Procedures, actions and vision

Outline for today:

- Remarks about the second assignments series
- Extending reasoning about actions
  - Procedural programming (Golog)
  - Probability and utility
- Model-based interpretation and computer vision
- Vision Applications

2nd Assignments: first part

- We have seen:
  - Logic programming and resolution
  - Bayesian networks and uncertainty
  - Probabilistic logic and explanations
  - AILog language
- The first part of the assignment asks you to *model* a specific probabilistic domain in AILog and to interact with AILog to compute (conditional) probabilities in that domain. You can ask AILog to *predict* and *condition*.
- The domain is about *relational crime*.

(Recap) Three solutions

We have seen three core types of models for action and change in logical representations

- **State-operator models**: potentially needs all frame axioms, requiring $O(FA)$ axioms (F and A are numbers of fluents and actions).
- **STRIPS**: uses CWA + procedural semantics of actions. It only needs to specify $O(A)$ actions, but the rules can become long and tedious.
- **Situation calculus**: requires one axiom *per fluent* and promises that they tend to stay compact (total $O(AE)$ with E the number of effects).

Sitcalc does not represent the state explicitly, i.e. inferring the truth value of a fluent *Goal*$($*do*$(\mathbf{a}, S_0))$ requires to *reason all the way back to the initial situation*.

(Recap) States and Operators

A simple logical model: each state has the location of the bananas (b), the monkey (m) and the box (l), as well as whether the monkey is on the box (o, is `y` or `n`), and whether the monkey has the bananas h (`y` or `n`).

Idea: use Prolog *lists of atoms*, e.g. `[loc1, loc2, loc3, n, n]`
A goal state will (at least) have as last element in the list a '`y`'.

States and Operators:

```
initial_state([loc1, loc2, loc3, n, n]).
goal_state([_, _, _, _, y]).
legal_move([B, M, M, n, H], climb_on, [B, M, M, y, H]).
...
```

Note that the representation here is fixed in size and order.

(Recap) STRIPS

A STRIPS operator $\langle \textit{Act}, \textit{Pre}, \textit{Add}, \textit{Del} \rangle$ represents the effects of an action using add and delete lists, preceded by preconditions.
Let O be an operator and let S be a state, i.e. a set of ground relational atoms. The *operational semantics* of applying O to S is

- first find a *matching* of Pre and S, i.e. find a subset $S' \subseteq S$ and a substitution θ such that $\textit{Pre}\theta \equiv S'$
- compute the new state as $S'' = (S \backslash \textit{Del}\theta) \cup \textit{Add}\theta$.

Example: Let $O = \langle \textit{Go}(x, y)$
$\{\textit{At}(\textit{Monkey}, x), \textit{On}(\textit{Monkey}, \textit{Floor})\}$,
$\{\textit{At}(\textit{Monkey}, x)\}, \{\textit{At}(\textit{Monkey}, Y)\}\rangle$,
and $S = \{\textit{On}(\textit{Monkey}, \textit{Floor}), \textit{At}(\textit{Monkey}, \textit{Loc1}), \ldots, etc.\}$

Taking $\textit{Go}(\textit{Loc1}, \textit{Loc2})$ spawns the new state
$S' = \{\textit{On}(\textit{Monkey}, \textit{Floor}), \textit{At}(\textit{Monkey}, \textit{Loc2}), \ldots, etc.\}$

(Recap) Sitcalc effect axioms

Suppose there are two positive effect axioms for the fluent *Broken* :

$\mathit{Fragile}(x) \rightarrow \mathit{Broken}(x, \mathit{do}(\mathit{drop}(r, x), s))$

$\mathit{NextTo}(b, x, s) \rightarrow \mathit{Broken}(x, \mathit{do}(\mathit{explode}(b), s))$

These can be rewritten as

$\exists r\{a = \mathit{drop}(r, x) \land \mathit{Fragile}(x)\} \lor \exists b\{a = \mathit{explode}(b) \land \mathit{NextTo}(b, x, s)\} \rightarrow \mathit{Broken}(x, \mathit{do}(a, s))$

Similarly, consider the negative effect axiom:

$\neg \mathit{Broken}(x, \mathit{do}(\mathit{repair}(r, x), s))$

which can be rewritten as

$\exists r\{a = \mathit{repair}(r, x)\} \rightarrow \neg \mathit{Broken}(x, \mathit{do}(a, s))$ In general, for any fluent F, we can rewrite all the effect axioms as two formulas of the form:

$P_F(\mathbf{x}, a, s) \rightarrow F(\mathbf{x}, \mathit{do}(a, s))$ (1)

$N_F(\mathbf{x}, a, s) \rightarrow \neg F(\mathbf{x}, \mathit{do}(a, s))$ (2)

(both are formulas with free variables which are among x_i, a and s)

(Recap) Explanation closure

Now make a completeness assumption regarding these effect axioms: assume that (1) and (2) characterize *all* the conditions under which an action a changes the value of fluent F.

This can be formalized by explanation closure axioms:

$\neg F(\mathbf{x}, s) \wedge F(\mathbf{x}, \mathit{do}(a, s)) \rightarrow P_F(\mathbf{x}, a, s)$ (3)

if F was false and was made true by doing action a then condition P_F must have been true

$F(\mathbf{x}, s) \wedge \neg F(\mathbf{x}, \mathit{do}(a, s)) \rightarrow N_F(\mathbf{x}, a, s)$ (4)

if F was true and was made false by doing action a then condition N_F must have been true

These explanation closure axioms are in fact disguised versions of frame axioms!

$\neg F(\mathbf{x}, s) \wedge \neg P_F(\mathbf{x}, a, s) \rightarrow \neg F(\mathbf{x}, \mathit{do}(a, s))$

$F(\mathbf{x}, s) \wedge \neg N_F(\mathbf{x}, a, s) \rightarrow F(\mathbf{x}, \mathit{do}(a, s))$

(Recap) Successor state axioms

Further assume that our KB entails the following

a) integrity of the effect axioms: $\neg\exists \mathbf{x}, a, s. P_F(\mathbf{x}, a, s) \wedge N_F(\mathbf{x}, a, s)$

b) unique names for actions:

$$A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \rightarrow (x_1 = y_1) \wedge \ldots \wedge (x_n = y_n)$$

$A(x_1, \ldots, x_n) \neq B(y_1, \ldots, y_n)$ where A and B are distinct.

Then it can be shown that KB entails that (1),(2),(3) and (4) together are logically equivalent to

$$F(\mathbf{x}, \mathit{do}(a, s)) \equiv P_F(\mathbf{x}, a, s) \vee (F(\mathbf{x}, s) \wedge \neg N_F(\mathbf{x}, a, s))$$

This is called the successor state axiom for F.

For example, the successor state axiom for the *Broken* fluent is:

$$\begin{aligned} &\mathit{Broken}(x, \mathit{do}(a, s)) \equiv \\ &\quad \exists r\{a = \mathit{drop}(r, x) \wedge \mathit{Fragile}(x)\} \\ &\quad \vee \exists b\{a = \mathit{explode}(b) \wedge \mathit{NextTo}(b, x, s)\} \\ &\quad \vee \mathit{Broken}(x, s) \wedge \neg\exists r\{a = \mathit{repair}(r, x)\} \end{aligned}$$

An object x is broken after doing action a
iff
a is a dropping action and x is fragile
or a is a bomb exploding
(where x is next to the bomb)
or x was already broken and
a is not the action of repairing it

(Recap) Planning (answer extraction)

planning = theorem proving

Having formulated planning this way, we can use resolution with answer extraction to find a sequence of actions:

$$KB \models \exists s.\mathit{Goal}(s) \wedge \mathit{Legal}(s)$$

Since all of the required elements here can directly be represented using Horn clauses, we can employ Prolog for planning:

```
onfloor(X,do(drop(X),S)).
holding(X,do(pickup(X),S)).
poss(drop(X),S) :- holding(X,S).
poss(pickup(X),S).
ontable(b,s0).
legal(s0).
legal(do(A,S)) :- poss(A,S), legal(S).
```

With the Prolog goal $? - \mathtt{onfloor}(\mathtt{b}, \mathtt{S}), \mathtt{legal}(\mathtt{S}).$
we get the solution $\mathtt{S} = \mathtt{do}(\mathtt{drop}(\mathtt{b}), \mathtt{do}(\mathtt{pickup}(\mathtt{b}), \mathtt{s0}))$

Sitcalc in AILog

Snippets of an example of a delivery robot (`delrob_sitc.ail`)

```
% INITIAL SITUATION
sitting_at(rob,o109,init).
sitting_at(parcel,lng,init).
...
% DERIVED RELATIONS
at(Obj,Pos,S) <- sitting_at(Obj,Pos,S).
...
adjacent(o109,o103,_).
adjacent(o103,o109,_).
...
% STATIC RELATIONS
blocks(door1,o103,lab2).
opens(k1,door1).
autonomous(rob).
```

Sitcalc in AILog (2)

```
% ACTION PRECONDITIONS
poss(move(Ag,Pos,Pos_1),S)  <-
   autonomous(Ag) &
   adjacent(Pos,Pos_1,S) &
   sitting_at(Ag,Pos,S).
...
% PRIMITIVE PREDICATE DEFINITIONS
carrying(Ag,Obj,do(pickup(Ag,Obj),S)) <-
   poss(pickup(Ag,Obj),S).
sitting_at(Obj,Pos,do(A,S) )  <-
   poss(A,S) &
   sitting_at(Obj,Pos,S) &
   ~ move_action(A,Obj,Pos) &
   ~ pickup_action(A,Obj).
move_action(move(Obj,Pos,_),Obj,Pos).
pickup_action(pickup(_,Obj),Obj).
```

Sitcalc in AILog (3)

Using AILog to predict and plan

```
ailog: bound 12.
ailog: ask at(parcel,o111,S).
Answer: at(parcel,o111,do(move(rob,o109,o111),do(move(rob,lng,o109),
  do(pickup(rob,parcel),do(move(rob,o109,lng),init))))).
  [ok,more,how,help]: more.
Query failed due to depth-bound 12.
     [New-depth-bound,where,ok,help]: ok.

ailog:  bound 8.
ailog: ask carrying(rob,k1,S).
Answer: carrying(rob,k1,do(pickup(rob,k1),do(move(rob,o103,mail),
  do(move(rob,o109,o103),init)))).
  [ok,more,how,help]: how.
   carrying(rob,k1,do(pickup(rob,k1),do(move(rob,o103,mail),
    do(move(rob,o109,o103),init)))) <-
      1: poss(pickup(rob,k1),do(move(rob,o103,mail),do(move(rob,o109,o103),init)))
   How? [Number,up,retry,ok,prompt,help]: ok.
Answer: carrying(rob,k1,do(pickup(rob,k1),do(move(rob,o103,mail),
  do(move(rob,o109,o103),init)))).
  [ok,more,how,help]: ok.
```

Limitations of situation calculus

So far, situation calculus has a number of limitations:

- no time: cannot talk about how long actions take, or when they occur
- only known actions: no hidden exogenous actions, no unnamed events
- no concurrency or multiple agents: cannot talk about doing multiple actions at the same time
- only discrete situations: no continuous actions, like pushing an object from A to B (with duration, for example)
- only hypotheticals: cannot say that an action has occurred or will occur (different for, for example, event calculus)
- only primitive actions: no actions made up of other parts, like conditionals or iterations
- no probabilistic information about the world included in the actions

We will deal with the last two issues in the following slides

The limitation of primitive actions

So far, we have no way of handling in the situation calculus complex actions made up of other actions such as:

- **conditionals**: if the car is in the driveway then drive else walk
- **iterations**: while there is a block on the table, remove one
- **non-deterministic choice**: pickup some block and put it on the floor

Would be nice to *define* such actions in terms of the primitive actions and inherit their solution to the frame problem

Need a compositional treatment of the frame problem for complex actions

This results in a logical programming language for discrete event simulation and high-level robot control!

Compare to Java or C, where it is natural to program using **for**-loops, **while**-loops, **if**-**then**-**else**-statements, **case**-statements and all that.

Unless a logic programming language for action supports those things, it would not be used widely for AI and robotics.

The Do formula

For each complex action A it is possible to define a formula of the situation calculus,

$$\mathit{Do}(A, s, s')$$

that says that action A when started in situation s may legally terminate in situation s'.

Primitive actions: $\mathit{Do}(A, s, s') = \mathit{Poss}(A, s) \wedge s' = \mathit{do}(A, s)$

Sequence: $\mathit{Do}([A, B], s, s') = \exists s''.\mathit{Do}(A, s, s'') \wedge \mathit{Do}(B, s'', s')$

Conditionals:

$\mathit{Do}([\mathtt{if}\ \varphi\ \mathtt{then}\ A\ \mathtt{else}\ B], s, s') = \varphi(s) \wedge \mathit{Do}(A, s, s') \vee \neg\varphi(s) \wedge \mathit{Do}(B, s, s')$

Nondeterministic branching: $\mathit{Do}([A|B], s, s') = \mathit{Do}(A, s, s') \vee \mathit{Do}(B, s, s')$

Nondeterministic choice: $\mathit{Do}([\pi x.A], s, s') = \exists x.\mathit{Do}(A, s, s')$

etc.

This results in programming language constructs with a purely logical situation calculus interpretation!

