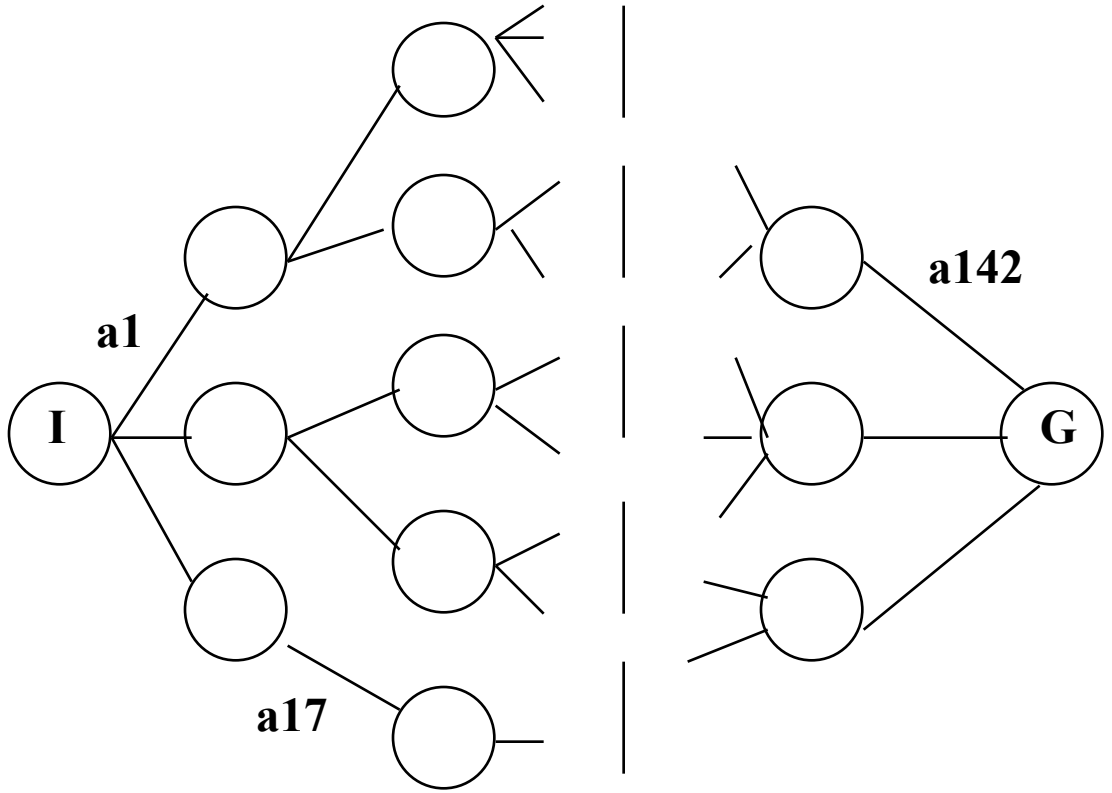
- How to convert video camera input to $Dirt(0, 1)$?
- Representing properties of dynamic, real-world is extremely hard (e.g., represent and reason about temporal information)

- Decision making using first-order logic is *undecidable*!
  - Even where we use *propositional* logic, decision making in the worst case means solving NP-complete problems

- Typical solutions:
  - weaken the logic
  - use symbolic, non-logical representations
  - shift the emphasis of reasoning from *run time* to *design time*

Planning Systems (in general)

- Planning systems find a sequence of actions that transforms an initial state into a goal state



Planning

- Planning involves issues of both Search and Knowledge Representation
- Sample planning systems:
  - Robot Planning (STRIPS)
  - Planning of biological experiments (MOLGEN)
  - Planning of speech acts
- For purposes of exposition, we use a simple domain – The Blocks World

