

Abduction and language processing with CHR

CHR Summer School – July 2013



Henning Christiansen, professor of Computer Science at Roskilde University, Denmark

© Henning Christiansen 2013; all rights reserved

My background

- ❖ PhD in Computer Science: syntax and semantics of programming languages, 1988
- ❖ Later interest in logic programming, as specification+implementation language and an object of study in itself
- ❖ Leading to NLP (natural language processing) and automated reasoning, in particular with Constraint Handling Rules
 - ❖ with applications in teaching, from hardcore CS students to linguists
- ❖ Recent interests include also
 - ❖ probabilistic-logic models for bioinformatics
 - ❖ formal linguistics, in particular language evolution
 - ❖ interactive installations
- ❖ Various: Organizer of several conferences and workshops, coordinator for international student exchanges (Erasmus etc.), a past as Head of CS Section and Study Director

Our principles

- ❖ Constraint store as a *knowledge base*
- ❖ CHR rules as “business logic” or “integrity constraints” \approx rules for world knowledge
- ❖ Prolog or additional CHR rules as “driver algorithm”

A motivating example . . .

A motivation example (1:3)

Consider the following Prolog program:

```
happy(X):- rich(X).  
happy(X):- professor(X), has(X,nice_students).
```

What is it supposed to mean?

Let's try it:

```
| ?- happy(henning).  
! Existence error in user:rich/1  
! procedure user:rich/1 does not exist  
! goal: user:rich(henning)
```

Another way of saying **no** :(

The problem: Prolog's *closed world* assumption

A motivation example (2:3)

Let's try with a little help from CHR:

```
:- use_module(library(chr)).  
:- chr_constraint rich/1, professor/1, has/2.  
happy(X):- rich(X).  
happy(X):- professor(X), has(X,nice_students).
```

Intuition: Make certain predicates *“open world”*.

Let's try it:

```
| ?- happy(henning).  
rich(henning) ? ;  
professor(henning),  
has(henning,nice_students) ? ;  
no
```

Looks more like it, but still not perfect . . .

A motivation example (3:3)

Adding a bit of “universal knowledge” in terms of a CHR rule:

```
:- use_module(library(chr)).  
:- chr_constraint rich/1, professor/1, has/2.  
professor(X), rich(X) ==> fail.  
happy(X):- rich(X).  
happy(X):- professor(X), has(X,nice_students).
```

Let's try it:

```
| ?- happy(henning), professor(henning).  
professor(henning),  
has(henning,nice_students) ? ;  
no
```

Thus:

- CHR constraints represent *concrete facts* about a given world.
- CHR rules represent *universal knowledge* valid in any world.

Historical background

- ❖ 1998: I found out that CHR existed and used it to implement a powerful automatic reasoning system [Christiansen, 1998]
- ❖ 1999: Visiting LMU, Munich, 1999, cooperating with Slim Abdennadher on CHR^V for abduction [Abdennadher, Christiansen, 2000]
- ❖ Around 2000: developing CHR Grammars [Christiansen, TPLP 2005]
- ❖ 2002: Visiting Verónica Dahl in Canada; replacing CHR^V by Prolog+CHR for abductive reasoning \approx Hyprolog, [Christiansen, Dahl, ICLP 2005]
- ❖ 2002 and onwards: different applications
- ❖ Since 2005 or before: applied the principle in teaching AI
- ❖ 2006-2011: Probabilistic abduction [Christiansen, 2008; Christiansen, Saleh, 2011]
- ❖ 2012: CHR adapted to knowledge bases to be used in intelligent, interactive installations.

See these and other references in the reference list.

Overview of this course

- ❖ **Abductive Reasoning with CHR**
 - ❖ Definition, implementation in CHR, applications, esp. for diagnosis
- ❖ **Language Analysis 1: With DCGs (= Prolog) plus CHR**
- ❖ **Language Analysis 2: CHR Grammars**
- ❖ **If time: iiCHR: an adaptation of CHR with persistent constraint stores shared by different agents**
 - ❖ Intended for autonomous systems and-or interactive installations

A few remarks before we start

- ❖ All example programs available on the website (*TBA*)
 - ❖ Tested in SICStus 4 and SWI
- ❖ No theorems (find them in the references), just programming :)
- ❖ No time for exercises during the course :(
- ❖ Please feel free to ask questions, to disagree even.

Part I

Abductive reasoning with CHR

Abduction????

A term due to C.S.Pierce (1839-1914); the trilogy:

- ❖ *Deduction*

- ❖ Reason “forward” in a sound way from what we know already; finding its logic consequences; i.e., nothing really new

- ❖ *Induction*

- ❖ Creating rules from example, so we can use these rules in new situations

- ❖ *Abduction*

- ❖ Figure out which currently unknown facts that can explain an observation; unsound from logical point of view ;-)

Abduction with CHR

You've seen it already!

```
:- use_module(library(chr)).  
:- chr_constraint rich/1, professor/1, has/2.  
prof(X), rich(X) ==> fail.  
happy(X):- rich(X).  
happy(X):- professor(X), has(X,nice_students).
```

```
| ?- happy(henning), professor(henning).  
professor(henning),  
has(henning,nice_students) ? ;  
no
```

In logic programming terms:

Figure out which facts should be added to the program to make the given goal succeed

Traditional definition of Abductive Logic Programming (ALP)

- ❖ An *abductive logic program* consist of
 - ❖ A number of *predicates*, some of which are called *abducibles*, Abd
 - ❖ A usual *logic program*, P , in which abducibles do not occur in the head of rules
 - ❖ A set of *integrity constraints*, IC , which are formulas that must always be true
- ❖ An abductive answer to a query Q is a set of abducible atoms Ans such that
 - ❖ $P \cup Ans \models Q$ and $P \cup Ans$ is consistent
- ❖ (It is also possible to include an answer substitution, but we ignore that)

Translating ALP into Prolog+CHR



Let us inspect our sample program:

```
:- use_module(library(chr)).  
:- chr_constraint rich/1, professor/1, has/2.  
prof(X), rich(X) ==> fail.  
happy(X):- rich(X).  
happy(X):- professor(X), has(X,nice_students).
```