Golog

- H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin and R. Scherl (1997) *GOLOG: A logic programming language for dynamic domains*, The journal of logic programming, vol 31(1-3), pp 59-83.
- Based on situation calculus
- Programming language for agents embedded in the (physical) world, based on rigourous logical formalization and supporting logical reasoning about knowledge and the world.
- Extended into many directions, e.g.
  - probabilistic knowledge
  - online/realtime control
  - multi-agent/game-theoretic aspects
  - knowledge and belief, epistemic logic
- Main book: Reiter (2001) *Knowledge in action: logical foundations for specifying and implementing dynamical systems*, MIT Press

Golog

Golog (Algol in logic) is a programming language that generalizes conventional imperative programming languages

- the usual imperative constructs + concurrency, nondeterminism, etc.
- bottoms out *not* on operatios on internal states (assignment statements, pointer updates) but on *primitive actions* in ther world (e.g. pickup a block)
- what the primitive actions do is user-specified by precondition and successor state actions

What does it mean to **execute** a Golog program?

- find a sequence of primitive actions such that performing them starting in some initial situation s would lead to a situation s' where the formula $\mathit{Do}(A, s, s')$ holds
- give the sequence of actions to a robot for *actual* execution in the world

Golog example

primitive actions:

pickup(x), *putonfloor*(x), *putontable(x)*

fluents:

Holding(x, s), *OnTable*(x, s), *OnFloor*(x, s)

action preconditions:

$$\begin{aligned}
&\mathit{Poss}(\mathit{pickup}(x), s) \equiv \forall z. \neg \mathit{Holding}(z, s)\\
&\mathit{Poss}(\mathit{putonfloor}(x), s) \equiv \mathit{Holding}(x, s)\\
&\mathit{Poss}(\mathit{putontable}(x), s) \equiv \mathit{Holding}(x, s)
\end{aligned}$$

Successor state axioms:

$$\mathit{Holding}(x, \mathit{do}(a, s)) \equiv a = \mathit{pickup}(x)$$

$$\vee \mathit{Holding}(x, s) \wedge a \neq \mathit{putontable}(x) \wedge a \neq \mathit{putonfloor}(x)$$

Golog example (2)

Successor state axioms:

$$\mathit{OnTable}(x, \mathit{do}(a, s)) \equiv a = \mathit{putontable}(x) \vee \mathit{OnTable}(x, s) \wedge a \neq \mathit{pickup}(x)$$

$$\mathit{OnFloor}(x, \mathit{do}(a, s)) \equiv a = \mathit{putonfloor}(x) \vee \mathit{OnFloor}(x, s) \wedge a \neq \mathit{pickup}(x)$$

Initial situation:

$$\forall x. \neg \mathit{Holding}(x, S_0), \mathit{OnTable}(x, S_0) \equiv x = A \vee x = B$$

Complex actions:

```
proc $\mathit{ClearTable}$
    while $\exists b.\mathit{OnTable}(b)$
            do $\pi b[\mathit{OnTable(b)?;\ RemoveBlock(b)}]$
proc $\mathit{RemoveBlock}(x)$ :
    $\mathit{pickup}(x)$ ; $\mathit{putonfloor}(x)$
```

Running Golog

To find a sequence of actions constituting a legal execution of a Golog program, we can use resolution with answer extraction again!

For the above example, we have $KB \models \exists s.\textit{Do}(\textit{ClearTable}, S_0, s)$

The result of this evaluation yields $s = \textit{do}(\textit{putonfloor}(B),$ $\textit{do}(\textit{pickup}(B), \textit{do}(\textit{putonfloor}(A), \textit{do}(\textit{pickup}(A), S_0))))$

and so a correct sequence is

$\langle \textit{pickup}(A), \textit{putonfloor}(A), \textit{pickup}(B), \textit{putonfloor}(B) \rangle$

When what is known about the actions and initial state can be expressed as Horn clauses, the evaluation can be done in Prolog:

The Golog interpreter has additional clauses such as:

```
do(A,S1,do(A,S1)) :- prim_action(A), poss(A,S1).
do(seq(A,B),S1,S2) :- do(A,S1,S3), do(B,S3,S2).
do(while(F,A),S1,S2) :- not holds(F,S1), S2=S1.
do(while(F,A),S1,S2) :- holds(F,S1), do(seq(A,while(F,A)),S1,S2).
```

This provides a convenient way of controlling a robot at a high-level.

The main Golog paper provides a nice example of *elevator control*

Programs as macros

Interesting additional planner Golog-style:

$\textbf{\textit{while}}\ \neg \textit{Goal}\ \textbf{\textit{do}}\ (\pi a)\ [\textit{Appropriate}(a)?;\ a]\ \textbf{\textit{endWhile}};$

Other useful properties one may use (for any program δ):

Correctness

$$\textit{Axioms} \models (\forall s).\textit{Do}(\delta, S_0, s) \rightarrow P(s)$$

(or stronger, for $\forall s_0, s)$.

Termination

$$\textit{Axioms} \models (\exists s)\textit{Do}(\delta, S_0, s)$$

(or stronger, $\textit{Axioms} \models (\forall s_0)(\exists s)\textit{Do}(\delta, S_0, s)$)

Second shortcoming: probability

Now that we know how logic, probability, resolution and actions work, it is fairly straightforward to combine all of them.

Easy to add probabilistic information to (ground) atoms as random variables (using probabilistic logic, explanations and solution formulas)

For actions we usually take the intuitive strategy to define a *probability distribution over deterministic alternatives*:

For example, using STRIPS, we extend an earlier example (M is monkey)

$O = \langle \mathit{Go}(x, y)$

$\quad \{\mathit{At}(\mathit{M}, x), \mathit{On}(\mathit{M}, \mathit{Floor})\},$

$\quad \{\mathit{At}(\mathit{M}, x)\}, \{\mathit{At}(\mathit{M}, Y)\}\rangle$,to

$O = \langle \mathit{Go}(x, y), \{\mathit{At}(\mathit{M}, x), \mathit{On}(\mathit{M}, \mathit{Floor})\},$

$\quad \{\{\mathit{At}(\mathit{M}, x)\}, \{\mathit{At}(\mathit{M}, Y)\}\rangle, \mathit{Prob} = 0.9\}, \{\emptyset, \emptyset\rangle, \mathit{Prob} = 0.1\}$

In other words, in 90% the action has the regular semantics, and in 10% of the cases, the action has no effect.

Can use it to (compactly) define Markov decision processes in relational domains! Challenge: probabilities and value functions dependent on the number of domain objects!

Probabilistic Sitcalc in AILog

A probabilistic world in situation calculus in AILog (`robot_sitc.cil`)

```
% initial situation has the following probabilities
%   P(locked(door,s0)) = 0.9
%   P(at_key(r101,s0)|locked(door,s0)) = 0.7
%   P(at_key(r101,s0)|unlocked(door,s0)) = 0.2
%   ( from which we conclude P(at_key(r101,s0))=0.65

prob locked(door,s0):0.9,unlocked(door,s0):0.1.
prob at_key_lo(r101,s0):0.7,at_key_lo(r123,s0):0.3.
prob at_key_unlo(r101,s0):0.2,at_key_unlo(r123,s0):0.8.

at(key,R,s0) <- at_key_lo(R,s0) & locked(door,s0).
at(key,R,s0) <- at_key_unlo(R,s0) & unlocked(door,s0).

% initially the robot is at room 111.
at(robot,r111,s0).
% path(From,To,Route,Risky,Cost)
%     Risky means whether it has to go past the stairs
path(r101,r123,direct,yes,50).
path(r101,r123,long,no,90).
path(r101,door,direct,yes,50).
```

Probabilistic Sitcalc in AILog (2)

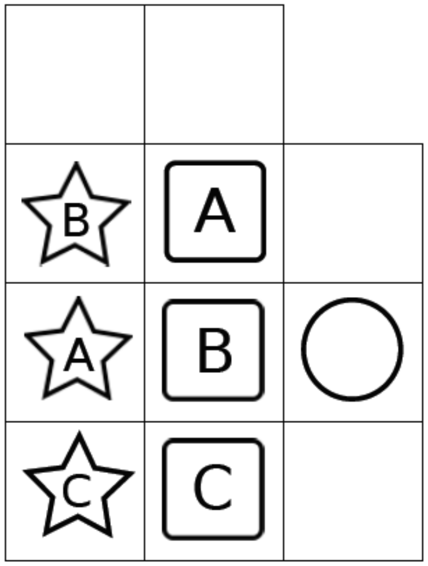
```
carrying(key,do(pickup(key),S)) <-
    at(robot,P,S) &
    at(key,P,S) &
    pickup_succeeds(S).

carrying(key,do(A,S)) <-
    carrying(key,S) &
    A \= putdown(key) &
    A \= pickup(key) &
    keeps_carrying(key,S).

prob pickup_succeeds(S):0.88, pickup_fails(S):0.12.
prob keeps_carrying(key,S):0.95, drops(key,S):0.05.

sense(at_key,S) <-
   at(robot,P,S) & at(key,P,S) & sensor_true_pos(S).
sense(at_key,S) <-
   at(robot,P1,S) & at(key,P2,S) &
   P1 \= P2 & sensor_false_neg(S).

prob sensor_true_pos(S):0.92, sensor_false_neg(S):0.08.
prob sensor_true_neg(S):0.97, sensor_false_pos(S):0.03.
```

Probabilistic Sitcalc in AILog (3)

```
AILog theory robot_sitc.cil loaded.
ailog: predict utility(V,do(enter_lab,do(goto(door,direct),s0))).
Answer: P(utility(160,do(enter_lab,do(goto(door,direct),s0)))|Obs)=0.9.
  [ok,more,explanations,worlds,help]: ok.
ailog: predict sense(at_key,do(goto(r101,direct), s0)).
Answer: P(sense(at_key,do(goto(r101,direct),s0))|Obs)=0.5634.
  [ok,more,explanations,worlds,help]: explanations.
  0: ass([],[sensor_true_pos(do(goto(r101,direct),s0)),would_not_fall_down_stairs(s
  1: ass([],[sensor_true_pos(do(goto(r101,direct),s0)),would_not_fall_down_stairs(s
  2: ass([],[sensor_false_neg(do(goto(r101,direct),s0)),would_not_fall_down_stairs(
  3: ass([],[sensor_false_neg(do(goto(r101,direct),s0)),would_not_fall_down_stairs(
  [ok,more,how i,help]: ok.
Answer: P(sense(at_key,do(goto(r101,direct),s0))|Obs)=0.5634.
  [ok,more,explanations,worlds,help]: ok.
ailog: observe sense(at_key,do(goto(r101,direct), s0)).
Answer: P(sense(at_key,do(goto(r101,direct),s0))|Obs)=0.5634.
  [ok,more,explanations,worlds,help]: ok.
ailog: predict utility(V,do(enter_lab, do(unlock_door, do(goto(door,long), do(picku
Answer: P(utility(90,do(enter_lab,do(unlock_door,do(goto(door,long),do(pickup(key),
  [ok,more,explanations,worlds,help]: ok.
```

2nd Assignments: second part

The famous box-pushing game Sokoban



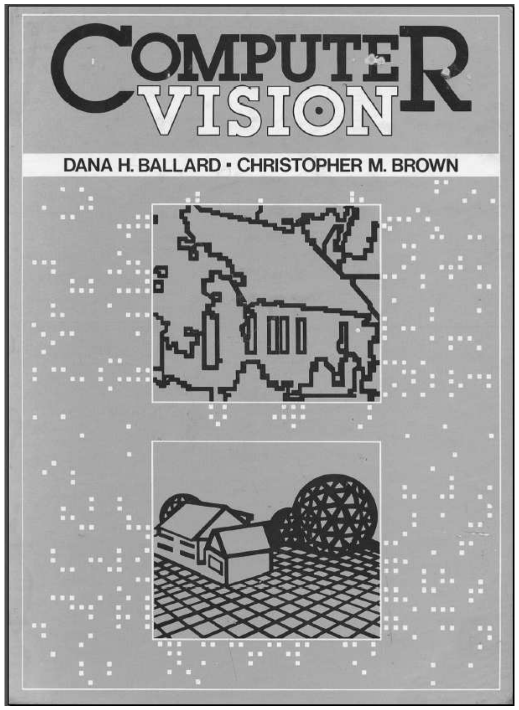
Example Sokoban problem. The grid is a 3x4 rectangle with the upper-right square missing. The agent is represented by the circle. There are three crates (represented by the boxes) A, B and C. The stars represent the goal locations for the respective crates.

The assignment will test your ability to *model* and *implement* this domain in a simple situation calculus.

