Reasoning about Complex Actions with Incomplete Knowledge: a Modal Approach

Matteo Baldoni

Thanks to: Cristina Baroglio, Laura Giordano, Alberto Martelli, Viviana Patti

Dipartimento di Informatica, Università di Torino

Outline

- Introduction and motivations: programming agents
- The modal action logic
- Proof Procedure: finding correct plans
- An application to adaptive web sites
- Conclusion
Introduction and Motivations: Programming Agents

Agent programming language

- A logical framework for reasoning about actions in a logic programming setting
- Aim: based on this framework, dealing with agent programming
- In this context we need to describe the behaviour of an agent that
  * chooses a course of actions conditioned on its beliefs on the environment
  * uses sensors for acquiring or updating its knowledge about the real world

Goal: closing all doors

? Which actions should I perform?

Reasoning about actions and planning!
Reasoning with incomplete knowledge

- In presence of **incomplete knowledge** on environment the agent can acquire information by means of **sensing actions**.

Since the robot does not see in advance the outcome of its sensing actions, it has to be prepared for any possible outcome.

**Goal:** closing all doors

- Which actions should I perform?
- Is door #1 open?
In order to know it sense the door!

A modal approach

- **atomic actions**: sensing actions and world actions
- each atomic (world or sensing) action is represented by a modality. Modal logics allows a very natural representation of actions as state transitions through the accessibility relation of Kripke structures.
- intentional attitudes (beliefs, knowledge...) are usually represented as modalities

---

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin_front_off(door2)</td>
<td>Bin_front_off(door2)</td>
</tr>
<tr>
<td>B¬in_front_off(door1)</td>
<td>B¬in_front_off(door1)</td>
</tr>
<tr>
<td>Bopen(door2)</td>
<td>Bopen(door2)</td>
</tr>
</tbody>
</table>

Epistemic state: a set of epistemic fluents

- complex actions are defined through (possibly recursive) procedure definitions, given by means of prolog-like clauses of our modal framework
The frame problem

- The frame problem: specifying in a general way the fluents which remain unaffected by the execution of a given action.

\[ S1 \]
\[ \neg \text{in_front_of}(doo}r2) \]
\[ \neg \text{open}(door2) \]
\[ \neg \text{in_front_of}(door1) \]
\[ \text{Bin_front_of}(door2) \]
\[ \text{Open}(door2) \]
\[ \text{toggle_switch}(door2) \]

\[ S2 \]
\[ \text{Bin_front_of}(door2) \]
\[ \text{in_front_of}(door1) \]
\[ \text{Open}(door2) \]
\[ \text{toggle_switch}(door2) \]
\[ \text{in_front_of}(door1) \]
\[ \text{open}(door2) \]

- Non monotonic solution by using persistency assumptions in an abductive framework: if a fluent \( F \) holds in \( S1 \) and it is consistent to assume that it will hold after the execution of \( \text{toggle_switch}(door2) \), then we conclude that it holds in \( S2 \).
**Action Theory**

* For each $a \in A$, $S \rightarrow [a]$ (epistemic operator: $b$)

- **Epistemic state $s$:** complete and consistent set of epistemic fluent literals. Consistency is guaranteed by seriality of $B$: $B \supset \neg B \neg l$
- It provides a three-valued interpretation of literals; a literal can be:
  - true, when $Bl \in s$
  - false, when $B\neg l \in s$
  - undefined, when both $\neg Bl$ and $B\neg l$ are in $s$ (denoted by $UI$)

**Simple action clauses:**
- **Action laws:** define direct effects of actions on an epistemic fluent (conditional effects can be represented)
  $$\\Box(Bl \wedge \ldots \wedge Bl_n \supset [a]Bl_0)$$
  $$\Box(Ml_1 \wedge \ldots \wedge Ml_n \supset [a]Ml_0)$$

- **Precondition laws:** allows to specify knowledge precondition on the executability of actions
  $$\Box(Bl_1 \wedge \ldots \wedge Bl_n \supset \langle a \rangle true)$$