Translating ALP into Prolog+CHR



Let us inspect our sample program:

```
:- use_module(library(chr)).  
:- chr_constraint rich/1, professor/1, has/2.  
prof(X), rich(X) ==> fail.  
happy(X):- rich(X).  
happy(X):- professor(X), has(X,nice_students).
```

Compare with “traditional” ALP

- ❖ Usually defined by difficult algorithms and implemented with complicated meta-interpreters; see references to work by Kowalski, Kakas & al, Decker, ...
- ❖ Our approach employs existing technology
 - ❖ in the most efficient way
 - ❖ with no meta-level overhead
 - ❖ and we can use all of Prolog and CHR (libraries, all sorts of dirty tricks)
- ❖ To my knowledge, far the most efficient implementation of ALP
- ❖ The cost? Only very limited use of negation (you can read about that)

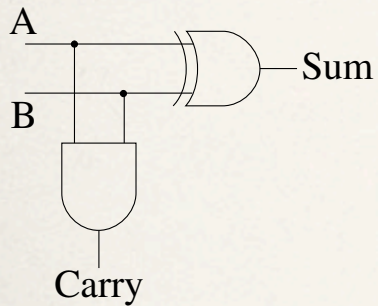
Applications of abduction

- ❖ Language interpretation
- ❖ Diagnosis
- ❖ Planning
- ❖ View update in databases

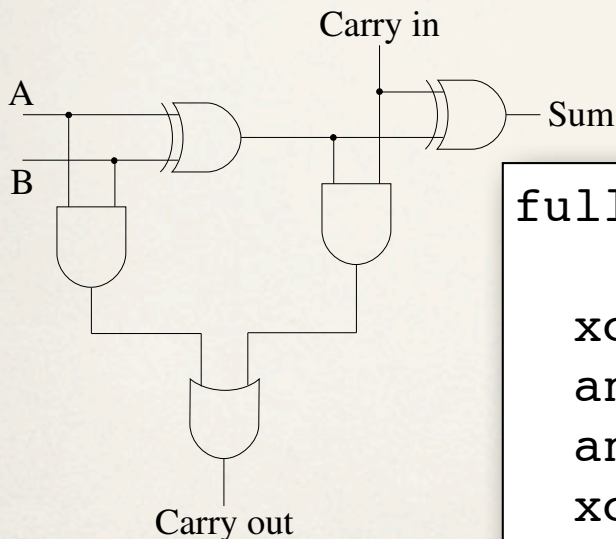
Diagnosis in Prolog+CHR

- ❖ Consider a complex system
 - ❖ we can only see it from the outside, i.e., observe *symptoms*
 - ❖ we have a *model* about how the system works inside
 - ❖ we have an idea of possible *diagnoses*, that can *explain* the symptoms
- ❖ Examples: a patient, a computer system, a car, . . .
- ❖ The problem: Given observed symptoms, suggest diagnoses
- ❖ Our example: Fault finding in logical circuits

A model of logical circuits in Prolog



```
halfadder(A, B, Carry, Sum):-  
    and(A, B, Carry),  
    xor(A, B, Sum).
```



```
fulladder(Carryin, A, B,  
          Carryout, Sum):-  
    xor(A, B, X),  
    and(A, B, Y),  
    and(X, Carryin, Z),  
    xor(Carryin, X, Sum),  
    or(Y, Z, Carryout).
```

```
not(0, 1).  
not(1, 0).  
  
and(0, 0, 0).  
and(0, 1, 0).  
and(1, 0, 0).  
and(1, 1, 1).  
  
xor(0, 0, 0).  
xor(0, 1, 1).  
xor(1, 0, 1).  
xor(1, 1, 0).  
  
or(0, 0, 0).  
or(0, 1, 1).  
or(1, 0, 1).  
or(1, 1, 1).
```

Adapt for diagnosis with CHR

Each logical gate is given an *identifier*, so we can distinguish:

```
fulladder(Carryin, A, B,  
          Carryout, Sum):-  
  xor(A, B, X, g1),  
  and(A, B, Y, g2),  
  and(X, Carryin, Z, g3),  
  xor(Carryin, X, Sum, g4),  
  or(Y, Z, Carryout, g5).
```

A gate may be *perfect* or *defect* (ok or ko) for specific inputs

```
:- chr_constraint state/3.
```

```
disturb(0,1).  
disturb(1,0).
```

```
and(A,B,X,Id):-  
  and(A,B,X),  
  state(Id,A+B,ok).  
  
and(A,B,X,Id):-  
  and(A,B,Z), disturb(Z,X),  
  state(Id,A+B,ko).  
  
or(A,B,X,Id):- . . .
```


Diagnosis may be based on different assumptions

1. *Periodic faults*, i.e., sometimes a gate works and sometimes it doesn't
2. *Consistent faults*, i.e., if something is wrong, it is always wrong
3. Consistent faults with *correct-behavior-produced-in-correct-way*

Diagnosis may be based on different assumptions

1. *Periodic faults*, i.e., sometimes a gate works and sometimes it doesn't
2. *Consistent faults*, i.e., if something is wrong, it is always wrong
3. Consistent faults with *correct-behavior-produced-in-correct-way*

```
%% No CHR rules needed
```

Let's try it:

```
| ?- fulladder(1,1,1,0,0)
state(g5,1+0,ko),
state(g4,1+0,ko),
state(g3,0+1,ok),
state(g2,1+1,ok),
state(g1,1+1,ok) ? ;
....
state(g5,0+0,ok),
state(g4,1+1,ok),
state(g3,1+1,ko),
state(g2,1+1,ko),
state(g1,1+1,ko) ? ;
```

A total of 8 solutions

Diagnosis may be based on different assumptions

1. *Periodic faults*, i.e., sometimes a gate works and sometimes it doesn't
2. *Consistent faults*, i.e., if something is wrong, it is always wrong
3. Consistent faults with *correct-behavior-produced-in-correct-way*

```
%% No CHR rules needed
```

Let's try it:

```
| ?- fulladder(0,1,1,1,0),  
      fulladder(0,1,0,0,1),  
      fulladder(0,0,1,0,1),  
      fulladder(1,0,1,1,1),  
      fulladder(1,1,1,0,0),  
      fulladder(0,0,0,0,1).
```

....