Pauze

The Early Days (1982)

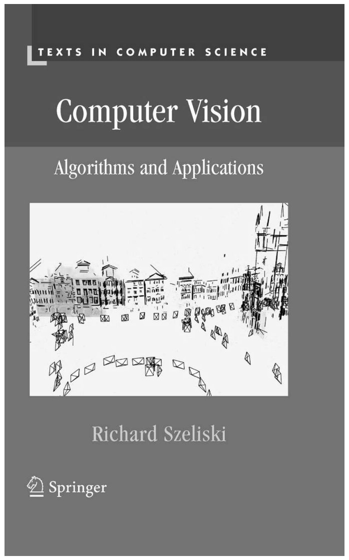
http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/bandb.htm



Chapters: computer vision issues, imagae formation, early processing, boundary detection, region growing, texture, motion, **representation of 2D geometric structures**, **representations of 3D structures**, **knowledge representation and use**, **matching**, **inference**, **goal achievement**,

Modern computer vision (2011)

http://szeliski.org/Book/


TEXTS IN COMPUTER SCIENCE

Computer Vision

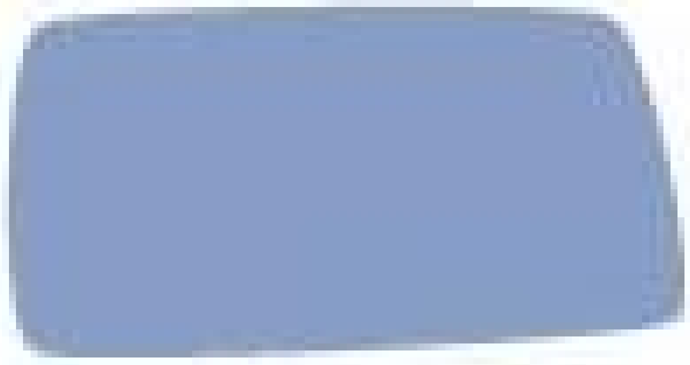
Algorithms and Applications

Richard Szeliski

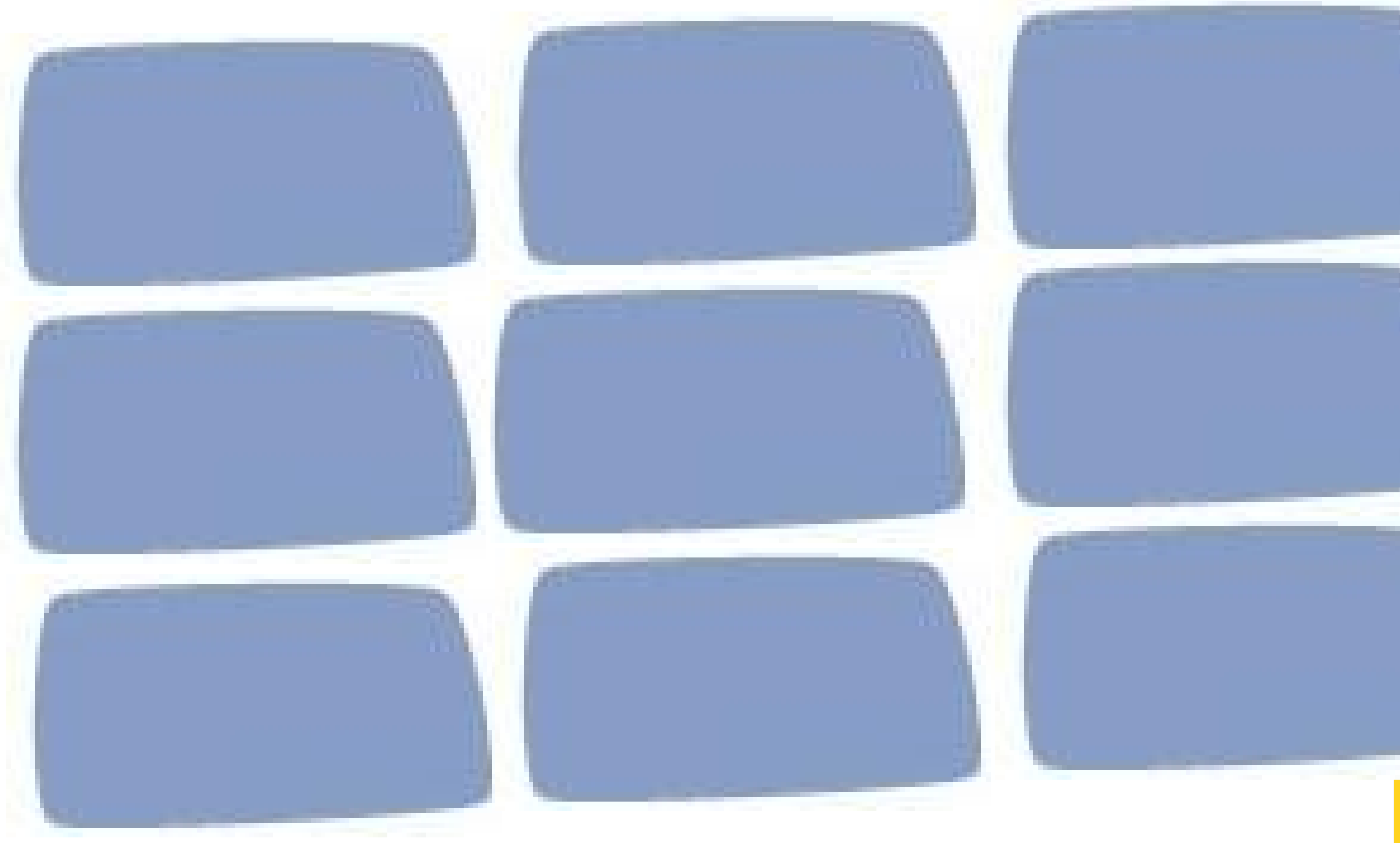
Springer

Chapters: introduction, image formation, image processing, feature detection and matching, segmentation, feature-based alignment, structure from motion, dense motion estimation, image stitching, computational photography, stereo correspondence, 3D reconstruction, image-based rendering, recognition,

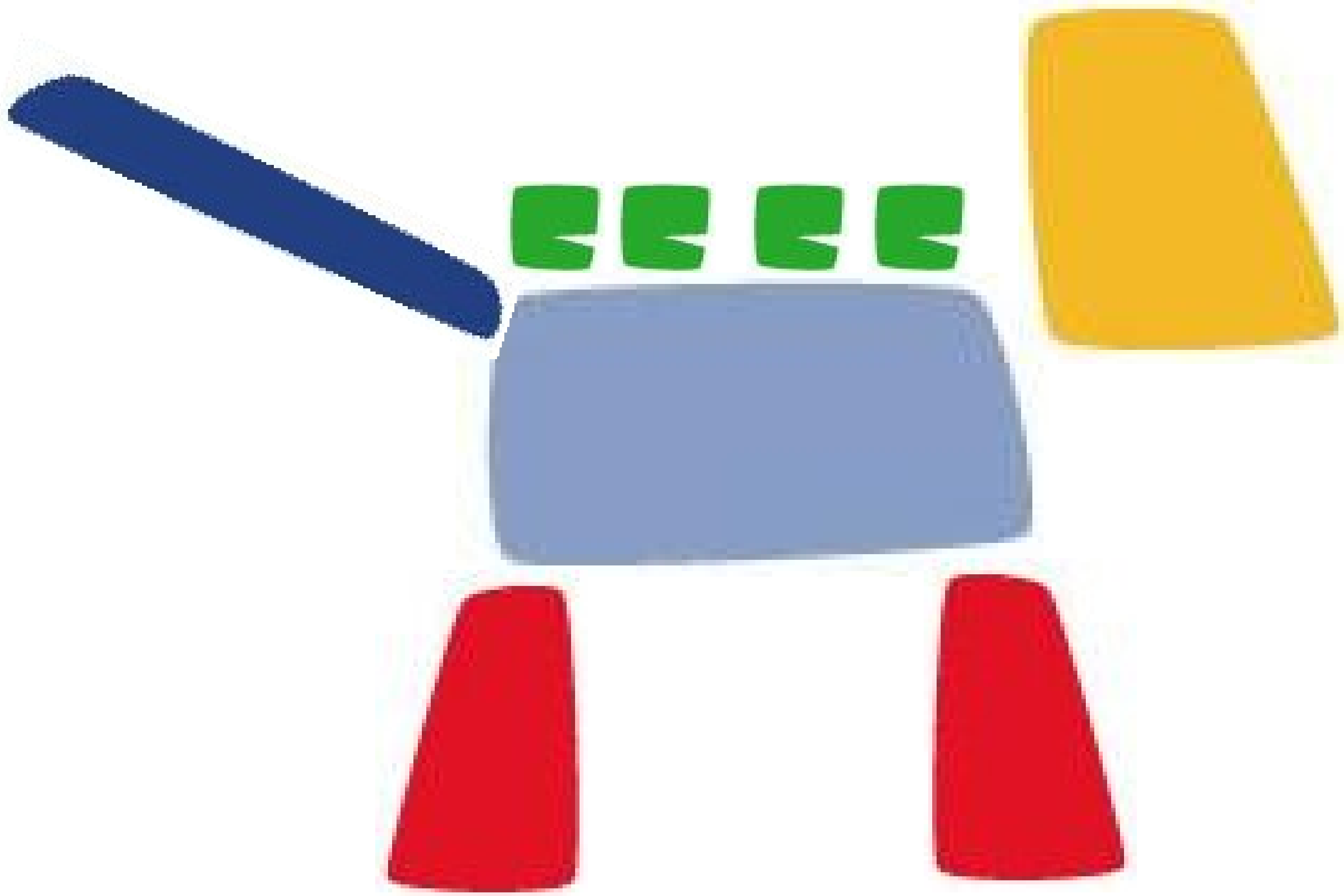
Not much attention for knowledge representation issues per se



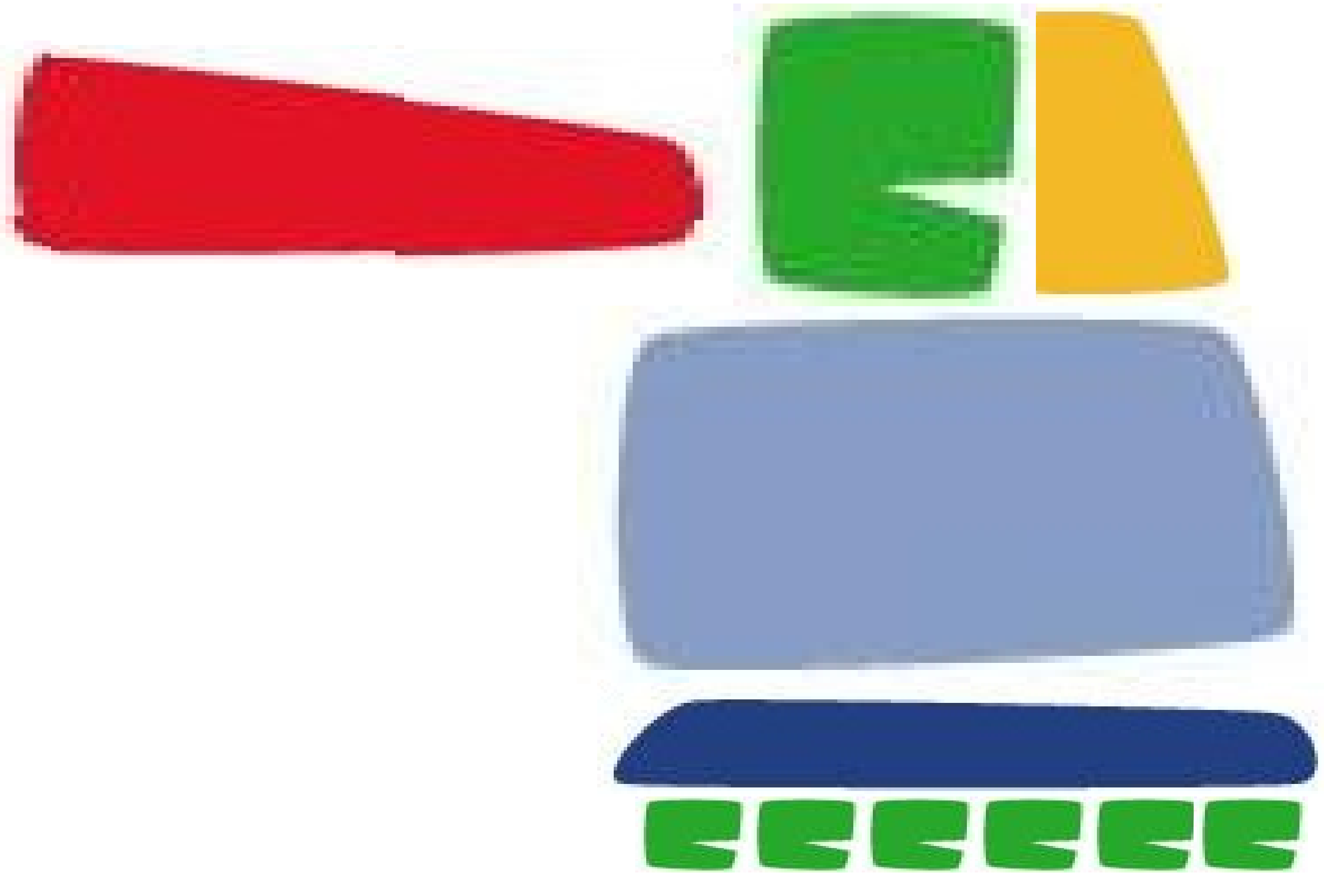
Another Brick in the Wall?