- **Sensing axioms:** we represent a binary sensing action $s$ for knowing whether the literal $l$ or its complement is true, by means of axioms of our logic:
  $$[s]\varphi \equiv [s^{Bl} \cup s^{B\neg l}]\varphi$$
  $$\Box(Bl_1 \wedge \ldots \wedge Bl_n \supset \langle s^{Bl} \rangle true)$$
  $$\Box(true \supset [s^{Bl}]Bl)$$
  $$\Box(Bl_1 \wedge \ldots \wedge Bl_n \supset \langle s^{B\neg l} \rangle true)$$
  $$\Box(true \supset [s^{B\neg l}]B\neg l)$$
**Example**

(a) \( \Box (B \text{-open}(I) \supset [\text{toggle.switch}(I)]B\text{-open}(I)) \)
(b) \( \Box (M \text{-open}(I) \supset [\text{toggle.switch}(I)]M\text{-open}(I)) \)
(c) \( \Box (M\text{-open}(I) \supset [\text{toggle.switch}(I)]B\text{-open}(I)) \)
(d) \( \Box (M\text{-open}(I) \supset [\text{toggle.switch}(I)]M\text{-open}(I)) \)

(e) \( \Box (\text{Bin.front.of}(I) \supset \langle \text{toggle.switch}(I) \rangle \text{true}) \)

(f) \([\text{sense.door}(I)]\varphi \equiv [\text{sense.door}(I)B\text{-open}(I) \cup \text{sense.door}(I)B\text{-open}(I)]\varphi\)

where the primitive actions \( \text{sense.door}(I)B\text{-open}(I) \) and \( \text{sense.door}(I)B\text{-open}(I) \) are ruled by the set of laws:

(g) \( \Box (\text{Bin.front.of}(I) \supset \langle \text{sense.door}(I)B\text{-open}(I) \rangle \text{true}) \)
(h) \( \Box (\text{true} \supset \langle \text{sense.door}(I)B\text{-open}(I) \rangle \text{true}) \)
(i) \( \Box (\text{Bin.front.of}(I) \supset \langle \text{sense.door}(I)B\text{-open}(I) \rangle \text{true}) \)
(j) \( \Box (\text{true} \supset \langle \text{sense.door}(I)B\text{-open}(I) \rangle \text{true}) \)

**Action Theory**

- **Procedure definitions**: we express complex actions’ definitions by means of axioms of the form

  \[ \langle p_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle \varphi. \]

  If \( p_0 \) is a procedure name and the \( p_i \), \( i=1,\ldots,n \) are procedure names or atomic or test actions (\( \langle \psi? \rangle \)), the axiom can be interpreted as a procedure definition that can be executed in a goal-directed way, similarly to standard logic programs.

- **Procedure definitions**: Prolog-like vs Algol-like
  [GOLOG, Reiter et al., 1994]

- **These axioms has the form of inclusion axioms**
  [Baldoni et al., Tableaux 1998, LNAI 1397]
We are interested in the class of *inclusion multimodal logics*

[Fariñas del Cerro and Penttonen, 1988]

They are characterized by a set of logical axioms of the form

\[ [t_1][t_2]...[t_n] \varphi \supset [s_1][s_2]...[s_m] \varphi \quad (n > 0, m \geq 0) \]

Motivations:
- non-homogeneous
- interaction axioms
- they have interesting computational properties

---

**Inclusion Modal Logics: examples**

- reflexivity
- transitivity
- inclusion
- mutual trans.
- persistency
- seriality
- symmetry
- euclideanness

\[ [t_1][t_2]...[t_n] \varphi \supset [s_1][s_2]...[s_m] \varphi \]
Inclusion Modal Logics: possible-worlds semantics

\[ \langle W, \{R_i | i \in MOD\}, V \rangle \]

- W is a set of "worlds";
- the \( R_i \)'s are the accessibility relations, one for each modality;
- V is a valuation function.

\( R_1 \circ R_2 \circ \ldots \circ R_n \supseteq R_1 \circ R_2 \circ \ldots \circ R_n \)

Example

- **Domain Description**: a pair \( (\Pi, s_0) \);
- \( \Pi \) is a tuple having as elements a set of simple action clauses for world actions, a set of axioms for sensing actions, a set of procedure axioms.

(t) \( \langle all.door.closed \rangle \varphi \subset \langle close.door(door1) \rangle \langle close.door(door2) \rangle \varphi \).

(u) \( \langle all.door.closed \rangle \varphi \subset \langle close.door(door2) \rangle \langle close.door(door1) \rangle \varphi \).