A total of 262144 solutions

Diagnosis may be based on different assumptions

1. *Periodic faults*, i.e., sometimes a gate works and sometimes it doesn't
2. *Consistent faults*, i.e., if something is wrong, it is always wrong
3. Consistent faults with *correct-behavior-produced-in-correct-way*

```
state(Id,Input,S1) \ state(Id,Input,S2) <=> S1=S2.
```

Let's try it:

```
| ?- fulladder(1,1,1,1,1).  
state(g5,1+0,ok),  
state(g4,1+0,ok),  
state(g3,0+1,ok),  
state(g2,1+1,ok),  
state(g1,1+1,ok) ? ;  
....  
state(g5,0+0,ko),  
state(g4,1+1,ko),  
state(g3,1+1,ko),  
state(g2,1+1,ko),  
state(g1,1+1,ko) ? ;
```

A total of 8 solutions

Diagnosis may be based on different assumptions

1. *Periodic faults*, i.e., sometimes a gate works and sometimes it doesn't
2. *Consistent faults*, i.e., if something is wrong, it is always wrong
3. Consistent faults with *correct-behavior-produced-in-correct-way*

```
state(Id,Input,S1) \ state(Id,Input,S2) <=> S1=S2.
```

Let's try it:

```
| ?- fulladder(0,1,1,1,0),  
      fulladder(0,1,0,0,1),  
      fulladder(0,0,1,0,1),  
      fulladder(1,0,1,1,1),  
      fulladder(1,1,1,0,0),  
      fulladder(0,0,0,0,1).
```

....

A total of 72 solutions

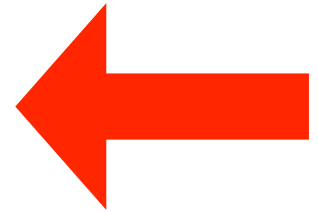
Diagnosis may be based on different assumptions

1. *Periodic faults*, i.e., sometimes a gate works and sometimes it doesn't
2. *Consistent faults*, i.e., if something is wrong, it is always wrong
3. Consistent faults with *correct-behavior-produced-in-correct-way*

```
state(Id,A,S1) \ state(Id,A,S2) <=> S1=S2.
```

Let's try it:

```
| ?- fulladder(0,1,1,1,0),  
    fulladder(0,1,0,0,1),  
    fulladder(0,0,1,0,1), !,  
    fulladder(1,0,1,1,1),  
    fulladder(1,1,1,0,0),  
    fulladder(0,0,0,0,1).
```



```
state(g1,0+0,ko),  
state(g3,0+1,ko),  
state(g4,1+0,ko),  
state(g4,1+1,ko),  
state(g5,1+1,ko), ... (rest is ok) ?
```

Only 1 solution!!

Diagnosis may be based on different assumptions: *Summary*

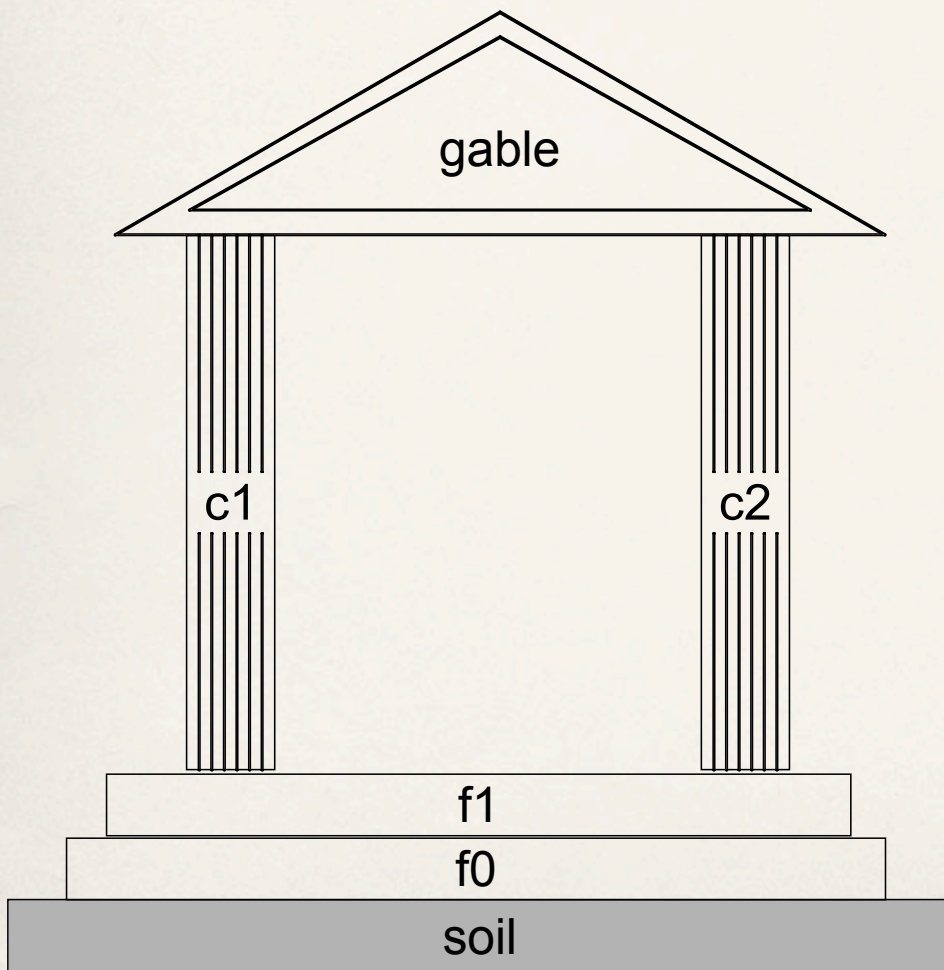
- ❖ Formulated in CHR with constraints for ok/not-ok for components
- ❖ Three alternative assumptions
 1. periodic faults
 2. consistent faults
 3. consistent faults with correct-behaviour-produced-in-correct way
- ❖ In practice, try 3, if it does not work, try 2 – and if that gives too many solutions, try to obtain more observations (i.e., test the device...)
- ❖ Problem for practical applications, say medical diagnosis, is the *lack of priority* between different diagnoses