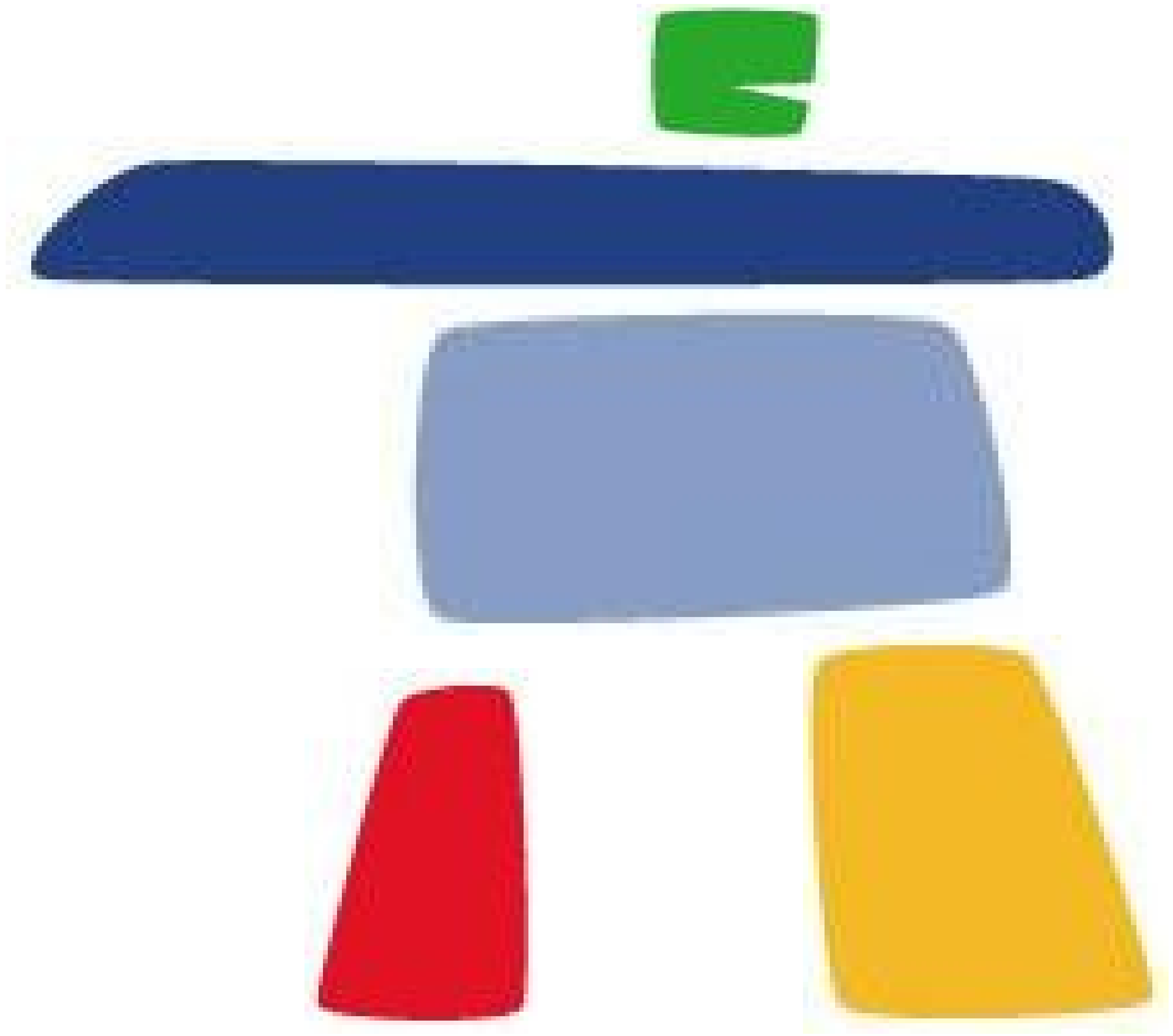
Jurassic Park?



Ceci n'est pas un Tank?



A Weird Cartoon Figure?

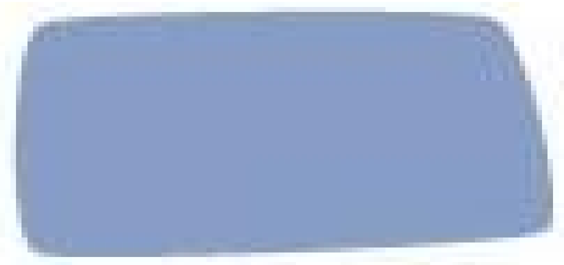


A Winter Athlete!

TM/MC

vancouver 2010

Context and Configurations

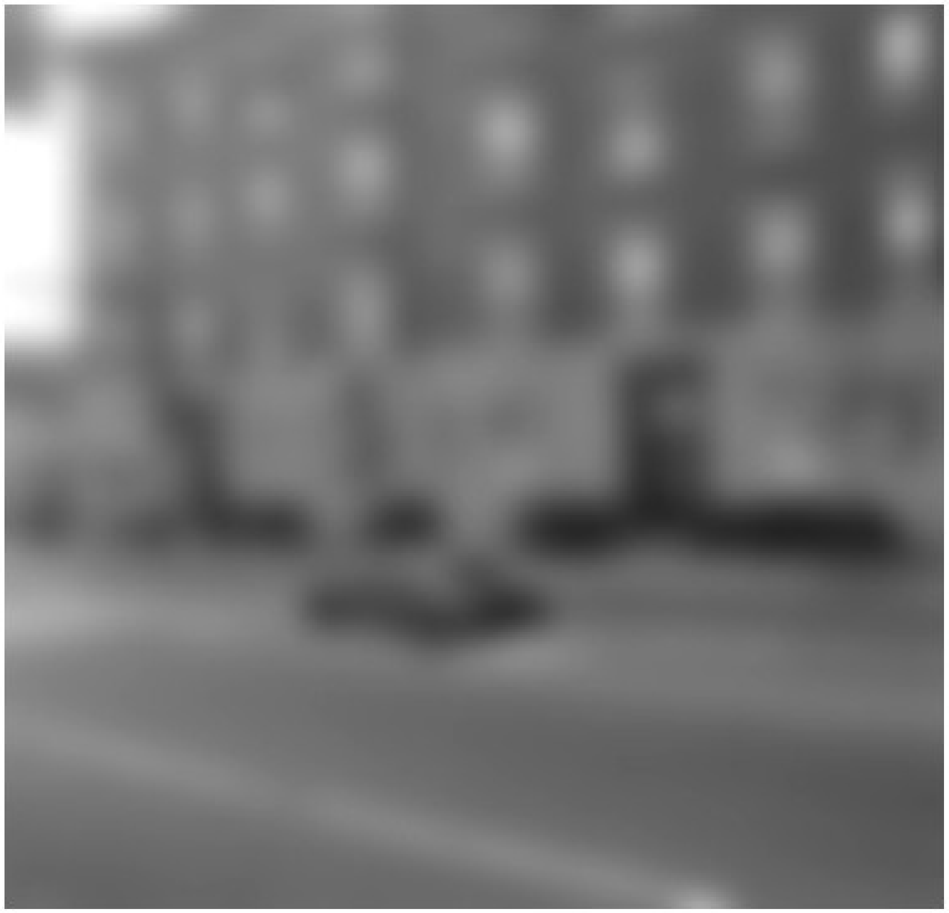
The meaning of 

depends on the **context** of surrounding visual components
and their specific **spatial configuration**

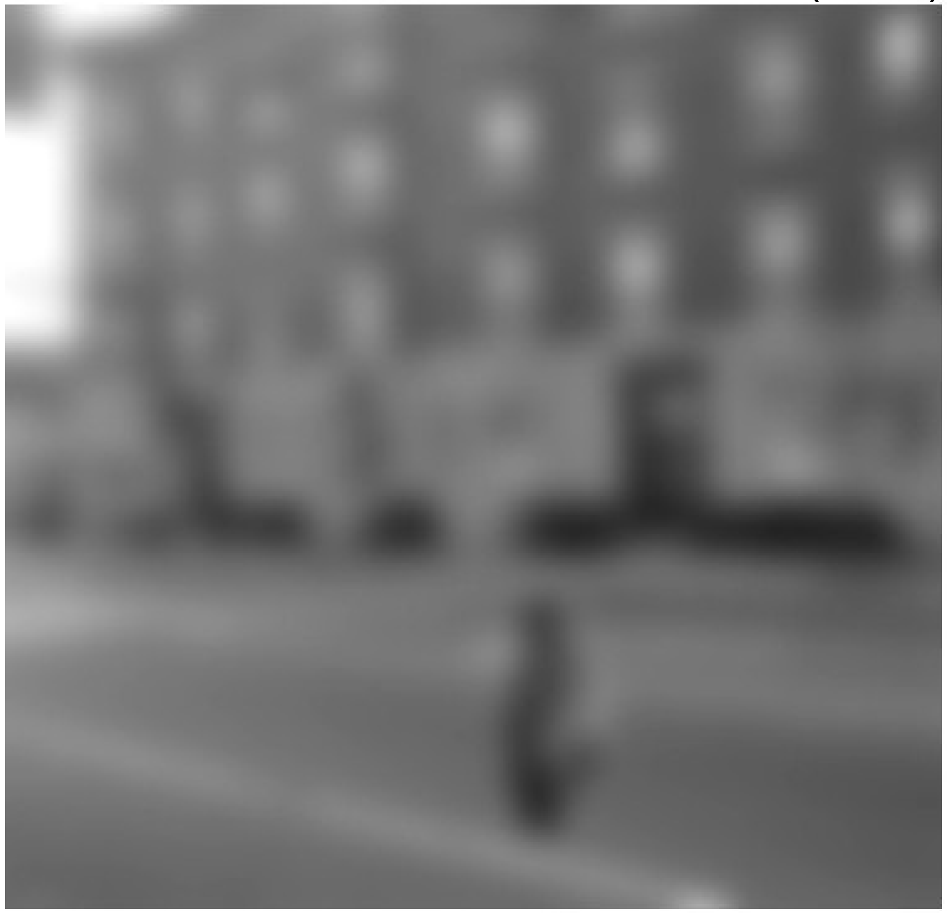
helped by background knowledge

The Value of Context in Vision

Torralba (2003)



A Car?



A Pedestrian?

An inactive lawnmower

An active lawnmower

High-Level Vision

High-level knowledge representation

On top of – but mixed with -- Low-level vision techniques

Part-based object representations

Scene understanding

Uncertain + Relational
→ ideal for SRL

Visual interpretation

Q: Is constraint satisfaction only useful when we are dealing with a nerdy *puzzle* of some sort?

A: No, it also can apply to ordinary human behaviours like vision.

Q: What is involved with vision? with seeing something?

A: It involves coming up with an *interpretation* for a 2-dimensional grid of colours and intensities.

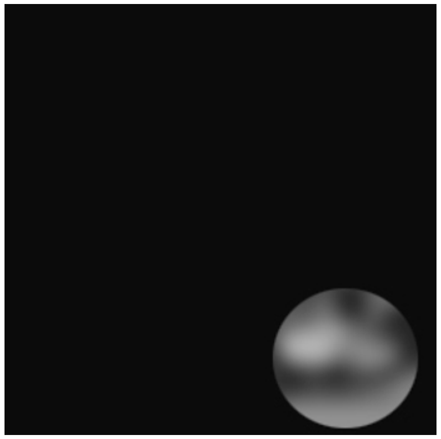
The main question:

what am I looking at?

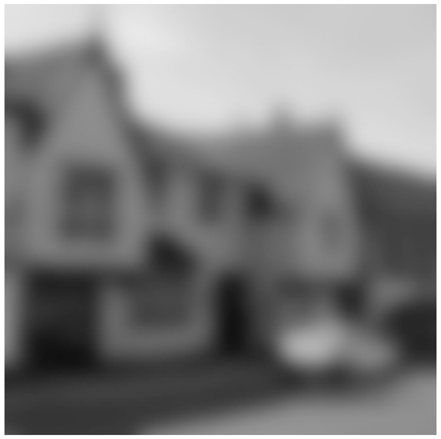
Thinking as part of seeing

Although much of the visual process is something that happens without any thought, part of seeing is using what you *know* about the world.

What is in the circle?



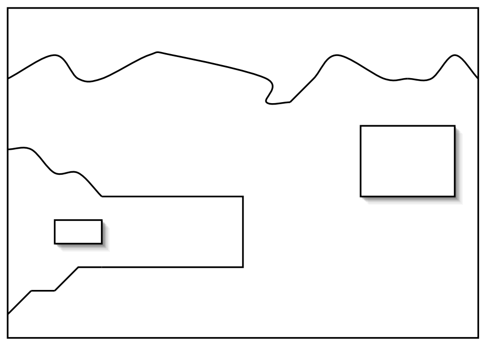
Now look at some surrounding context



A similar but simpler case: aerial sketch maps

Would like to label the regions in the sketch map on the right as either

`grass`, `water`, `pavement`,
`house`, `vehicle`

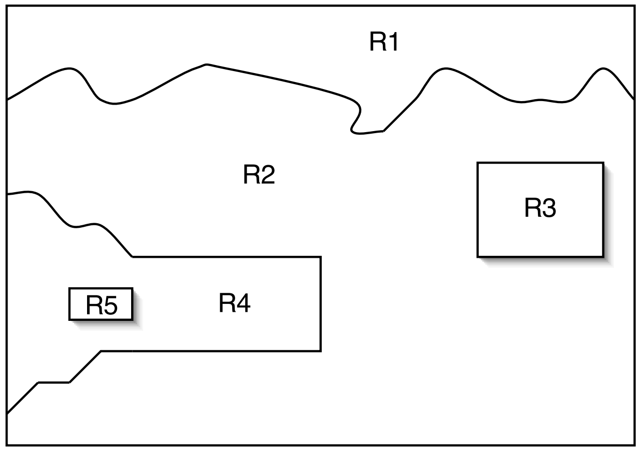


subject to constraints such as

- a region cannot border or be surrounded by another region with the same label
- houses cannot be next to or surrounded by water
- vehicles must be next to or surrounded by pavement
- pavement cannot be completely inside any other region
- houses, vehicles and pavement are regular (straight-edged); grass and water are irregular
- vehicles are small; the other regions are large

Visual properties

The task here is to determine how properties of the image can be translated into suitable constraints.



The more we extract from the image, the more we are able to rule out incorrect interpretations

In our example image:

1. region R5 is small; the others are large
2. region R3 and R5 are regular; R1 and R2 are irregular; R4 could go either way
3. region R1 borders on R2; R2 borders on R4
4. region R3 is inside R2; R5 is inside R4

Visual constraints

We can handle constraints (1) and (2) with facts like

```
large(grass).
small(vehicle).
regular(pavement).
irregular(water).
```

To handle (3), we simply ensure that the two regions do not *violate* any of the given rules about borders:

- the two regions must be different
- they must not be `house` and `water`
- if one is `vehicle`, the other must be `pavement`

Constraint (4) is handled analogously.