The Blocks World

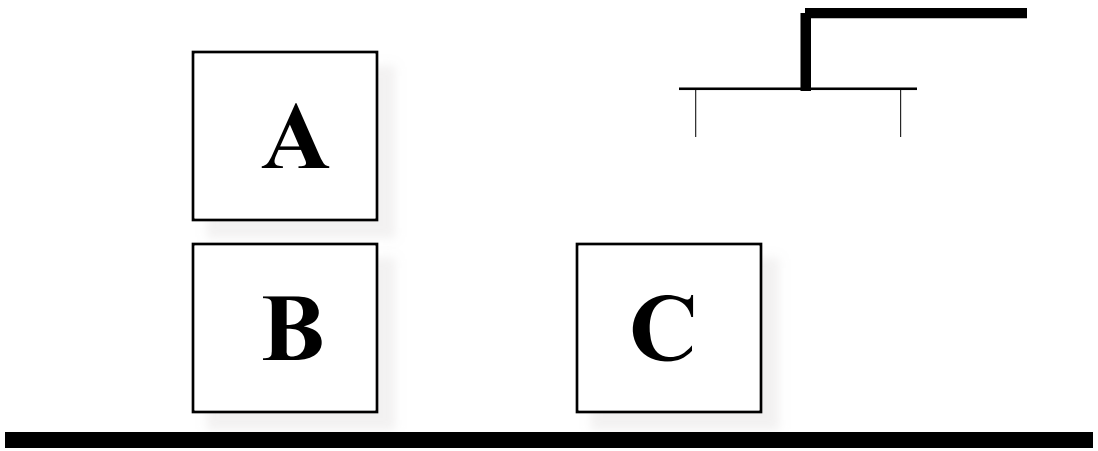
- The Blocks World (today) consists of equal sized blocks on a table
- A robot arm can manipulate the blocks using the actions:

```
UNSTACK(a,b)
STACK(a,b)
PICKUP(a)
PUTDOWN(a)
```

The Blocks World

- We also use predicates to describe the world:

```
ON(A,B)
ONTABLE(B)
ONTABLE(C)
CLEAR(A)
CLEAR(C)
ARMEMPTY
```



Logical Formulas to Describe Facts Always True of the World

- And of course we can write general logical truths relating the predicates:

$$\exists x \ HOLDING(x) \rightarrow \neg ARMEMPTY$$

$$\forall x (ONTABLE(x) \rightarrow \neg \exists y (ON(x, y)))$$

$$\forall x (\neg \exists y (ON(y, x)) \rightarrow CLEAR(x))$$

So…how do we use theorem-proving techniques to construct plans?

Green's Method

- Add state variables to the predicates, and use a function `DO` that maps actions and states into new states

$$DO: A \times S \rightarrow S$$

- Example:

`DO(UNSTACK(x,y),s)` is a new state

UNSTACK

- So to characterize the action UNSTACK we could write:

$$\begin{aligned}&CLEAR(x,s)\wedge ON(x,y,s)\rightarrow\\ &\quad (HOLDING(x,DO(UNSTACK(x,y),s))\wedge\\ &\quad\ CLEAR(y,DO(UNSTACK(x,y),s)))\end{aligned}$$

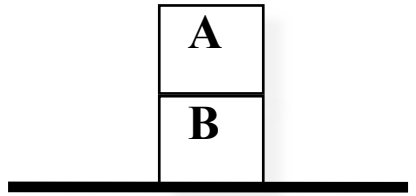
- We can prove that if S_0 is

$$ON(A,B,S_0)\wedge ONTABLE(B,S_0)\wedge CLEAR(A,S_0)$$ then

$$HOLDING(A,DO(UNSTACK(A,B),S_0))\wedge CLEAR(B,DO(UNSTACK(A,B),S_0))$$

S_1

S_1



More Proving

- The proof could proceed further; if we characterize PUTDOWN:

$$\mathrm{HOLDING}(x,s) \rightarrow \mathrm{ONTABLE}(x,\mathrm{DO}(\mathrm{PUTDOWN}(x),s))$$

- Then we could prove:

$$\mathrm{ONTABLE}(A,\ \mathrm{DO}(\mathrm{PUTDOWN}(A),\ \mathrm{DO}(\mathrm{UNSTACK}(A,B),\ S0)))$$

$$\frac{\dfrac{}{s_1}}{s_2}$$

- The nested actions in this constructive proof give you the plan:
  1. UNSTACK(A,B); 2. PUTDOWN(A)