Planning as Abduction

- ❖ **Problem:** Given a number of tasks + restrictions on the order in which they can be done.
- ❖ **Solution:** An assignment of a time point to each task so the restrictions are obeyed.
- ❖ **In our terms**
 - ❖ Abducibles (CHR constraints): Assignment of a time point to a task
 - ❖ Integrity constraints (CHR *rules*): The restrictions
 - ❖ The goal (\approx desired observation): “*The work has been done.*”

Planning as Abduction, example

Architect's drawing:



CHR rules:

```
mount(P0,Time0), mount(P1,Time1) ==>  
  supports(P0,P1), Time0 > Time1  
  | fail.
```

```
mount(P,Time0), mount(P,Time1) ==>  
  Time0 \= Time1  
  | fail.
```

Prolog facts:

```
part(gable).  
part(c1).  
...  
supports(soil,f0).  
supports(f0,f1).
```

Driver algorithm in Prolog: next slide

CHR rules:

```
mount(P0,Time0), mount(P1,Time1) ==>
    supports(P0,P1), Time0 > Time1
| fail.

mount(P,Time0), mount(P,Time1) ==>
    Time0 \= Time1
| fail.
```

Prolog facts:

```
part(gable).
part(c1).
...
supports(soil,f0).
supports(f0,f1).
```

Driver algorithm in Prolog:

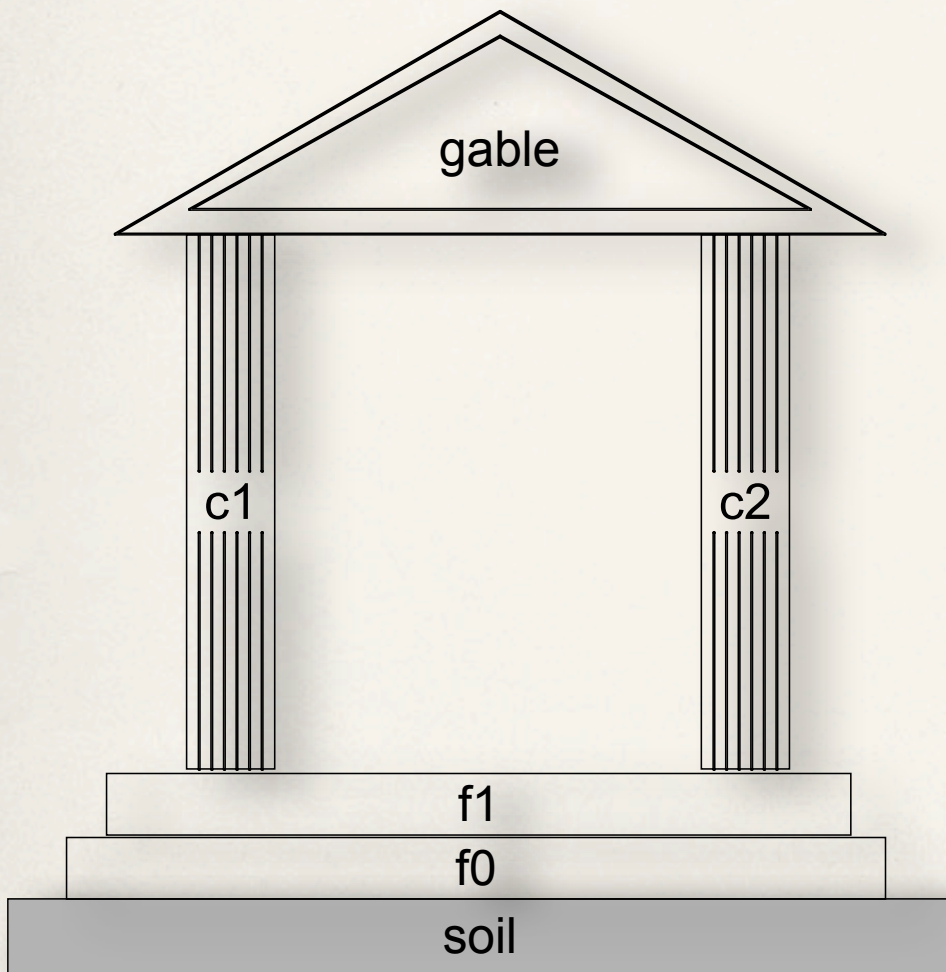
```
built:- mount(soil,0), build(1).
build(6):- !.
build(Time):-
    part(P),
    mount(P,Time),
    Time1 is Time+1,
    build(Time1).
```

```
| ?- build.
mount(gable,5),
mount(c2,4),
mount(c1,3),
mount(f1,2),
mount(f0,1),
mount(soil,0) ? ;

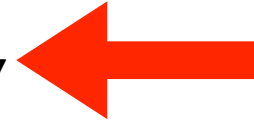
mount(gable,5),
mount(c1,4),
mount(c2,3),
mount(f1,2),
mount(f0,1),
mount(soil,0) ? ;

no
```

*Wanna see an animation
of the first solution?*



```
| ?- build.  
mount(gable,5),  
mount(c2,4),  
mount(c1,3),  
mount(f1,2),  
mount(f0,1),  
mount(soil,0) ? ;  
  
mount(gable,5),  
mount(c1,4),  
mount(c2,3),  
mount(f1,2),  
mount(f0,1),  
mount(soil,0) ? ;  
  
no
```



More on planning

- ❖ With the same technique, we can extend with
 - ❖ *Duration*, e.g., it takes 8 hours to mount a column
 - ❖ *Resources*, e.g., to mount a column, we need 1 crane and 12 workers
 - ❖ *Restrictions* += At any time, the resources in use must not exceed the maximum available (say, 2 cranes and 30 workers)
- ❖ *Your exercise (voluntary!)*: Extend the example and implement the scheme above
- ❖ *Your next exercise (difficult & voluntary)*: Extend your program so it tries to find a solution that minimizes the no. of unoccupied workers — or, alternatively, the solution that finishes the building as early as possible.