What is allowed in an interpretation?

regions.pl

```
% The five types of regions that can appear in an image
region(grass).   region(water).   region(pavement).
region(house).   region(vehicle).

% small(X) holds when region X can be small in an image.
small(vehicle).

% regular(X) holds when region X can be regular in an image.
regular(pavement).  regular(house).  regular(vehicle).

% border(X,Y) holds when region X can border region Y.
border(X,Y) :- \+ bad_border(X,Y), \+ bad_border(Y,X).

   % Unacceptable borders
   bad_border(X,X).
   bad_border(house,water).
   bad_border(vehicle,X) :- \+ X=pavement.

% inside(X,Y) holds when region X can be surrounded by Y.
inside(X,Y) :- \+ bad_inside(X,Y).

   % Unacceptable containment
   bad_inside(X,X).
   bad_inside(house,water).
   bad_inside(vehicle,X) :- \+ X=pavement.
   bad_inside(pavement,_).
```

The aerial sketch map interpretation in Prolog

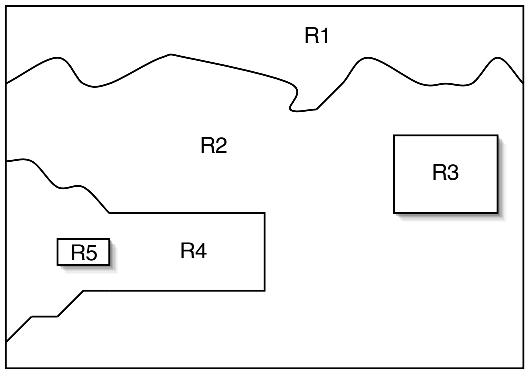
sketch.pl

```
solution(R1,R2,R3,R4,R5) :-
   region(R1), region(R2), region(R3), region(R4), region(R5),
   % Size constraints
      \+ small(R1), \+ small(R2), \+ small(R3),
      \+ small(R4), small(R5),
   % Regularity constraints (none for R4)
       regular(R3), regular(R5), \+ regular(R2), \+ regular(R1),
   % Border constraints
       border(R1,R2), border(R2,R4),
   % Containment constraints
       inside(R3,R2), inside(R5,R4).

% The definitions of region, small, border, etc. are elsewhere.
```

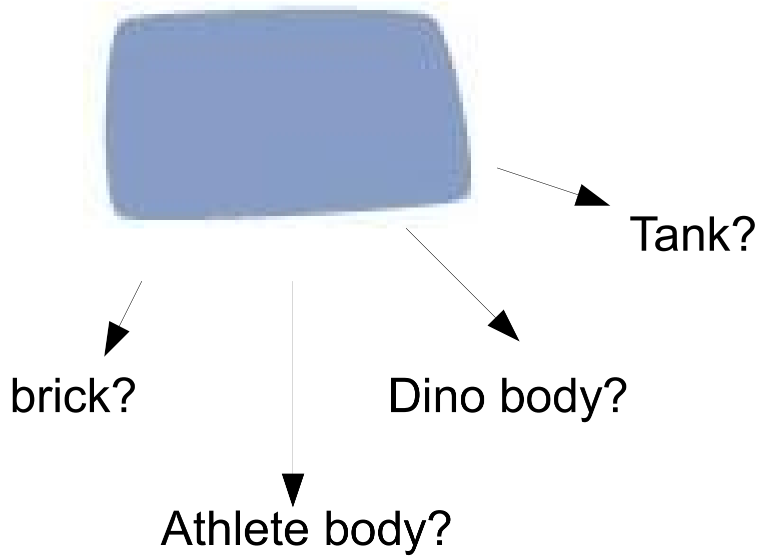
Loading this and the region constraints:

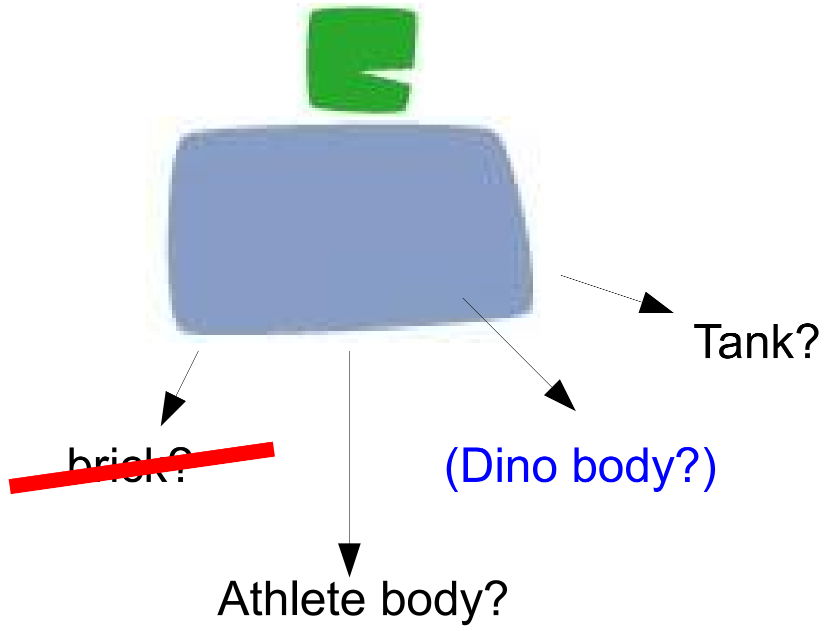
```
?- solution(R1,R2,R3,R4,R5).
R1 = water,  R2 = grass,  R3 = house,
R4 = pavement,  R5 = vehicle
```

and this is the only solution.



Explanations and Hypotheses



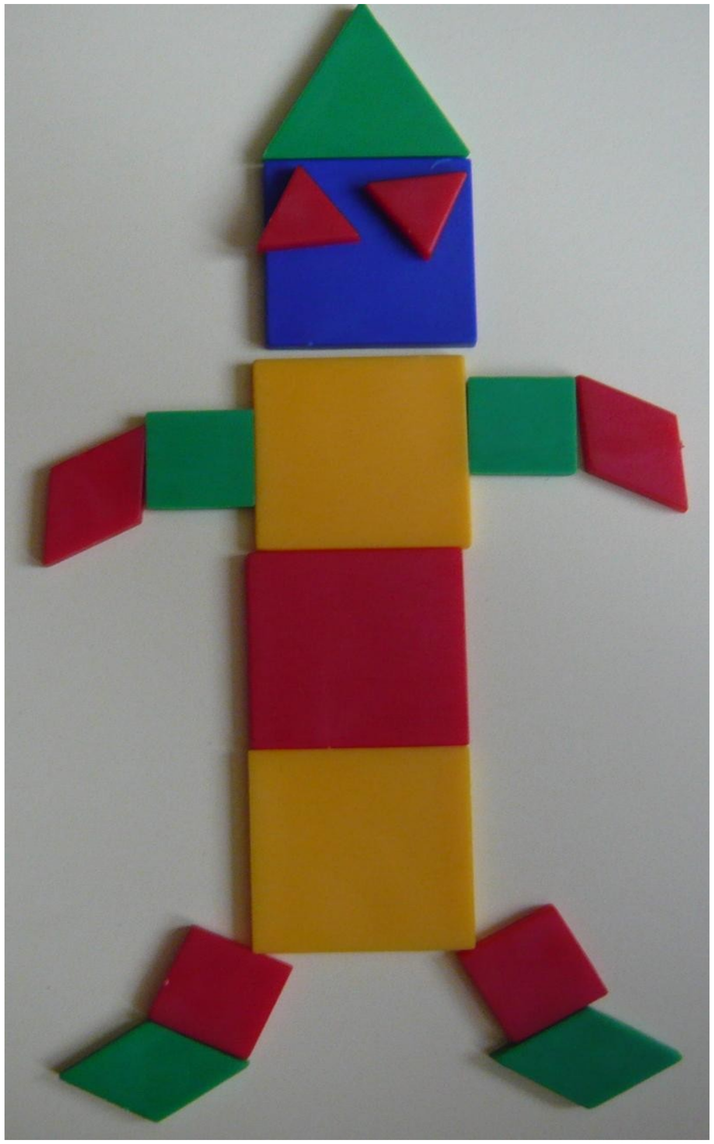


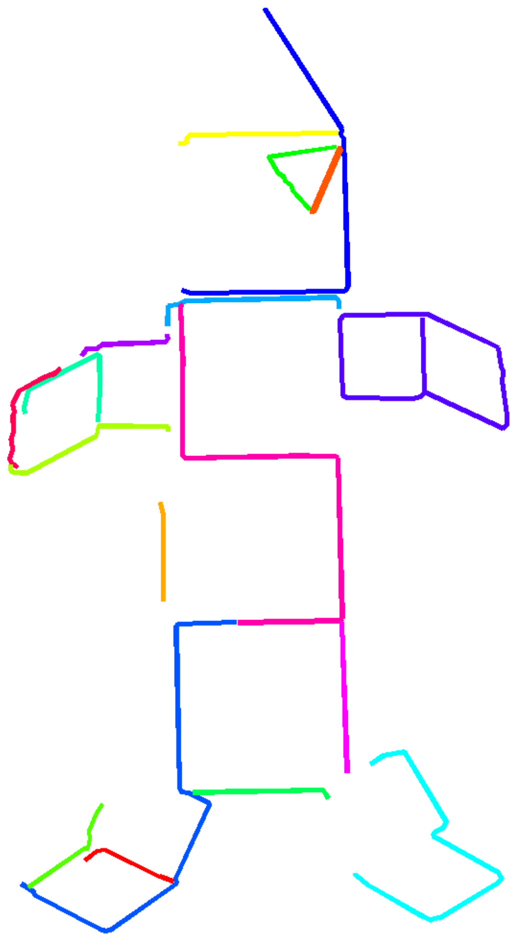
In a (probabilistic) logical setting:

- **Logical reasoning to find explanations of visual input**
- **Constraints + hypothesis testing**
- **Most probable (semantic) explanations**

→ **Models are needed to do all of this**

Beyond Shapes: Bongard 3.0.





Real images

Low-level segmentation

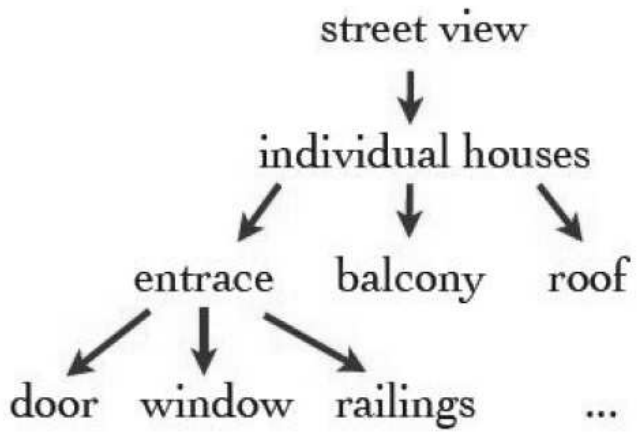
Uncertainty

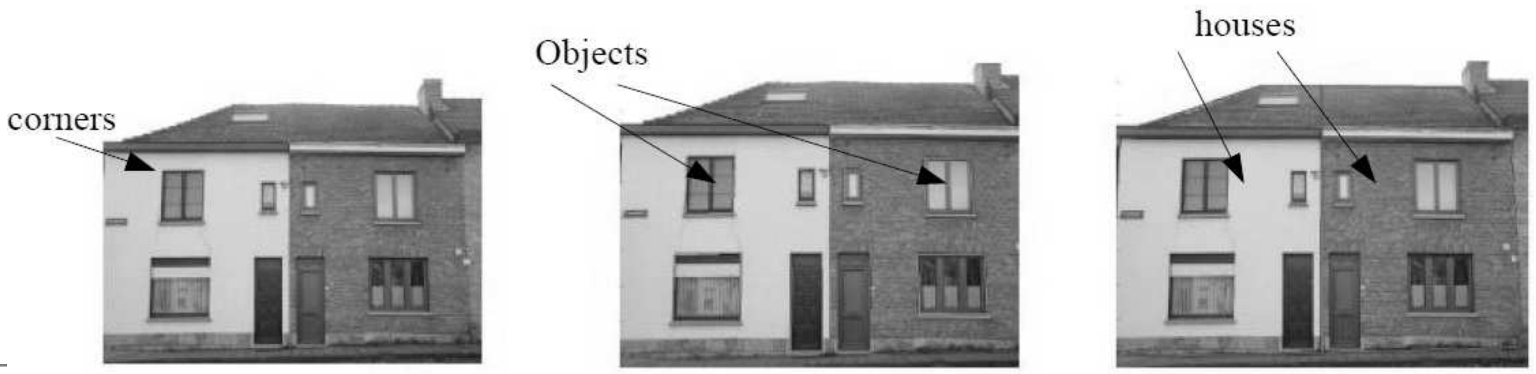
Noise

Example: Houses (1)

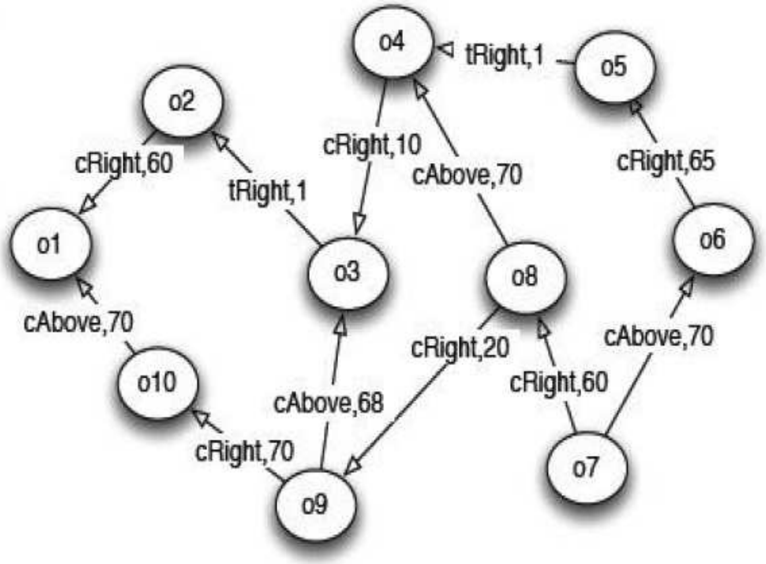
Antanas, van Otterlo, Oramas, Tuytelaars, De Raedt, Neurocomputing (2013, in press)