(k) \( \langle close.door(I) \rangle \varphi \subset \langle B-open(I) \rangle \varphi \)

(l) \( \langle close.door(I) \rangle \varphi \subset \langle (B-open(I) \land Bin.front.of(I)) \rangle \langle toggle.switch \rangle \varphi \)

(m) \( \langle close.door(I) \rangle \varphi \subset \langle (B-open(I) \land Bin.front.of(I)) \rangle \langle sense.door(I) \rangle \langle close.door(I) \rangle \varphi \)

(n) \( \langle close.door(I) \rangle \varphi \subset \langle (M-open(I) \land B-in.front.of(I)) \rangle \langle go.to.door(I) \rangle \langle close.door(I) \rangle \varphi \)
### Reasoning about actions

- The **planning problem** in general...

  Given an **initial state** and a **goal Fs**, is there a **sequence of actions** that, when executed in the initial state, leads to a state in which **Fs** holds?

### The planning problem

**In our framework...**

Is there a possible execution of \( p_1; \ldots; p_n \) leading to a state in which some condition **Fs** holds?

The procedure definition constrain the search space of reachable states in which to search for the wanted sequence

<table>
<thead>
<tr>
<th>query</th>
<th>(&lt;p_1&gt;\ldots&lt;p_n&gt;Fs \ (n \geq 0))</th>
</tr>
</thead>
<tbody>
<tr>
<td>answer</td>
<td>(a_1, a_2, \ldots, a_m)</td>
</tr>
</tbody>
</table>

If yes, the answer is an **execution trace**, a **primitive action sequence** from the initial state to the final one, which represents a linear plan.
Proof Procedure:
finding correct plans

The proof procedure

The rules reduce the complex actions in the query to a sequence of primitive and test actions; they verify if the primitive actions are possible and if the test actions are successful...until...

\[
\begin{align*}
&\frac{a_1, \ldots, a_m \vdash_{pa} \left(p_1; \ldots; p_n, p_0\right) F x \ w. \ a. \ \sigma}{a_1, \ldots, a_m \vdash_{pa} (p; p_2; \ldots; p_n) F x \ w. \ a. \ \sigma} \\
&\frac{\frac{a_1, \ldots, a_m \vdash_{pa} F x' \ a_1, \ldots, a_m \vdash_{pa} (p_2; \ldots; p_n) F x \ w. \ a. \ \sigma}{a_1, \ldots, a_m \vdash_{pa} \left(p; F x'; p_2; \ldots; p_n\right) F x \ w. \ a. \ \sigma}}{a_1, \ldots, a_m \vdash_{pa} \left(p; p_2; \ldots; p_n, F x\right) F x \ w. \ a. \ \sigma} \\
&\frac{\frac{a_1, \ldots, a_m \vdash_{pa} F x' \ w. \ a. \ \sigma}{a_1, \ldots, a_m \vdash_{pa} \left(p; F x'; p_2; \ldots; p_n\right) F x \ w. \ a. \ \sigma}}{a_1, \ldots, a_m \vdash_{pa} \left(p; p_2; \ldots; p_n, F x\right) F x \ w. \ a. \ \sigma} \\
&\frac{\frac{a_1, \ldots, a_m \vdash_{pa} F x' \ w. \ a. \ \sigma}{a_1, \ldots, a_m \vdash_{pa} \left(p; F x'; p_2; \ldots; p_n\right) F x \ w. \ a. \ \sigma}}{a_1, \ldots, a_m \vdash_{pa} \left(p; p_2; \ldots; p_n, F x\right) F x \ w. \ a. \ \sigma} \\
&\frac{\frac{\frac{a_1, \ldots, a_m \vdash_{pa} F x' \ w. \ a. \ \sigma}{a_1, \ldots, a_m \vdash_{pa} \left(p; F x'; p_2; \ldots; p_n\right) F x \ w. \ a. \ \sigma}}{a_1, \ldots, a_m \vdash_{pa} \left(p; p_2; \ldots; p_n, F x\right) F x \ w. \ a. \ \sigma}}{a_1, \ldots, a_m \vdash_{pa} \left(p; p_2; \ldots; p_n, F x\right) F x \ w. \ a. \ \sigma}
\end{align*}
\]

where \( p \in \mathcal{P} \) and \( (\psi) \psi \in \{p_1'; \ldots; p_n'; \psi\} \in H_{\mathcal{P}} \) and \( a \in \mathcal{A} \) and \( \Box(F x' \supset (a)(true)) \in \mathcal{H}_{\mathcal{A}} \) and \( s \in \mathcal{S} \) and \( t \in dom(s) \) and \( \sigma = a_1; \ldots; a_m \)