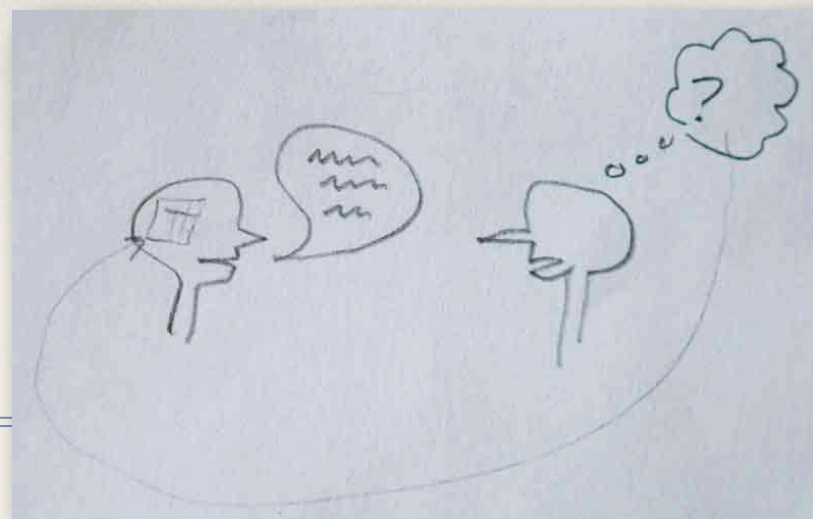
End of Part I

Abductive reasoning with CHR

Part II

Language analysis with Prolog and CHR

Overall principles



- ❖ My favourite metaphor: *“Interpretation as abduction”*
 - ❖ Jerry R. Hobbs, Mark E. Stickel, Douglas E. Appelt, Paul A. Martin: Interpretation as Abduction. Artif. Intell. 63(1-2): 69-142 (1993)
 - ❖ Also Charniac, McDermott (1985), Gabbay & al (1997), Christiansen (1993)
- ❖ We use Prolog’s Definite Clause Grammars (DCGs) extended with CHR
- ❖ Resulting method:
 - ❖ Integrates semantic and pragmatic analysis (in contrast to tradition methods)
 - ❖ A great experimental tool for students and researcher in linguistics; easy to approach and “advanced” analyses can be specified in very short time.

A short historical note

- ❖ Basic idea comes from CHR Grammars (Christiansen, 2001-2005), that we will look at later in the course
- ❖ Idea of using DCGs emerged through joint work with Verónica Dahl, 2002 and onwards....
 - ❖ Lead to the *Hyprolog* system (Christiansen, Dahl, ICLP, 2005)
 - ❖ adds a thing layer of syntactic sugar upon Prolog+CHR that supports abduction
 - ❖ and so-called *assumptions*, which another kind of tool (related to abduction, though), coming from Verónica Dahl's earlier work.
- ❖ Here we show things expressed directly in Prolog(DCG)+CHR

Overview

- ❖ Recall Definite Clause Grammars
- ❖ Adding semantics / pragmatics: Using CHR as knowledge base as we have seen already
- ❖ Examples

Definite Clause Grammars

- ❖ Syntactic sugar on top of Prolog
- ❖ System adds difference lists “behind the curtain”
- ❖ In Prolog from its very beginning
- ❖ Very popular for teaching, prototyping, and some realistic applications
- ❖ Easy to add features and “constraints”

Definite Clause Grammars

- * Syntactic sugar on top of Prolog
- * System adds difference lists “behind the curtain”
- * In Prolog from its very beginning
- * Very popular for teaching, prototyping, and some realistic applications
- * Easy to add features and “constraints”

```
s --> np(N), v(N), np(_).  
s --> np(N), is(N), [at], np(_).  
np(N) --> n(N).  
v(sing)--> [sees].  
v(plur)--> [see].  
is(sing)--> [is].  
is(plus)--> [are].  
  
n(sing) --> [peter].  
n(sing) --> [mary].  
n(sing) --> [jane].  
n(sing) --> [the,chr,summer,school].  
n(sing) --> [hennings,course].  
n(sing) --> [vacation].  
n(plur) --> n(sing), [and], n(_).
```

Definite Clause

```
s(S0,S3):- np(S0,S1,N), v(S1,S2,N), v(S2,S3).  
....  
v([sees|S0],S0,sing)
```

- * Syntactic sugar on top of Prolog

- * System adds difference lists “behind the curtain”

- * In Prolog from its very beginning

- * Very popular for teaching, prototyping, and some realistic applications

- * Easy to add features and “constraints”

```
s --> np(N), v(N), np(_).  
s --> np(N), is(N), [and], np(_).  
np(N) --> n(N).  
v(sing)--> [sees].  
v(plur)--> [see].  
is(sing)--> [is].  
is(plus)--> [are].  
  
n(sing) --> [peter].  
n(sing) --> [mary].  
n(sing) --> [jane].  
n(sing) --> [the,chr,summer,school].  
n(sing) --> [hennings,course].  
n(sing) --> [vacation].  
n(plur) --> n(sing), [and], n(_).
```


Adding semantics/pragmatics

- ❖ Traditionally:
 - ❖ “*Semantics*” = context-independent (lambda) terms
 - ❖ “*Pragmatics*” = relating “Semantics” to context, e.g., mapping variables to (identifiers of) “real worlds”
- ❖ The present approach *blurs this distinction*, which suits much better my intuition about how humans process language
- ❖ You may see this in the examples