- Given
  - a hierarchical decomposition of images into concepts of different granularity
  - concept examples at each layer
  - a new image
- Find
  - concept delineations at each layer, i.e repeated structures



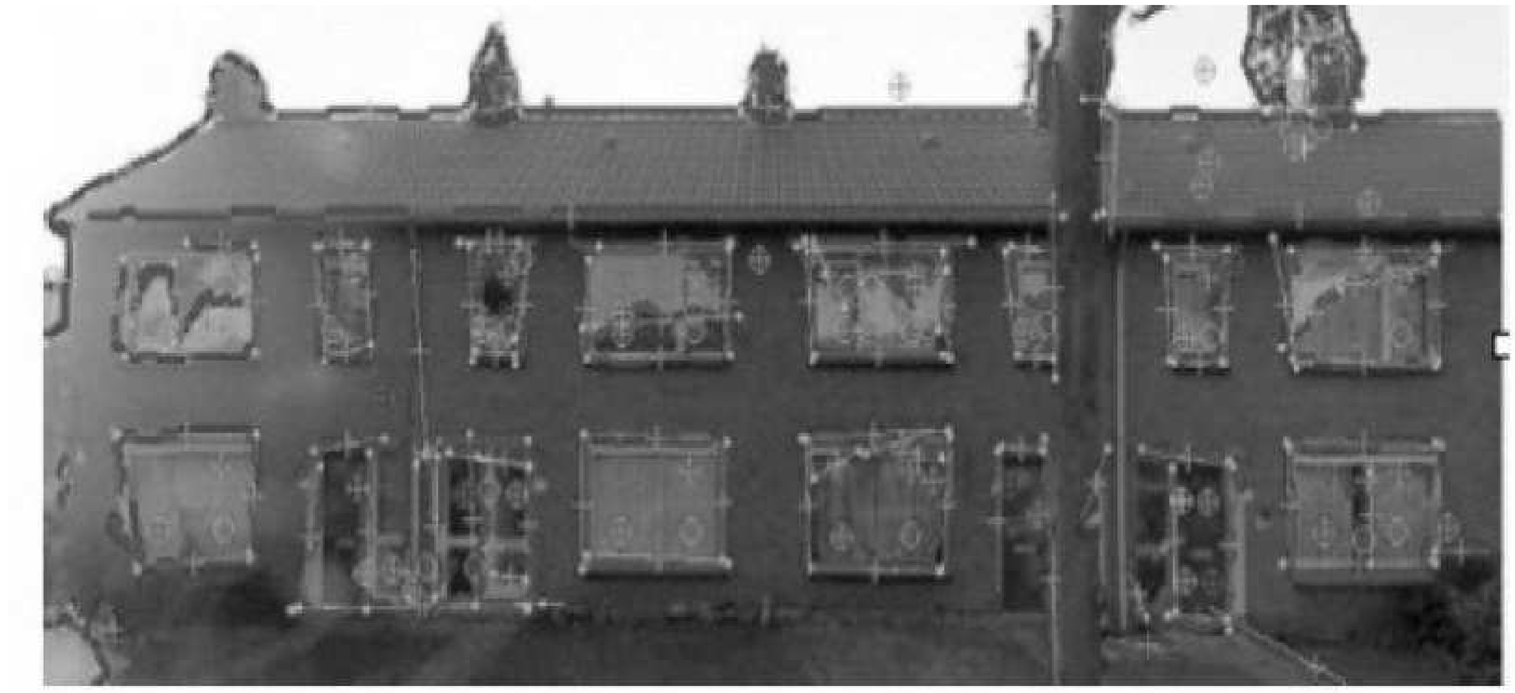


Houses (2)

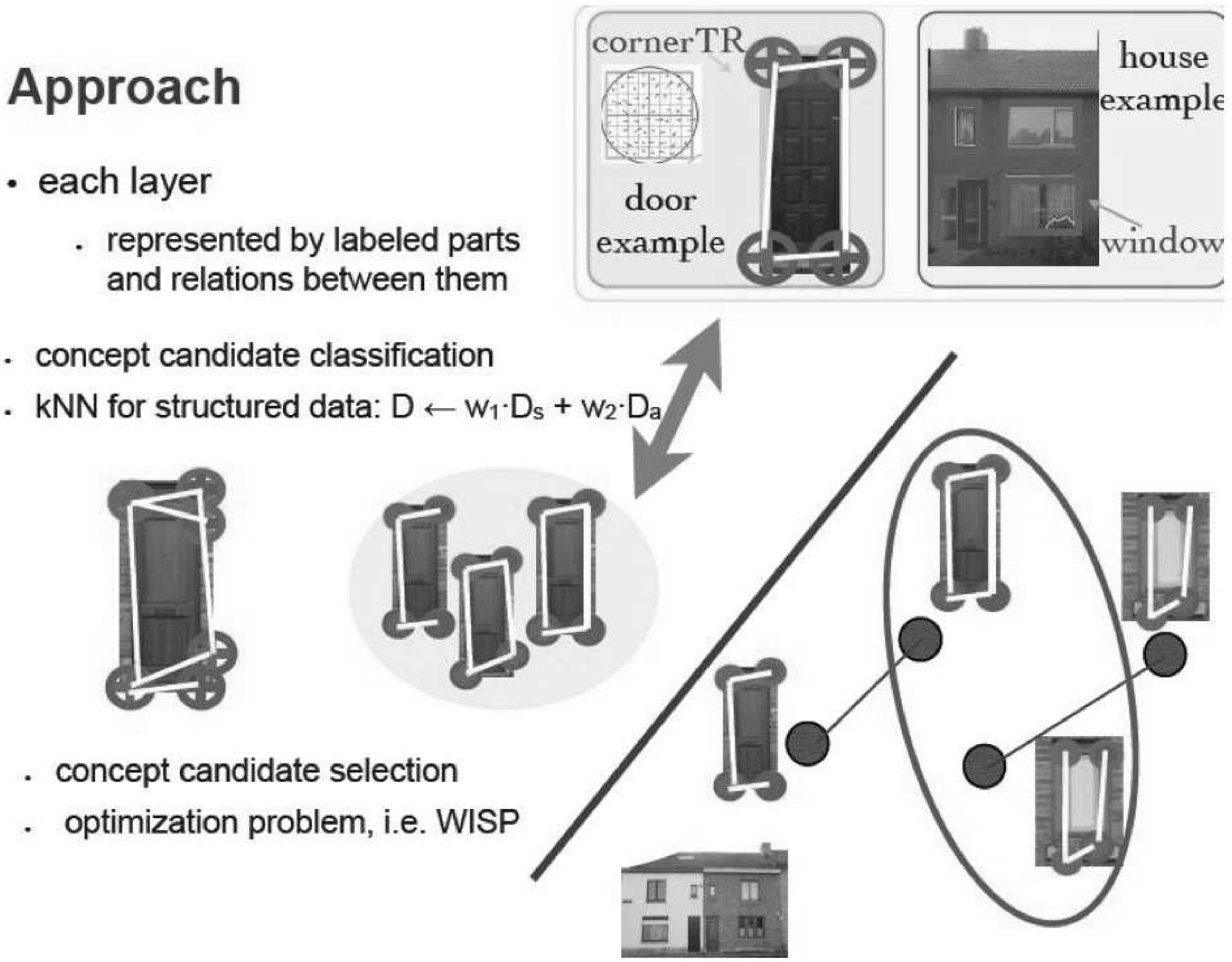


- Each layer: features + spatial relations
  - Configurations of:
  - **Block**: houses
  - **House**: windows, doors, etc
  - **Door**: detected line corners

Houses (3)



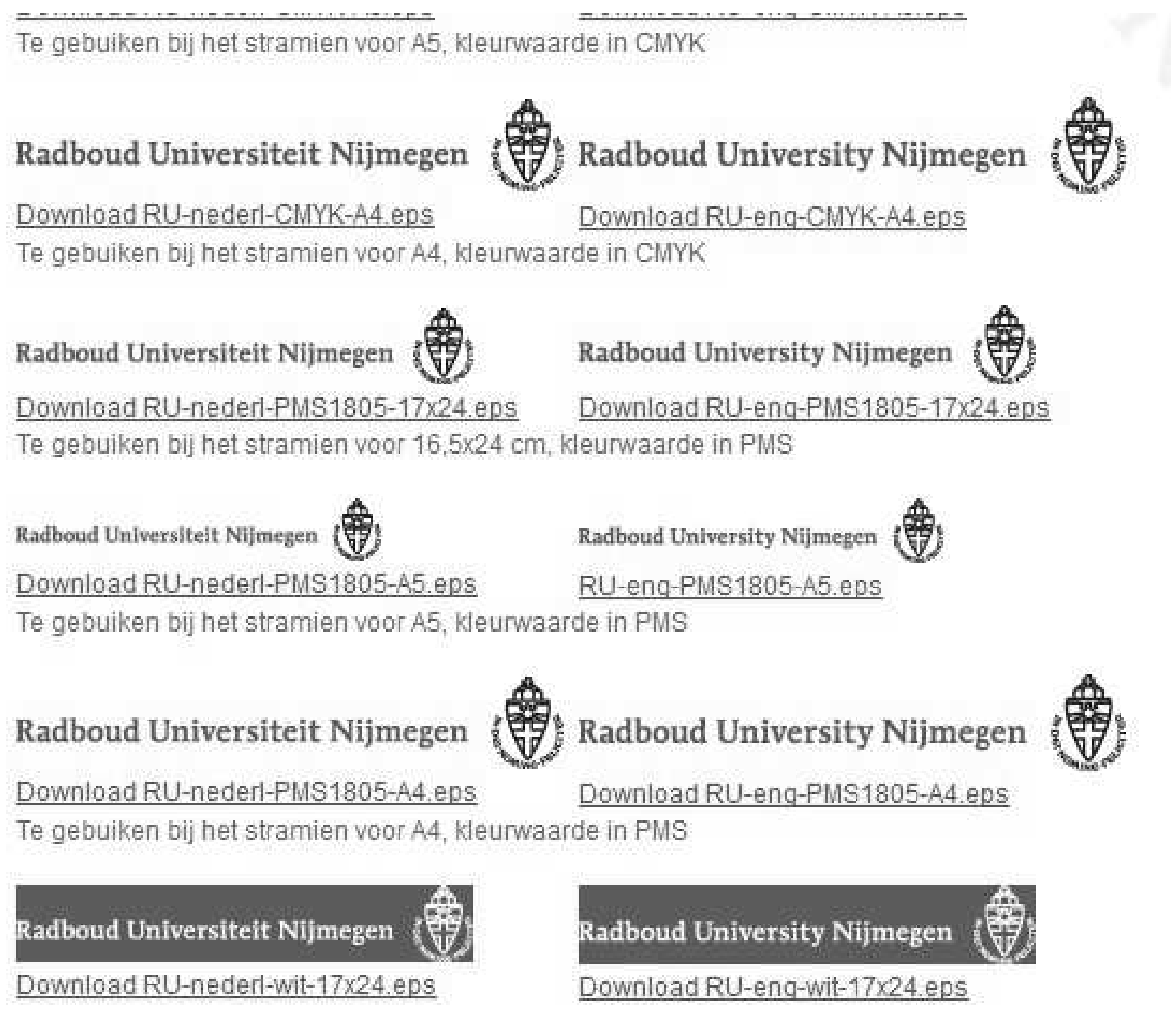
Houses (4)



Approach

- each layer
  - represented by labeled parts and relations between them
- concept candidate classification
- kNN for structured data: $D \leftarrow w_1 \cdot D_s + w_2 \cdot D_a$
- concept candidate selection
- optimization problem, i.e. WISP

Example: Logos



Te gebuiken bij het stramien voor A5, kleurwaarde in CMYK

Radboud Universiteit Nijmegen | Radboud University Nijmegen

Download RU-nederl-CMYK-A4.eps | Download RU-eng-CMYK-A4.eps

Te gebuiken bij het stramien voor A4, kleurwaarde in CMYK

Radboud Universiteit Nijmegen | Radboud University Nijmegen

Download RU-nederl-PMS1805-17x24.eps | Download RU-eng-PMS1805-17x24.eps

Te gebuiken bij het stramien voor 16,5x24 cm, kleurwaarde in PMS

Radboud Universiteit Nijmegen | Radboud University Nijmegen

Download RU-nederl-PMS1805-A5.eps | RU-eng-PMS1805-A5.eps

Te gebuiken bij het stramien voor A5, kleurwaarde in PMS

Radboud Universiteit Nijmegen | Radboud University Nijmegen

Download RU-nederl-PMS1805-A4.eps | Download RU-eng-PMS1805-A4.eps

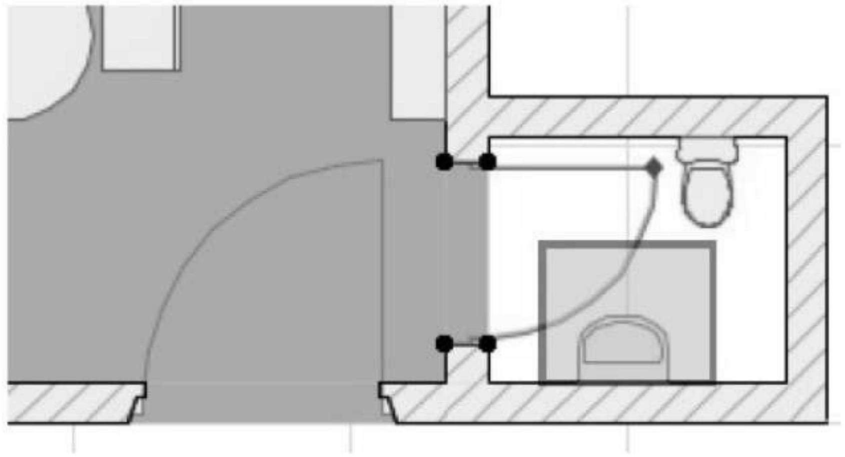
Te gebuiken bij het stramien voor A4, kleurwaarde in PMS