Fig. 2. The derivation relation \( \vdash_{pa} \).
The proof procedure

The sequence of primitive actions to be executed - \( a_1, \ldots, a_n \) - has been already determined: the rules allow to check if the fluent conjunction \( F_s \) is true after \( a_1, \ldots, a_n \).

\[
\begin{align*}
6) & \quad \frac{a_1, \ldots, a_m \vdash \text{true}}{a_1, \ldots, a_{m-1} \vdash F_s} \\
7a) & \quad \frac{a_1, \ldots, a_{m-1} \vdash F_s}{a_1, \ldots, a_m \vdash F} \quad \text{where } m > 0 \text{ and } \Delta(F_s) \supset [a_m]F \in \Pi_d \\\n7b) & \quad \frac{a_1, \ldots, a_m \vdash \overline{F}}{a_1, \ldots, a_m \vdash F} \quad \text{where } a_m = s^r \\
7c) & \quad \frac{a_1, \ldots, a_m \vdash \overline{F}}{a_1, \ldots, a_m \vdash \text{false}} \quad \text{where } m > 0 \\
7d) & \quad \frac{\varepsilon \vdash \overline{F}}{a_1, \ldots, a_m \vdash \overline{F}} \quad \text{where } F \in S_0
\end{align*}
\]

Generating conditional plans

We introduce a proof procedure that constructs a conditional plan, which achieves the goal for all the possible outcomes of sensing.

Instead of making assumptions on possible outcomes of sensing... the computation splits in more branches...
Soundness and answer’s properties

Soundness of the proof procedure w.r.t. the unique acceptable solution:

Theorem 1. Let \((H, S_0)\) be an e-consistent dynamic domain description and let \((p_1; p_2; \ldots; p_n)Fs\) be a query. Let \(\Delta\) be the unique abductive solution for \((H, S_0)\). If \((p_1; p_2; \ldots; p_n)Fs\) succeeds from \((H, S_0)\) with answer \(\sigma\), then \(\Sigma_{(H, S_0) \cup \Delta} \models (p_1; p_2; \ldots; p_n)Fs\).

Answer’s properties:

Proposition 2. Let \((H, S_0)\) be an e-consistent dynamic domain description and let \((p_1; p_2; \ldots; p_n)Fs\) be a query. Let \(\Delta\) be the unique abductive solution for \((H, S_0)\). If \(\epsilon \models_{pp} (p_1; p_2; \ldots; p_n)Fs\) with answer \(\sigma\), then:

(a) \(\Sigma_{(H, S_0) \cup \Delta} \models (\epsilon)Fs \supseteq (p_1; p_2; \ldots; p_n)Fs\);
(b) \(\Sigma_{(H, S_0) \cup \Delta} \models \sigma Fs\);
(c) \(\Sigma_{(H, S_0) \cup \Delta} \models (\epsilon)Fs \supseteq [\sigma]Fs\)

A more complex example: the air conditioning problem!

2 luglio 2002 Corso di Agenti Intelligenti per il Dottorato di Informatica
The air conditioning problem

GOAL: to set high both the fan speed of the air conditioning units of the room

Atomic actions

:- use_module(library(dyLog)).

- go_to_the_unit(X) possible_if ~out_room.
- go_to_the_unit(X) causes in_front_of(X) if true.
- go_to_the_unit(X) causes ~in_front_of(Y) if ?in_front_of(Y) & +(X \neq Y).

- flow(X,State) domain [X in [unit1,unit2], State in [off,low,high]].

- turn_dial(X) possible_if ?in_front_of(X) & ?cover_up(X).
- turn_dial(X) causes flow(X,off) if ?flow(X,high).
- turn_dial(X) causes flow(X,low) if ?flow(X,off).
- turn_dial(X) causes flow(X,high) if ?flow(X,low).
- turn_dial(X) causes ~flow(X,State) if ?flow(X,State).