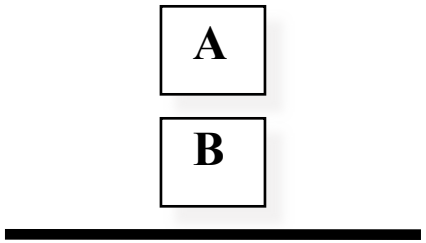
More Proving

- So if we have in our database:

$$ON(A,B,S_0) \wedge ONTABLE(B,S_0) \wedge CLEAR(A,S_0)$$

and our goal is

$$\exists s: ONTABLE(A,s)$$



we could use theorem proving to find the plan

But could I prove:

```
ONTABLE(B,
        DO(PUTDOWN(A),
           DO(UNSTACK(A,B),S0)))
```

?

The Frame Problem

- How do you determine *what changes* and *what doesn't change* when an action is performed?
- One solution: "Frame axioms" that specify how predicates can remain unchanged after an action
- Examples:

$$ONTABLE(z,s) \rightarrow ONTABLE(z,DO(UNSTACK(x,y),s))$$

$$ON(m,n,s) \wedge DIFF(m,x) \rightarrow ON(m,n,DO(UNSTACK(x,y),s))$$

Frame Axioms

- Problem: Unless we go to a higher-order logic, Green's method forces us to write many frame axioms
- Example:

$$COLOR(x,c,s) \rightarrow COLOR(x,c,DO(UNSTACK(y,z),s))$$

- We want to avoid this…other approaches are needed…

Laboratory on planning

- We will do more on planning
  - PDDL (planning domain definition language)
  - Practical exercise in groups

AGENT0 and PLACA

- Much of the interest in agents from the AI community has arisen from Shoham's notion of *agent oriented programming* (AOP) -1990
- AOP a 'new programming paradigm, based on a societal view of computation'
- The key idea that informs AOP is that of 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 the intentional stance to program machines

AGENT0

- 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
- Results only reported on first two components
- Relationship between logic and programming language is *semantics*
- We will skip over the logic(!), and consider the first AOP language, AGENT0

AGENT0

- AGENT0 is implemented as an extension to LISP
  - One of the oldest high-level programming language, favored programming language for artificial intelligence
- 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 (how commitments are added over time)
- The key component, which determines how the agent acts, is the commitment rule set

AGENT0

- Each commitment rule contains
  - a message condition
  - a mental condition
  - an action
- 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)

AGENT0

- Actions may be
  - Private: an internally executed computation, or
  - Communicative: sending messages
- Messages are constrained to be one of three types:
  - request to commit to action
  - un-request to refrain from actions
  - inform which pass on information
  - Requests and un-requests usually change the addressee's commitments; informs change the addressee's beliefs.

AGENT0

- A commitment rule:

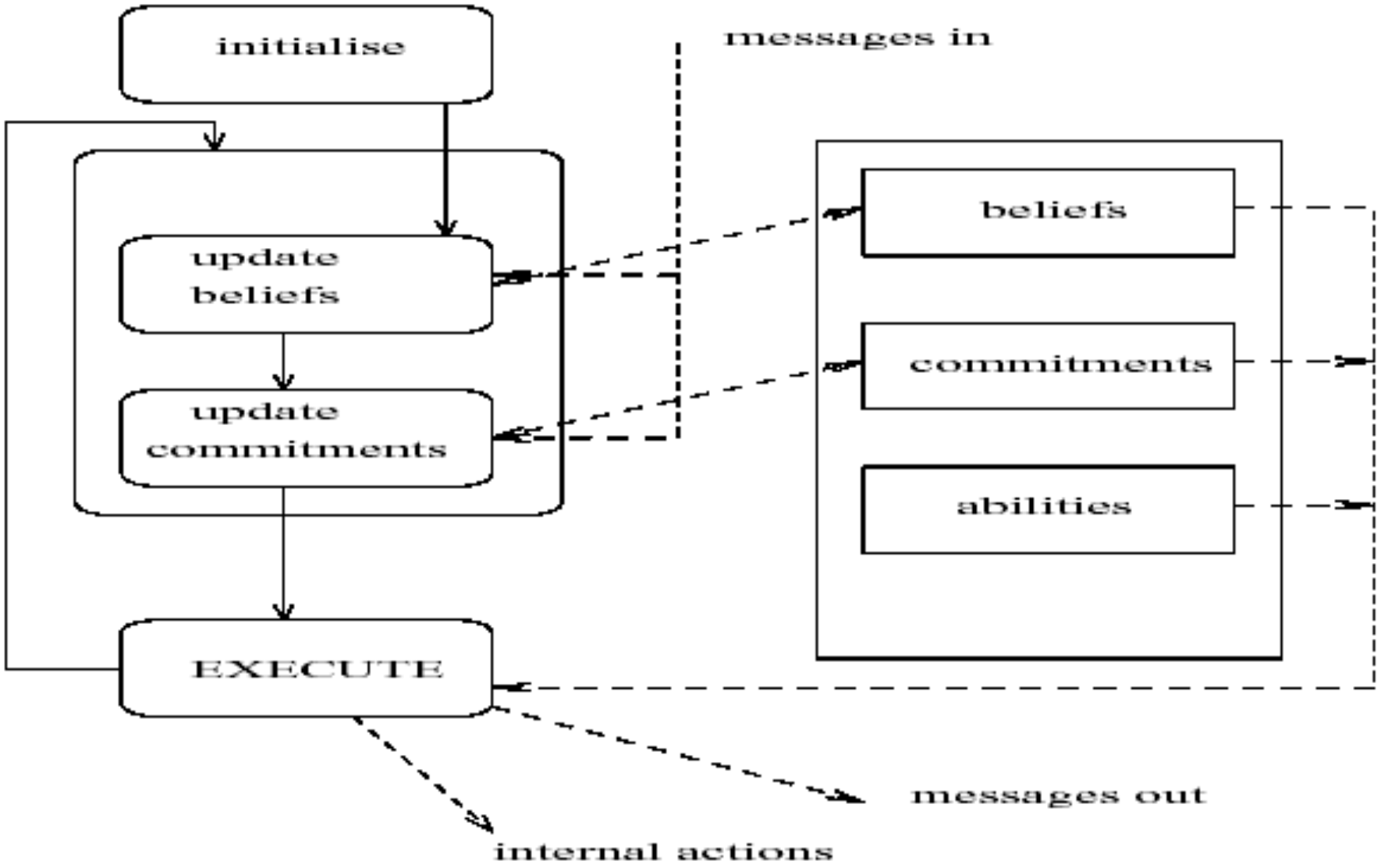
```
COMMIT(
     (agent, REQUEST, DO(time,action)), ;;; msg condition
     (B,
        [now, Friend agent] AND
        CAN(self, action) AND
         NOT [time, CMT(self, any-action)]
     ), ;;; mental condition
     self,
     DO(time, action))
```

- 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`**

AGENT0 control loop

1. Read all current messages, updating beliefs – and hence commitments – where necessary
2. Execute all commitments for the current cycle where the capability condition of the associated action is satisfied.
3. Goto 1.

AGENT0


initialise
messages in
update beliefs
update commitments
EXECUTE
beliefs
commitments
abilities
messages out
internal actions

AGENT0 and PLACA

- AGENT0 provides support for multiple agents to cooperate and communicate, and provides basic provision for debugging…
- …it was, 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 (**PLACA** - Planning Communicating Agents)
- **PLACA** language was intended to address one severe drawback to AGENT0:
  - the inability of agents to plan, and 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

AGENT0 and PLACA

An example mental change rule:

```
(((self ?agent REQUEST (?t (printed ?x)))
(AND (CAN-ACHIEVE (?t printed ?x)))
     (NOT (BEL (*now* shelving)))
     (NOT (BEL (*now* (vip ?agent))))
     ((ADOPT (INTEND (5pm (printed ?x)))))
     ((?agent self INFORM
          (*now* (INTEND (5pm (printed ?x))))))))
```