A DGC with CHR for sem/pragm

First version: Only noting facts

```
:- chr_constraint at/2, see/2.
story --> [] ; s, ['.'], story.
s --> np(X), [sees], np(Y),
      {see(X,Y)}.
s --> np(X), [is,at], np(E),
      {at(E,X)}.
s --> np(X), [is,on,vacation],
      {at(vacation,X)}.
np(peter)      --> [peter].
np(mary)       --> [mary].
np(jane)       --> [jane].
np(chr_summer_school)
               --> [the,chr,summer,school].
np(hennings_course)
               --> [hennings,course].
np(vacation)  --> [vacation].
```

```
:- phrase(story,
          [peter,sees,mary, '.',
           peter,sees,jane, '.',
           peter,is,at,the,
             chr,summer,school, '.',
           mary,is,at,hennings,course, '.',
           jane,is,on,vacation, '.']).
at(vacation,jane),
at(hennings_course,mary),
at(chr_summer_school,peter),
see(peter,jane),
see(peter,mary) ?
```


2nd version: Adding world knowledge

```
:- chr_constraint at/2, in/2, see/2, skypes/2.
at(chr_summer_school,X) ==> in(berlin,X).
in(Loc1,X) \ in(Loc2,X) <=> Loc1=Loc2.
at(hennings_course,X) ==> at(chr_summer_school,X).
at(vacation,X) ==> in(Loc,X), diff(Loc,berlin).
see(X,Y) ==> true |
    (in(L,X), in(L,Y)
    ; in(Lx,X), in(Ly,Y), diff(Lx,Ly), skypes(X,Y)).
diff(...) <=> ... . % Homemade version of dif/1 for nicer output
% Grammar rules: Exactly the same as before
```

```
| ?- phrase(story, [mary,is,at,hennings_course,',' ] ).
at(chr_summer_school,mary),
at(hennings_course,mary),
in(berlin,mary) ?
```

2nd version: Adding world knowledge

```
:- chr_constraint at/2, in/2, see/2, skypes/2.
at(chr_summer_school,X) ==> in(berlin,X).
in(Loc1,X) \ in(Loc2,X) <=> Loc1=Loc2.
at(hennings_course,X) ==> at(chr_summer_school,X).
at(vacation,X) ==> in(Loc,X), diff(Loc,berlin).
see(X,Y) ==> true |
    (in(L,X), in(L,Y)
    ; in(Lx,X), in(Ly,Y), diff(Lx,Ly), skypes(X,Y))
diff(...) <=> ... . % Homemade version of dif/1 for nicer output
% Grammar rules: Exactly the same as before
```

```
| :- phrase(story,
    [peter,sees,mary,'.',
     peter,sees,jane,'.',
     peter,is,at,the,
       chr,summer,school,'.',
     mary,is,at,hennings,course,'.',
     jane,is,on,vacation,'.']).
```

```
at(vacation,jane),
at(chr_summer_school,mary),
at(hennings_course,mary),
at(chr_summer_school,peter),
in(_A,jane),
in(berlin,mary),
in(berlin,peter),
see(peter,jane),
see(peter,mary),
skypes(peter,jane),
diff(berlin,_A) ?
```


A realistic example: Extracting UML diagrams from Use Cases

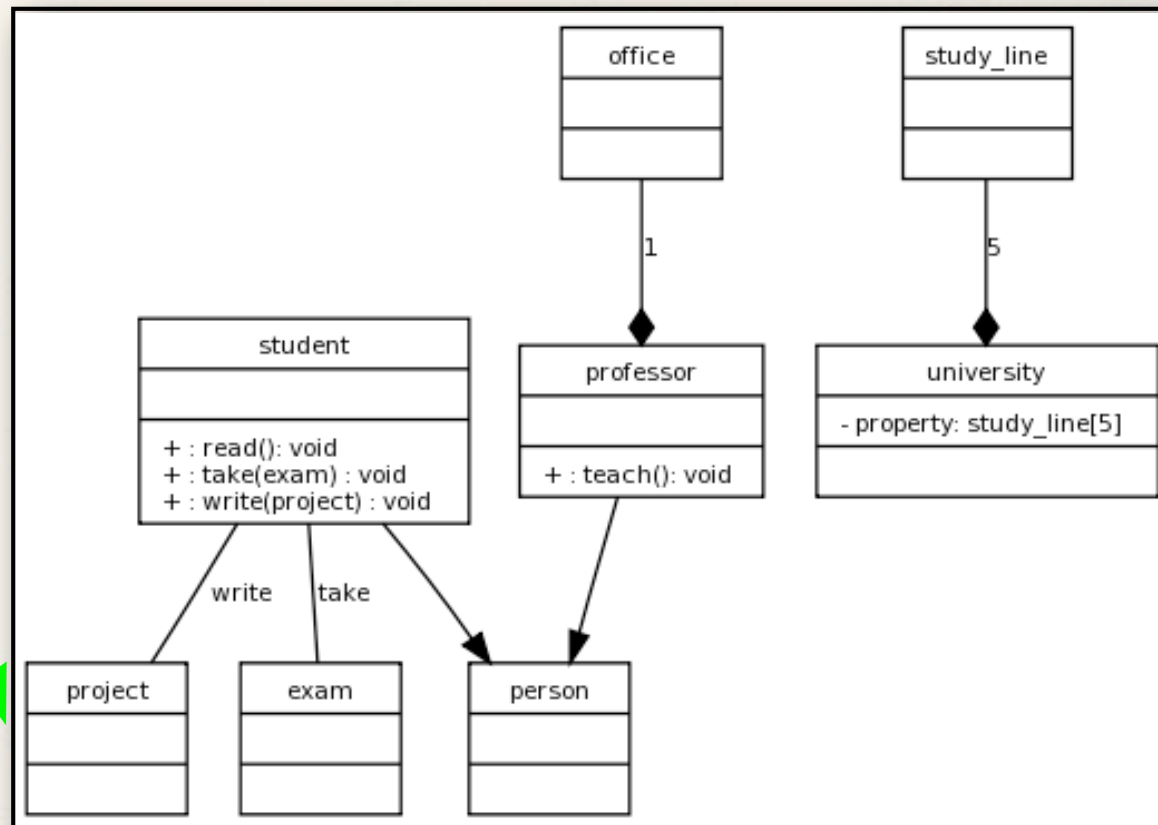
- ❖ Based on 4 week project work with two students [Christiansen, Have, Tveitane, 2007 a+b]
- ❖ Only a brief sketch; here using the full power of CHR without caring about formal details ;-)
- ❖ Use cases?? In the OOA/OOP tradition, small *stories* about the world which the system to be developed will fit it.
- ❖ According to OOA principles, UML diagrams describing classes and their property, etc., are produced manually from use cases...
- ❖ But why not do it automatically, when we have a tool such as Prolog +CHR which is perfectly suited for semantic/pragmatic analysis