Radboud Universiteit Nijmegen | Radboud University Nijmegen

Download RU-nederl-wit-17x24.eps | Download RU-eng-wit-17x24.eps

Example: Design

Bhatt, Lee, Schultz (2011):



(a) Wash Sink and Door

```
safety(Door, Object) :-
            operational_space(Door, Op),
            functional_space(Object, Fs),
            not(topology(Op, Fs, overlaps)).
```

Example: Events

Damen and Hogg: bicycle theft recognition:

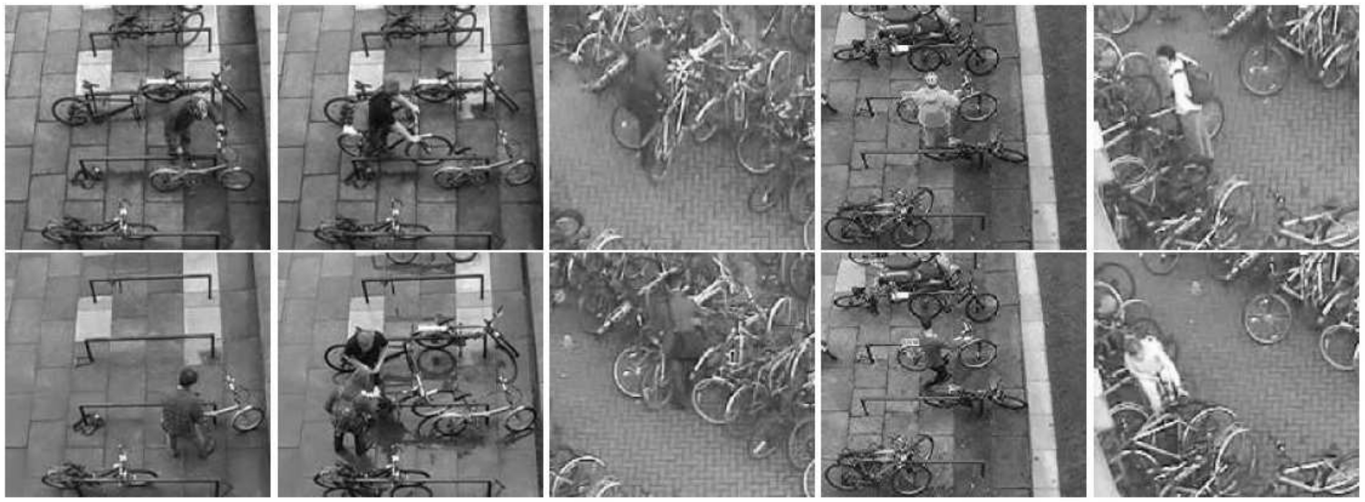
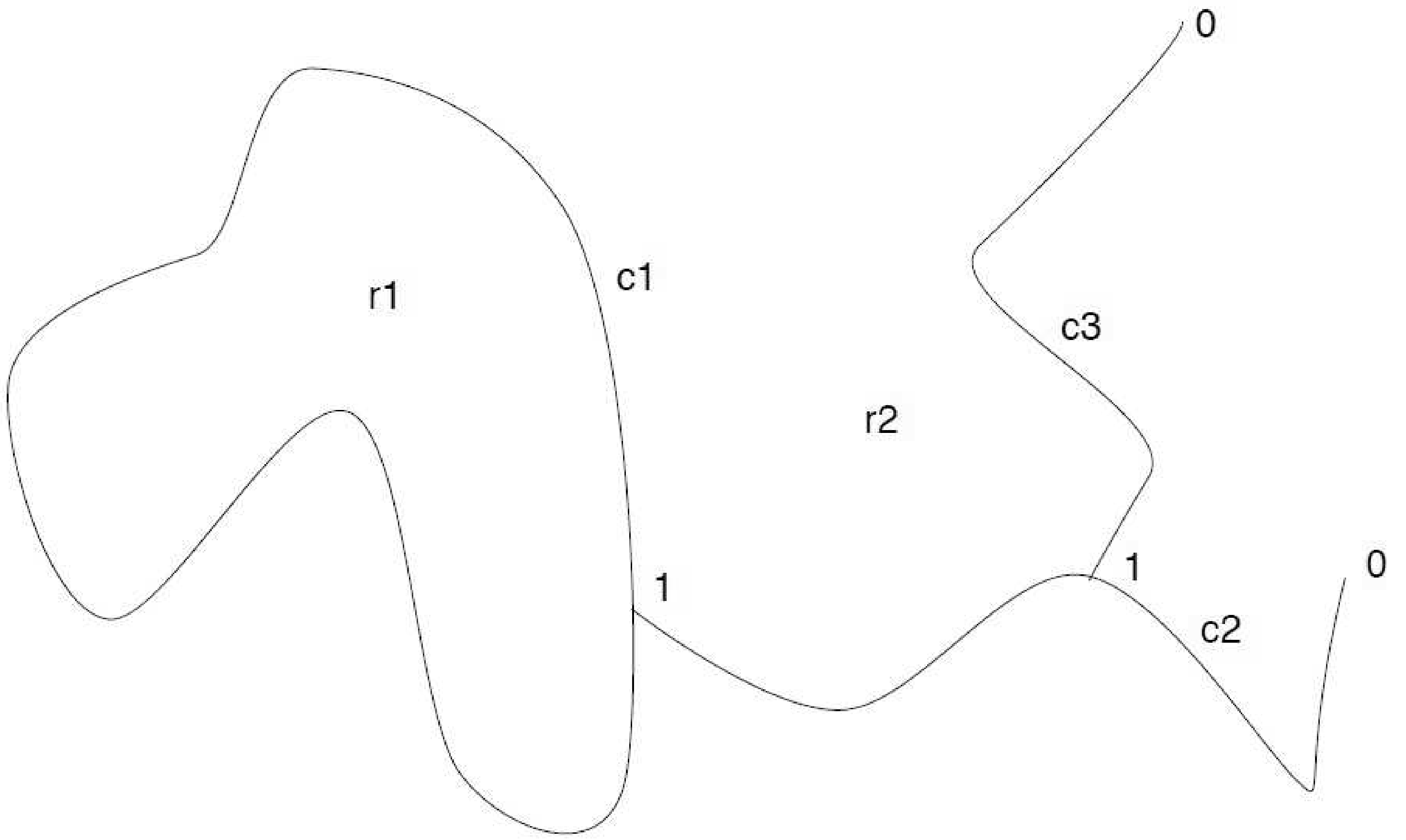


Figure 13. Five examples of connected events. The first four are correctly connected. The fourth column represents a simulated theft. The fifth example shows an incorrect connection. Recall that no clothing color comparison is performed. Individuals are connected by linking the person to a cluster and correctly linking dropped to picked bicycle clusters.

Metro stations, CCTV, robot manipulation (3D/Kinect), etc.

One of our MSc students is doing an internship with Philips Eindhoven on **logical event recognition** for video data in psychiatric care.

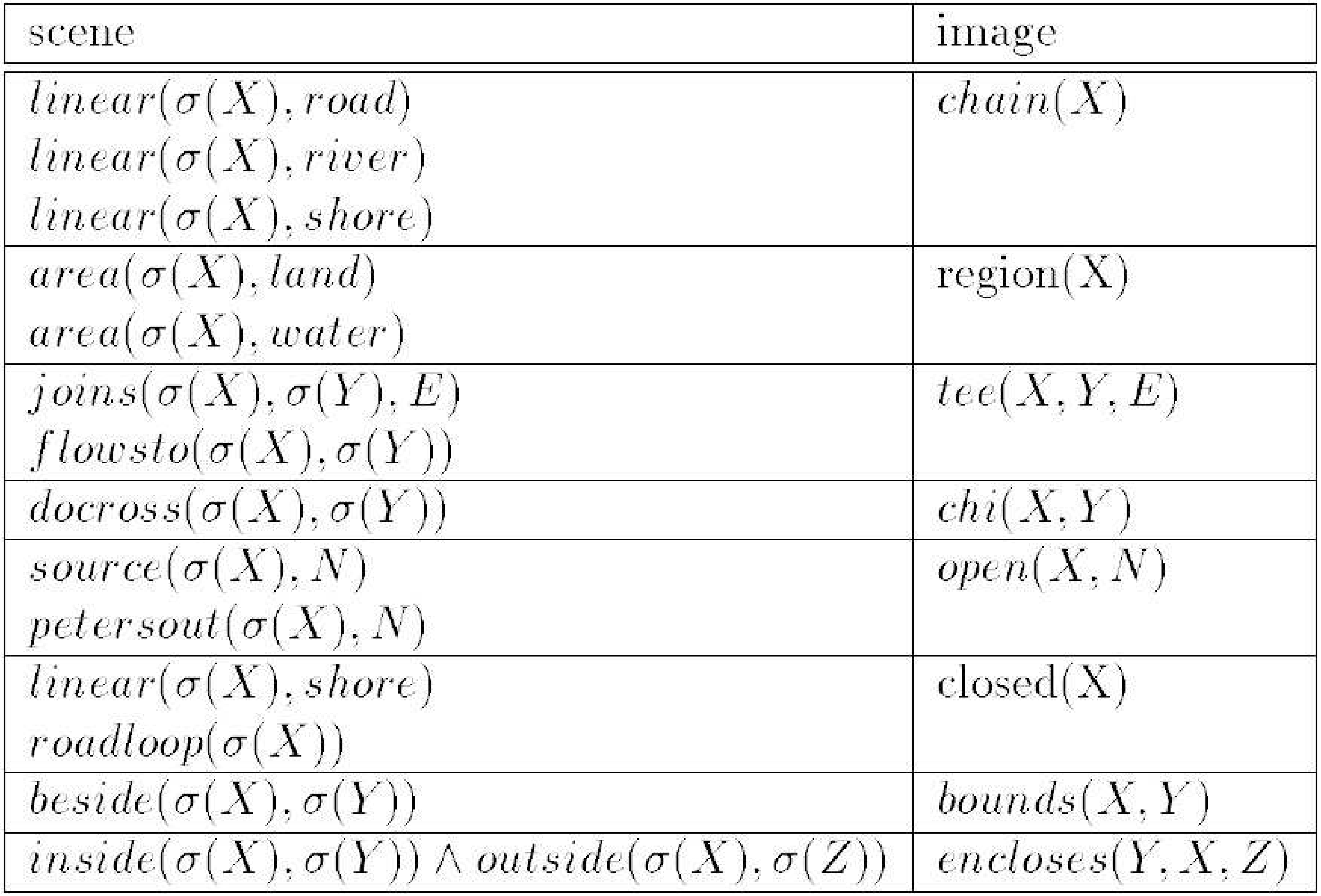
Image interpretation in AILog



Following Reiter and Mackworth, for each image object I we assume a scene object $\sigma(I)$ which it depicts.

From: Poole (1993) *Probabilistic Horn Abduction and Bayesian Networks*, Artificial Intelligence 64, pp 81-129 (in particular Section 5.3)

Image interpretation in AILog 2



scene	image
$linear(\sigma(X), road)$ $linear(\sigma(X), river)$ $linear(\sigma(X), shore)$	$chain(X)$
$area(\sigma(X), land)$ $area(\sigma(X), water)$	region(X)
$joins(\sigma(X), \sigma(Y), E)$ $flowsto(\sigma(X), \sigma(Y))$	$tee(X, Y, E)$
$docross(\sigma(X), \sigma(Y))$	$chi(X, Y)$
$source(\sigma(X), N)$ $petersout(\sigma(X), N)$	$open(X, N)$
$linear(\sigma(X), shore)$ $roadloop(\sigma(X))$	closed(X)
$beside(\sigma(X), \sigma(Y))$	$bounds(X, Y)$
$inside(\sigma(X), \sigma(Y)) \wedge outside(\sigma(X), \sigma(Z))$	$encloses(Y, X, Z)$

Image interpretation in AILog 3

region$(I) \leftarrow$ *area*$(\sigma(I), T)$.
disjoint$([$*area*$(S,$ *land*$) : 0.3,$ *area*$(S,$ *water*$) : 0.7])$.

chain$(I) \leftarrow$ *linear*$(\sigma(I), T)$.
disjoint$([$*linear*$(S,$ *road*$) : 0.2,$ *linear*$(S, river) : 0.5,$ *linear*$(S, shore) : 0.3])$.