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

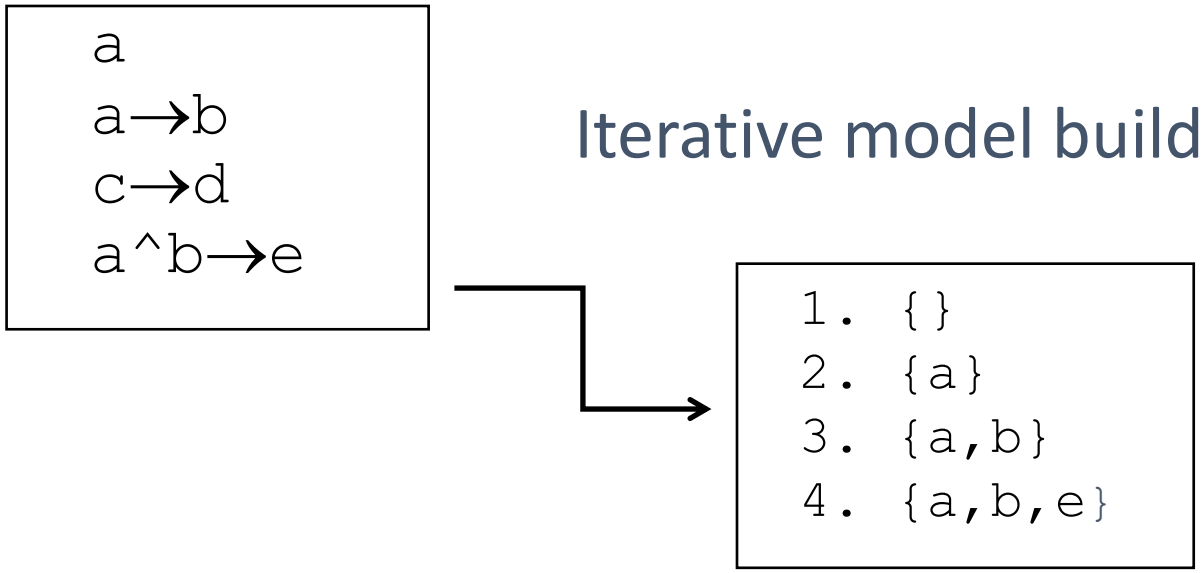
- adopt the intention to print it by 5pm, and
- inform them of your newly adopted intention

Concurrent MetateM

- An agent programming language based on direct execution of logical formulas
  - Close to the idea of agent as a theorem prover
- Several agents execute concurrently
- Communication by asynchronous broadcast message passing
- 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
- Execution corresponds to the iterative building of a model of the specification
- The agent execution procedure is correct
  - if there exists a model of the specification, the agent will build it

Iterative model bulding

- Common idea in computational logic: build a model for a set of formulas by
  - starting from an empty *interpretation*
  - at each step, extending the interpretation to satisfy the formulas that are false
- If formulas are implications, this process can be seen as “firing” of rules



Agent architecture

A MetateM agent is composed of:

- an agent interface, which defines how the agent interacts with its environment (i.e., other agents); and
- a computational engine, which defines how the agent acts, based on the MetateM paradigm of executable temporal logic

- Temporal logic is classical logic augmented by modal operators for describing how the truth of propositions changes over time

Agent interface

An agent interface is composed of

- an agent identifier*:* unique name for the agent;
- environment propositions*:* a set of symbols defining which messages will be accepted by the agent; and
- component propositions*:* a set of symbols defining the messages that the agent may send

For example, a 'stack' object's interface:

```
stack(pop,push)[popped,stackfull]
```

`{pop,push}` = environment prop.
`{popped,stackfull}` = component prop.

- 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

Agent specification

- Uses Propositional MetateM Logic (PML): propositional logic augmented with temporal operators
- The specification is given by means of PML formulas
- A program rule is a PML formula of the form
  `antecedent about past →`
  `consequent about present and future`
- Whenever the agent detects that the antecedent is true, it tries to make the consequent true too (iterative model building).
- “Declarative past, imperative future” [Gabbay, 1987].