Example of input and output

From uses cases:

- ❖ *The professor teaches. A student reads, writes projects and takes exams. Henning is a professor. He has an office. The university has five study lines. Students and professors are persons.*

... extract info and produce



Summary: Language analysis with DGC+CHR

- ❖ Natural and straightforward integration of semantic/pragmatic analysis with parsing
- ❖ 10^6 times easier for this purpose than any other, known tools
- ❖ DCGs (i.e., Prolog) provide parsing plus auxiliary predicates
- ❖ CHR constraint store as knowledge base; CHR rules for world knowledge

End of part II

Language analysis with Prolog and CHR

Part III

CHR Grammars

CHR Grammars, background

- ❖ Around 2000, I noticed that it was easy to write bottom-op parsers with CHR
- ❖ Experiments showed that there was much more power in this principles than expected:
 - ❖ very flexible context-dependent rules, gaps, parallel matching, ...
 - ❖ interesting treatment of ambiguity
 - ❖ having parsing to depend on “semantics”, and a lot of other stuff
- ❖ 2002: CHR Grammar system released; SICStus 3 only
- ❖ Main publication 2005 [JLP]
- ❖ 2010 or -11: New versions for SICStus 4 and SWI
- ❖ Applications: The full power of CHR Grammars still needs to be discovered

CHR Grammars, overview

- ❖ Bottom-up parsing with CHR, our principle
- ❖ A grammar notation and its translation into CHR
- ❖ What we can do in CHR Grammars, derived from the translation into CHR
 - ❖ We have squeezed as much power as possible out of CHR without caring whether it is useful (*our preferred design methodology ;-)*)

Bottom-up parsing with CHR

Encode the string as a set of constraints with *word boundaries*

"Peter likes Mary"

`token(0,1,peter), token(1,2,likes), token(2,3,mary).`

A bottom-parser that checks word / phrase boundaries

```
:- chr_constraint np/2, verb/2,
   sentence/2, token/3.

token(N0,N1,peter) ==> np(N0,N1).
token(N0,N1,mary)  ==> np(N0,N1).
token(N0,N1,likes) ==> verb(N0,N1).
np(N0,N1), verb(N1,N2), np(N2,N3)
    ==> sentence(N0,N3).
```

```
?- ... .
np(0,1),
verb(1,2),
np(2,3),
sentence(0,3),
token(0,1,peter),
token(1,2,likes),
token(2,3,mary) ?
```


A grammar notation upon CHR

Why write this?

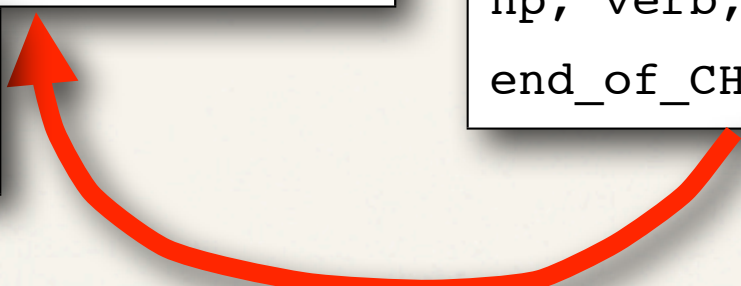
```
:- chr_constraint np/2, verb/2,  
    sentence/2, token/3.  
  
token(N0,N1,peter) ==> np(N0,N1).  
token(N0,N1,mary) ==> np(N0,N1).  
token(N0,N1,likes) ==> verb(N0,N1).  
np(N0,N1), verb(N1,N2), np(N2,N3)  
    ==> sentence(N0,N3).
```

```
?- token(0,1,peter),  
    token(1,2,likes),  
    token(2,3,mary).
```

When we would like to write this:

```
:- grammar_symbol np/0, verb/0,  
    sentence/0.  
  
[peter] ::> np.  
[mary] ::> np.  
[likes] ::> verb.  
  
np, verb, np ::> sentence.  
end_of_CHRG_source.
```

```
?- parse([peter,likes,mary]).
```