- flow(X,off) if ?flow(X,high).
- flow(X,low) if ?flow(X,high).
- flow(X,off) if ?flow(X,low).
- flow(X,high) if ?flow(X,low).
- flow(X,low) if ?flow(X,off).
- flow(X,high) if ?flow(X,off).
[...]

2 luglio 2002 Corso di Agenti Intelligenti per il Dottorato di Informatica 25
Sensing actions

cover_up(X) domain (X in [unit1, unit2]).

open_protective_cover(X) possible_if ?in_front_of(X).

open_protective_cover(X) causes cover_up(X) if ?true.

check_dial(X) possible_if ?in_front_of(X) & ?cover_up(X).

check_dial(X) senses flow(X, State).

check_cover(X) possible_if ?in_front_of(X).

check_cover(X) senses cover_up(X).

Complex action (agent behaviour)

set_the_fan_speed_high(X) isp ?flow(X, high).
set_the_fan_speed_high(X) isp ?(~flow(X, high)) & ?(~cover_up(X)) & ?(u(flow(X, high))) & open_protective_cover(X) & set_the_fan_speed_high(X).

set_the_fan_speed_high(X) isp ?(u(cover_up(X))) & ?(u(flow(X, high))) & check_cover(X) & set_the_fan_speed_high(X).

set_the_fan_speed_high(X) isp ?(~cover_up(X)) & ?(~flow(X, high)) & open_protective_cover(X) & set_the_fan_speed_high(X).

set_the_fan_speed_high(X) isp ?(~flow(X, high)) & ?(~in_front_of(X)) & go_to_the_unit(X) & set_the_fan_speed_high(X).

set_the_fan_speed_high(X) isp ?(u(flow(X, high))) & check_dial(X) & set_the_fan_speed_high(X).

set_the_fan_speed_high(X) isp ?(u(flow(X, high))) & ?(~in_front_of(X)) & go_to_the_unit(X) & set_the_fan_speed_high(X).

all_unit_high isp set_the_fan_speed_high(unit1) & set_the_fan_speed_high(unit2).

all_unit_high isp set_the_fan_speed_high(unit2) & set_the_fan_speed_high(unit1).
Extracting a plan

?- plan(?flow(high,unit1) & ?flow(high,unit2) after all_units_high, S)

An Application to Adaptive Web Sites
Implementation and Applications

- DyLOG have been implemented in Sicstus Prolog
  http://www.di.unito.it/~alice

- Baldoni et. al.
  PADL’01 [LNCS 1990], HCII’01, AI*IA’01 Demo session

  The aim of these works is to describe our approach on adaptation to the user in the web applications context.

  The approach is based on the use of our agent logic programming language in order to program a rational agent which exploits its planning capabilities for dynamically generating a web site, adapted to the current user’s needs;

  Current case study: supporting student in building personalized studiorum itinera

Curriculum sequencing problem

- We focus on study plan, that is ...
- ... a curriculum is a sequence of courses or learning modules that allows the learning goal to be achieved
- A learning goal expresses a body of coherent and consistent knowledge, e.g. expert about “bioinformatics”
Curriculum sequencing problem: course as “action”

- Each course can be attended with success only if some knowledge prerequisites are satisfied (precondition)
- Each course supplies a set of competence (effects), these effects could be unconditioned or conditioned to the knowledge of some others competences
- Competences are not courses!

Curriculum sequencing problem: competences are structured

- Competences are structured into a hierarchy of competences
- Initial student knowledge and learning goal are expressed by means of a set of competences
- Target: build curricula based on competences
A rational agent as tutor: adaptation as reasoning about action

- Adaptation as interaction between the user and a rational agent aimed at presenting and discussing study plan proposals
- Study plan proposals are obtained after a reasoning process applied to the domain description and the specific needs expressed by the user

Curriculum sequencing problem: interaction with the user

- Initial and final situations are specified by the user and they depend on user's knowledge, needs and goals
- The system should interact with the user in order to acquire this information
**Curriculum sequencing problem: interaction with the user**

- Many courses might supply the same competences, so many curricula might satisfy the learning goal.
- The system should plan interactions with the user in order to allow him/her to choose the preferred courses.