Temporal connectives

Operator	Meaning
$\bigcirc\varphi$	φ is true tomorrow
$\mathbf{O}\varphi$	φ was true yesterday
$\Diamond\varphi$	at some time in the future, φ
$\Box\varphi$	always in the future, φ
$\diamondsuit\varphi$	at some time in the past, φ
$\blacksquare\varphi$	always in the past, φ
$\varphi\mathcal{U}\psi$	φ will be true until ψ
$\varphi\mathcal{S}\psi$	φ has been true since ψ
$\varphi\mathcal{W}\psi$	φ is true unless ψ
$\varphi\mathcal{Z}\psi$	φ is true zince φ

Temporal connective meanings

In the present state,

- $\bigcirc\varphi$ (resp. $\mathbf{O}\varphi$) is true iff φ is true in the next (resp. previous) state.
- $\Diamond\varphi$ (resp. $\Diamond\!\!\cdot\varphi$) is true iff φ is true in *some* future (resp. past) state. *including* the present state.
- $\Box\varphi$ (resp. $\blacksquare\varphi$) is true iff φ is true in *all* future (resp. past) states, *including* the present state.
- $\varphi\mathcal{U}\psi$ (resp. $\varphi\mathcal{S}\psi$) is true iff ψ is true in some future (resp. past) state, and φ is true until (resp. since) then.
- $\varphi\mathcal{W}\psi$ (resp. $\varphi\mathcal{Z}\psi$) is true iff φ is true in all future (resp. past) states until (resp. since) ψ is true, which may never happen.

Examples

❒ important(agents)

Agents are important now, and they will always be

◇important(Janine)

Sometime in the future, Janine will be important

¬friends(us)U apologize(you)

We are not friends until you apologize

○ apologize(you)

Tomorrow (in the next state) you shall apologize

Concurrent MetateM control loop

1. Update the agent's history by receiving messages (those that match with the agent's environment propositions)
2. Check which rules fire, comparing each rule's antecedent with the current history
3. Jointly execute the fired rules with commitments from previous cycles (see next slide)
4. goto 1

Execution of fired rules and commitments

- Collect the consequents of newly fired rules with old commitments: these become the new *constraints*
- Try to create the next state satisfying such constraints
  - it may not be possible to satisfy all constraints creating the next states; in this case, unsatisfied commitments are carried over to the next state;
  - current constraints are a *disjunctive* formula: the agent must choose among a number of execution possibilities
- Disjunctive constraints
  - $\Diamond a$ is satisfied by making a true in any of the future states;
  - $\neg\Box a$ is satisfied by making a false in all the future states.

Communication

- When a proposition becomes true, it is checked against the agent's component propositions: if it matches, it is broadcast as a message.
- On receipt of a message, each agent check it against its environment propositions: if it matches, the agent adds it to its history.

Sample Concurrent MetateM specification

- Simple resource allocation scenario
- Three agents:
  - *rp,* resource producer
  - *rc1* and *rc2,* resource consumers

Sample Concurrent MetateM specification

- Producer
  - Receives *ask1,ask2*
  - Sends *give1,give2*
  - Can only give a resource to an agent at a time, and will eventually give the resource to an agent that asks for it
- *rp specification*

$$
\begin{aligned}
&rp(ask1, ask2)[give1, give2]: \\
&\qquad \bullet ask1 \quad \rightarrow \quad \Diamond give1 \\
&\qquad \bullet ask2 \quad \rightarrow \quad \Diamond give2 \\
&\qquad \mathrm{start} \quad \rightarrow \quad \Box\neg(give1 \wedge give2)
\end{aligned}
$$

Sample Concurrent MetateM specification

- First consumer
  - Receives *give1*
  - Sends *ask1*
  - Asks for the resource at every cycle
- rc1 specification

rc1(*give1*)[*ask1*]:

start → *ask1*

○*ask1* → *ask1*

Sample Concurrent MetateM specification

- Second consumer
  - Receives *ask1, give2*
  - Sends *ask2*
  - Asks for the resource at each cycle following one where *rc1* asked for the resource and *rc2* did not
- rc2 specification

$$rc2(ask1, give2)[ask2]:$$
$$\bigcirc(ask1 \wedge \neg ask2) \quad \rightarrow \quad ask2$$

Sample run

Time	Agent		
	rp	*rc1*	*rc2*
0		*ask1*	
1	*ask1*	*ask1*	*ask2*
2	*ask1,ask2,give1*	*ask1*	
3	*ask1,give2*	*ask1,give1*	*ask2*
4	*ask1,ask2,give1*	*ask1*	*give2*
5	...	...	...