$tee(X, Y, E) \leftarrow joins(\sigma(X), \sigma(Y), E) \wedge linear(\sigma(X), road).$
$tee(X, Y, E) \leftarrow joins(\sigma(X), \sigma(Y), E) \wedge linear(\sigma(X), river) \wedge$
$\qquad linear(\sigma(Y), road) \wedge source(\sigma(X), E).$
$tee(X, Y, E) \leftarrow linear(\sigma(X), river) \wedge canflowto(\sigma(Y)) \wedge$
$\qquad flowsto(\sigma(X), Y) \wedge mouth(\sigma(X), E).$
$canflowto(S) \leftarrow linear(S, river).$
$canflowto(S) \leftarrow linear(S, shore).$
$disjoint([joins(S, T, E) : 0.05, notjoins(S, T, E) : 0.95]).$
$disjoint([mouth(S, 0) : 0.5, mouth(S, 1) : 0.5]).$
$disjoint([flowsto(R, S) : 0.1, notflowsto(R, S) : 0.9]).$
$disjoint([source(R, 1) : 0.5, source(R, 0) : 0.5]).$

Image interpretation in AILog 3b

chi$(X, Y) \leftarrow$ *crossable*$(\sigma(X), \sigma(Y)) \wedge$ *docross*$(\sigma(X), \sigma(Y))$.

$crossable(X, Y) \leftarrow linear(X, XT) \wedge linear(Y, YT) \wedge crosstype(XT, YT).$
$crosstype(road, road).$
$crosstype(road, river).$
$crosstype(river, road).$
$crosstype(road, shore).$
$crosstype(shore, road).$
$disjoint([docross(X, Y) : 0.2, dontcross(X, Y) : 0.8]).$

Image interpretation in AILog 4

$open(X, N) \leftarrow linear(\sigma(X), river) \wedge source(\sigma(X), N).$
$open(X, N) \leftarrow linear(\sigma(X), road) \wedge petersout(\sigma(X), N).$
$disjoint([petersout(X, E) : 0.1, doesntpeterout(X, E) : 0.9]).$
$closed(X) \leftarrow linear(\sigma(X), shore).$
$closed(X) \leftarrow linear(\sigma(X), road) \wedge roadloop(\sigma(X)).$
$disjoint([roadloop(X) : 0.01, notloop(X) : 0.99]).$

Image interpretation in AILog 5

$$bounds(X,Y) \leftarrow linear(\sigma(X),XT) \wedge area(\sigma(Y),YT) \wedge$$
$$beside(\sigma(X),\sigma(Y)) \wedge possbeside(XT,YT).$$
$possbeside(road,land).$
$possbeside(river,land).$
$possbeside(shore,land).$
$possbeside(shore,water).$
$disjoint([beside(X,Y):0.1, notbeside(X,Y):0.9]).$
$disjoint([inside(X,Y):0.1, outside(X,Y):0.1, noside(X,Y):0.8]).$
$$encloses(Y,X,Z) \leftarrow outside(\sigma(X),\sigma(Z)) \wedge inside(\sigma(X),\sigma(Y)) \wedge linear(\sigma(X),XT) \wedge$$
$$area(\sigma(Y),YT) \wedge area(\sigma(Z),ZT) \wedge possreg(YT,XT,ZT).$$
$possreg(land,road,land).$
$possreg(land,shore,water).$
$possreg(water,shore,land).$

Image interpretation in AILog 6

$chain(c1) \wedge chain(c2) \wedge chain(c3) \wedge region(r1) \wedge region(r2) \wedge tee(c2, c1, 1) \wedge bounds(c2, r2) \wedge bounds(c1, r1) \wedge bounds(c1, r2) \wedge encloses(r1, c1, r2), open(c2, 0) \wedge closed(c1) \wedge open(c3, 0) \wedge tee(c3, c2, 1) \wedge bounds(c3, r2)$

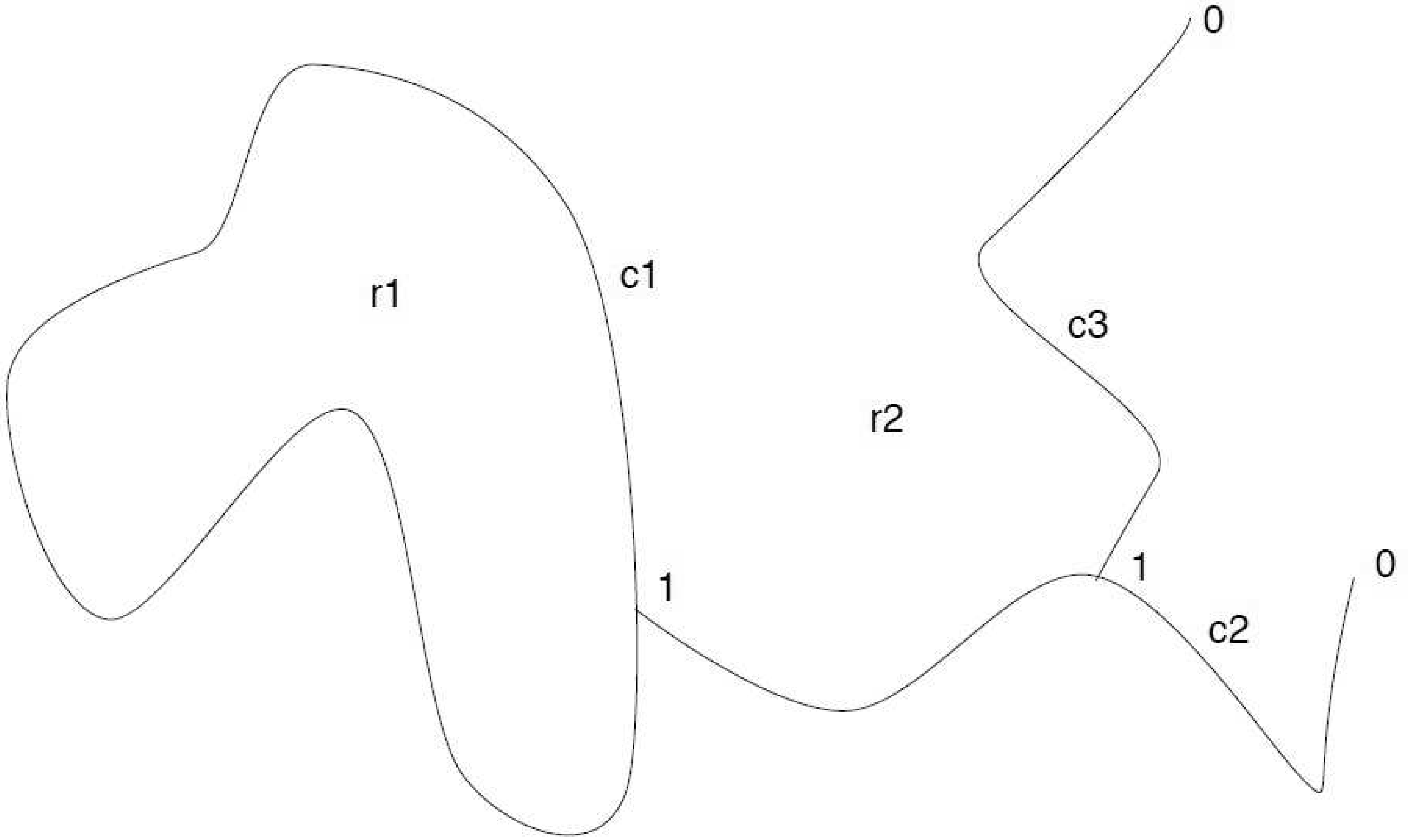


Image interpretation in AILog 7

Explanation: $\{linear(\sigma(c1), shore), linear(\sigma(c2), river), linear(\sigma(c3), river),$
$area(\sigma(r1), water), area(\sigma(r2), land), flowsto(\sigma(c2), \sigma(c1)), mouth(\sigma(c2), 1),$
$beside(\sigma(c2), \sigma(r2)), beside(\sigma(c1), \sigma(r1)), beside(\sigma(c1), \sigma(r2)), outside(\sigma(c1), \sigma(r2)),$
$inside(\sigma(c1), \sigma(r1)), source(\sigma(c2), 0), source(\sigma(c3), 0), flowsto(\sigma(c3), \sigma(c2)),$
$mouth(\sigma(c3), 1), beside(\sigma(c3), \sigma(r2))\}$
Prior $= 9.8438 \times 10^{-12}$

Explanation: $\{linear(\sigma(c1), shore), linear(\sigma(c2), river), linear(\sigma(c3), road),$
$area(\sigma(r1), water), area(\sigma(r2), land), flowsto(\sigma(c2), \sigma(c1)), mouth(\sigma(c2), 1),$
$beside(\sigma(c2), \sigma(r2)), beside(\sigma(c1), \sigma(r1)), beside(\sigma(c1), \sigma(r2)), outside(\sigma(c1), \sigma(r2)),$
$inside(\sigma(c1), \sigma(r1)), source(\sigma(c2), 0), petersout(\sigma(c3), 0), joins(\sigma(c3), \sigma(c2), 1),$
$beside(\sigma(c3), \sigma(r2))\}$
Prior $= 7.875 \times 10^{-13}$

Image interpretation in AILog 8

Explanation: $\{linear(\sigma(c1), shore), linear(\sigma(c2), road), linear(\sigma(c3), road), area(\sigma(r1), water), area(\sigma(r2), land), joins(\sigma(c2), \sigma(c1), 1), beside(\sigma(c2), \sigma(r2)), beside(\sigma(c1), \sigma(r1)), beside(\sigma(c1), \sigma(r2)), outside(\sigma(c1), \sigma(r2)), inside(\sigma(c1), \sigma(r1)), petersout(\sigma(c2), 0), petersout(\sigma(c3), 0), joins(\sigma(c3), \sigma(c2), 1), beside(\sigma(c3), \sigma(r2))\}$
Prior $= 6.3 \times 10^{-14}$

Explanation: $\{linear(\sigma(c1), road), linear(\sigma(c2), road), linear(\sigma(c3), road), area(\sigma(r1), land), area(\sigma(r2), land), joins(\sigma(c2), \sigma(c1), 1), beside(\sigma(c2), \sigma(r2)), beside(\sigma(c1), \sigma(r1)), beside(\sigma(c1), \sigma(r2)), outside(\sigma(c1), \sigma(r2)), inside(\sigma(c1), \sigma(r1)), petersout(\sigma(c2), 0), roadloop(\sigma(c1)), petersout(\sigma(c3), 0), joins(\sigma(c3), \sigma(c2), 1), beside(\sigma(c3), \sigma(r2))\}$
Prior $= 1.8 \times 10^{-16}$

Image interpretation in AILog 9

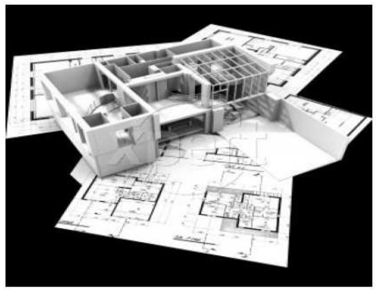
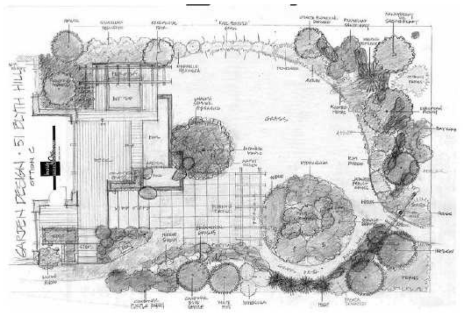
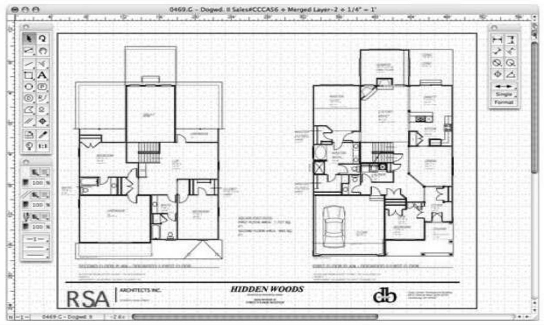
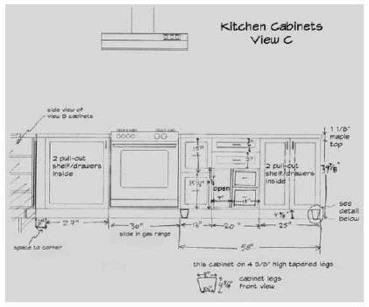
The prior probability of the image is the sum of the prior probabilities of these four explanations, namely 1.0694×10^{-11}. We can use these explanations to compute arbitrary conditional probabilities. For example,

$$P(linear(\sigma(c2), river)|image) = \frac{9.8438 \times 10^{-12} + 7.875 \times 10^{-13}}{1.0694 \times 10^{-11}} = 0.99419$$

Remarks

- Levesque's example is based on constraint satisfaction, and has much more to do with deduction
- Poole's example is based on (probabilistic) explanations, and abduction
- In general, interpretation of images is finding those scene objects that *could have generated* the image we see
- A probabilistic explanation tells us what is most likely on the picture, e.g. if *square* $\leftarrow$ *house* $: 0.5$ and *square* $\leftarrow$ *car* $: 0.2$ then it is far more likely that the image depicts a house.
- We can extend Poole's setting with *probabilistic observations*, i.e. observations that have probabilities attached, meaning that some observations are not sure (e.g. noise in sensors, or detection algorithms)
- The KB about the images becomes a probabilistic theory in AILog and we can condition on the observations found in the image

Assignment 2; task 3

In this assignment, you will choose any image domain (examples given throughout these slides, and additionally one can think of floor plans, Nijntje-images, pictures of houses and so on), and you will use AILog for image interpretation.

Conclusions

- For image interpretation, or more general vision, we need to combine lots of uncertainty (noisy sensors, uncertain knowledge) with lots of knowledge about the world (in the form of logic)
- We had already established that planning is theorem proving
- Now we can say that vision is again theorem proving, yet more focused on explanations (or; a kind of abductive diagnosis)
- Computer vision is coming back to (logical) knowledge representation now that probabilistic extensions are becoming mature