**DyLOG: a modal logic programming language to reason about actions**

- Tutor implemented in DyLOG.
- Based on a logic theory for reasoning about action and change in a modal logic programming setting:
  - *Primitive actions* - preconditions and effects
  - *Sensing and suggesting actions* - interacting with the user
  - *Procedure definitions* - the agent’s behaviour
The planning problem

Given an initial state and a goal $F_s$ (in our case: learning goal), is there a sequence of actions (courses) that, when executed in the initial state (student initial knowledge), leads to a state in which $F_s$ holds?

Planning problem in DyLOG

- Is there a possible execution of a procedure $p$ leading to a state in which some condition $F_s$ holds?
- **Answer**: an execution trace "$a_1, a_2, ..., a_m$", i.e. a primitive action sequence from the initial state to the final one, which represents a linear plan
- $p$ is the procedure that encodes the tutor behaviour
Planning problem as reasoning

Initial situation

\[ \text{curriculum} \]

Final situation

Initial student knowledge

Learning goal

Domain Theory

\[ \text{U} \]

Initial Knowledge

as a result of the reasoning process

\[ \text{course} \quad \text{course} \quad \ldots \quad \text{course} \quad \text{course} \]

Procedures can contain sensing and suggesting actions, whose outcomes are unknown at planning time.

All the possible alternatives are to be taken into account.

By applying the plan metapredicate to a procedure containing sensing and suggesting actions we obtain a conditional plan whose branches correspond to the possible outcomes of sensing.
Verification problem as reasoning

Initial situation

Curriculum

course, course, course, ... ; course, course, course

Initial student knowledge

Software agent

Final situation

Learning goal

Domain Theory

\[ U \models \langle \text{course}; \ldots; \text{course} \rangle \text{Learning goal} \]

Initial Knowledge

- In this case the sequence is given

DyLOG metapredicate “plan” for plan extraction

Domain Theory

\[ U \models \langle \text{procedure} \rangle \text{Learning goal} \]

Initial Knowledge

plan(procedure, Learning goal, Plan)

- "procedure" can be nondeterministic; the metapredicate plan will extract from it a plan, corresponding to a possible execution of the procedure, that starts from the current state and leads to a state in which the learning goal holds
The virtual tutor in DyLOG: main procedure

- The procedure `advice` extracts a plan that will be executed

The virtual tutor in DyLOG

- `add_course` is a primitive action
- `offer_course_on` is a suggesting action (it implies an interaction with the user)
An extracted plan: execution

Each path to a leaf is actually a study plan that will allow the student to acquire the desired competence.

Branches represent alternative courses to acquire a competence.

Execution of a conditional plan is aimed at helping the user to choose the preferred path.

A multiagent architecture: User, Executor and Reasoner

- The system, named WLog, has a multi-agent architecture: modular, flexible and scalable.
- The Executor is the interface between the user and the reasoner.
- Protocols: HTTP and FIPA like.
The WLog architecture

The reasoner

- The reasoner has been implemented in the logic programming language DyLOG
- DyLOG interpreter has been implemented in Sicstus Prolog (v3.8.4)
The executor

- The executor is the agent that *actually interacts* with the user
- It presents the pages planned by the reasoner and sends it the run-time user’s choices
- It is built by means of a set of *Java servlets*

The agent facilitator

- It maintains a list of subscribed agents together with the information on their services
- It offers a centralized "post office", i.e. a mail box for each subscribed agent
- Exchanged messages are defined following the FIPA standard
Conclusion

- Using reasoning about actions and in particular *temporal projection* to achieve adaptation in curriculum sequencing
  - extracting a study plan
  - verifying a study plan

- How to repair a wrong study plan? How to suggest modification? How to help the user to repair a submitted curriculum? We are working on these issues ...
Conclusion

- Extending the framework for dealing with communicative agents:
  - representing primitive communicative actions in the modal framework
  - specifying locally conversation schemas that the agents are expected to follow
  - based on the logical framework, planning dialogue acts and appropriate reply of the agent during a conversation

http://www.di.unito.it/~alice
(Advanced Logic In Computing Environment)

- The DyLOG interpreter
- Technical papers on the logical foundations of DyLOG
- Documentation on WLog system implementation
- This slide: http://www.di.unito.it/~baldoni/didattica