Snow white example

- Toy example
- Exemplifies agent characterization in resource allocation scenaria by means of logical relations over intentional notions.
- Agents:
  - Snow White
  - The modified seven dwarfs: Eager, Mimic, Jealous, Greedy, Courteous, Generous, Shy
- Messages:
  - *ask(x):* dwarf *x* asks Snow White for a sweet;
  - *give(x):* Snow White gives dwarf *x* a sweet.

(note variables!)

Eager and Mimic

- Eager initially asks for a sweet and, from then on, whenever he receives a sweet, asks for another

$$
\begin{array}{rcl}
\textit{eager}(\textit{give})[\textit{ask}]: & & \\
\texttt{start} & \rightarrow & \textit{ask}(\textit{eager}) \\
\bullet\!\!\rightarrow \textit{give}(\textit{eager}) & \rightarrow & \textit{ask}(\textit{eager})
\end{array}
$$

- Mimic asks for a sweet whenever he sees eager asking for one

$$
\begin{array}{rcl}
\textit{mimic}(\textit{ask})[\textit{ask}]: & & \\
\bullet\!\!\rightarrow \textit{ask}(\textit{eager}) & \rightarrow & \textit{ask}(\textit{mimic})
\end{array}
$$

Jealous and greedy

- Jealous asks for a sweet whenever he sees eager receiving one

$$jealous(give)[ask]: \quad \bullet give(eager) \rightarrow ask(jealous)$$

- Greedy asks for a sweet as often as he can

$$greedy()[ask]: \quad start \rightarrow \Box ask(greedy)$$

Courteous

- Courteous asks for a sweet only when eager, mimic, jealous and greedy have all asked for one, since courteous last asked

$$
\begin{aligned}
&courteous(ask)[ask]: \\
&((\neg ask(courteous))\mathcal{S} ask(eager)\ \wedge \\
&(\neg ask(courteous))\mathcal{S} ask(mimic)\ \wedge \\
&(\neg ask(courteous))\mathcal{S} ask(jealous)\ \wedge \\
&(\neg ask(courteous))\mathcal{S} ask(greedy)) \quad \rightarrow \quad ask(courteous)
\end{aligned}
$$

Generous

- Generous asks for a sweet only when eager, mimic, jealous, and courteous have all received one, since generous last asked for one

$$
\begin{aligned}
&\textit{generous}(\textit{give})[\textit{ask}]: \\
&((\neg \textit{ask}(\textit{generous}))\mathcal{S}\textit{give}(\textit{eager}) \ \wedge \\
&(\neg \textit{ask}(\textit{generous}))\mathcal{S}\textit{give}(\textit{mimic}) \ \wedge \\
&(\neg \textit{ask}(\textit{generous}))\mathcal{S}\textit{give}(\textit{jealous}) \ \wedge \\
&(\neg \textit{ask}(\textit{generous}))\mathcal{S}\textit{give}(\textit{greedy}) \ \wedge \\
&(\neg \textit{ask}(\textit{generous}))\mathcal{S}\textit{give}(\textit{courteous})) \ \rightarrow \ \textit{ask}(\textit{generous})
\end{aligned}
$$

Shy

- Shy only asks for a sweet when he sees no one else asking

$$
\begin{array}{l}
shy(ask)[ask]: \\
\qquad \texttt{start} \quad \rightarrow \quad \Diamond ask(shy) \\
\qquad \bullet ask(x) \quad \rightarrow \quad \neg ask(shy) \\
\qquad \bullet ask(shy) \quad \rightarrow \quad \Diamond ask(shy)
\end{array}
$$

Snow White

- Snow White has some sweets (resources), which she will give to the dwarfs (resource consumers)
- She will only give to one dwarf at a time
- She will always eventually give to a dwarf that asks

$$
\begin{aligned}
&\mathit{snowwhite}(\mathit{ask})[\mathit{give}]: \\
&\qquad \bullet\!\!\circ \mathit{ask}(x) \quad \rightarrow \quad \Diamond \mathit{give}(x) \\
&\mathit{give}(x) \wedge \mathit{give}(y) \quad \rightarrow \quad x = y
\end{aligned}
$$

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!