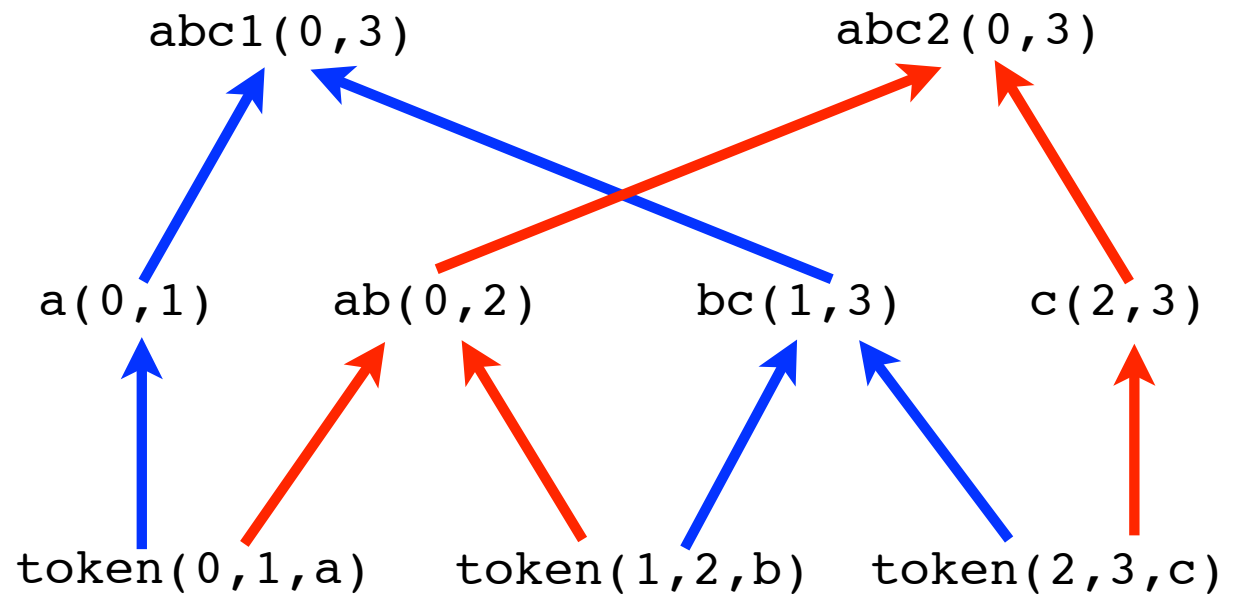


The CHR compiler
compile-on-load using `term_expansion`

Inherent handling of ambiguity

```
[a]      ::> a.  
[b,c]    ::> bc.  
[a,b]    ::> ab.  
[c]      ::> c.  
a, bc    ::> abc1.  
ab, c    ::> abc2.
```

```
| ? parse([a,b,c])
```



- ❖ I.e., all possible parses are run “in parallel”
- ❖ You can limit this by, e.g., simplification rules;
 - ❖ in the example, you would end up with only `abc1(0,3)`.
- ❖ Thus the semantics *very* procedural! (good or bad?)

What else can we put in? (1:5)

- * $::>$ translates into $==>$
- * $<:>$ translates into $<=>$
- * Order dependent syntax for simpagations

$a, !b, c <:> ac.$

translated into

$b(N1, N2) \setminus a(N0, N1), c(N2, N3) <=> ac(N0, N3).$

Obs: Mistake in
printed / online
version of slides

What else can we put in? (2:5)

Gaps in the head

```
[blip], 7...10, [blop] ::> blipblop
```

- ✧ translated into

```
token(N0,N1,blip), token(N2,N3,blop) ::>  
  N2-N1 >= 7, N2-N1 =< 10  
| blipblop(N0,N3).
```

- ✧ This may be relevant for biologic applications such as RNA folding

Obs: Mistake in
printed / online
version of slides

What else can we put in? (3:5)

Left and right context

* *left-context* $-\backslash$ *core-to-be-reduced* $/-$ *right-context* $::>$

* For example

$c1, \dots, c2 -\backslash c3, c4 /- \dots, c5 <:> c34.$

* translated into

$c1(_, N1), c2(N2, N3), c3(N3, N4), c4(N4, N5), c5(N6, _)$
 $<=> N1=<N2, N5=<N6 \mid c34(N3, N5).$

What else can we put in? (4:5)

Parallel matching

- * *one-reading-of-the-text* \$\$ *another-reading-of-the-text* ::>
- * For example: $a \text{ } \$\$ \text{ } b <:> c.$
- * translates into: $a(N0, N1), a(N0, N1) \leq => c(N0, N1).$
- * And: $a, 5 \dots 12 \text{ } \$\$ \text{ } b, c <:> d$
- * translates into:
$$a(N0, N1), b(N0, N11), c(N11, N2) \\ \leq => N1 - N2 \geq 5, N1 - N2 \leq 12 \mid d(N0, N2)$$
- * *Applications? I forgot why I included it, but it is smart, isn't it?*

What else can we put in? (5:5)

- ❖ Further equipment for abduction (see paper on CHR_G)
- ❖ All sorts of utilities and options (see online User's Guide)
- ❖ Extra-grammatical constraints in the head and body of rules (...)

Example: Simplification and context for disambiguation

An abstract and highly ambiguous grammar:

```
e, [+], e /- (['+'];[')']';[eof]) <:> e.  
e, [*], e /- (['*'];['+'];[')']';[eof]) <:> e.  
e, [^], e /- [X] <:> X \= ^ | e.  
['('], e, [')'] <:> e.  
[N] <:> integer(N) | e.
```


Example: Simplification and context for disambiguation

An abstract ~~and highly ambiguous~~ grammar:

```
e, [+], e /- (['+'];[')']';[eof]) <:> e.  
e, [*], e /- (['*'];['+'];[')']';[eof]) <:> e.  
e, [^], e /- [X] <:> X \= ^ | e.  
['('], e, [')'] <:> e.  
[N] <:> integer(N) | e.
```

Here we used LR(1) items as right context to disambiguate...
just one special case of what we can do

Example: Context used for tagger-like rules

Classify np's according to position of the verb

```
name(A) /- verb(_) <:> subject(A) .  
verb(_) -\ name(A) <:> object(A) .  
name(A), [and], subject(B) <:> subject(A+B) .  
object(A), [and], name(B) <:> object(A+B) .
```

subject(martha) subject(peter) object(paul)

↑ ↙ ↗ ↑

name(martha) /- verb(likes) [and] name(peter) /- verb(hates) -\ name(paul)

Martha likes and Peter hates Paul

A possible extension....

Obs: Mistake in
printed / online
version of slides

- ❖ Index tokens by multivel indexes, e.g.:

```
grammar_symbol( [Doc-no, Sec-no, Sent-no, Token-no] , ... )
```

- ❖ Thus long distance references can be used to control, only within same doc or everywhere, only in same section, only in same sentence...

```
m( [D, Sec, Sent, N0] , [D, Sec, Sent, N1] ) ,  
  n( [D, Sec, Sent, N1] , [D, Sec, Sent, N2] )  
==> ....
```

- ❖ Used by [van de Camp, Christiansen, 2012/3] for resolving time expressions in bibliographical texts

Summary of CHRGs

- ❖ A powerful language specification language
- ❖ A powerful language processing system
- ❖ Exemplifies how you can use CHR to implement fairly advanced, knowledge-based systems
- ❖ A compile-on-load implementation technique, you can use for other purposes
- ❖ The power of CHRGs has not been explored fully; biological applications are under consideration

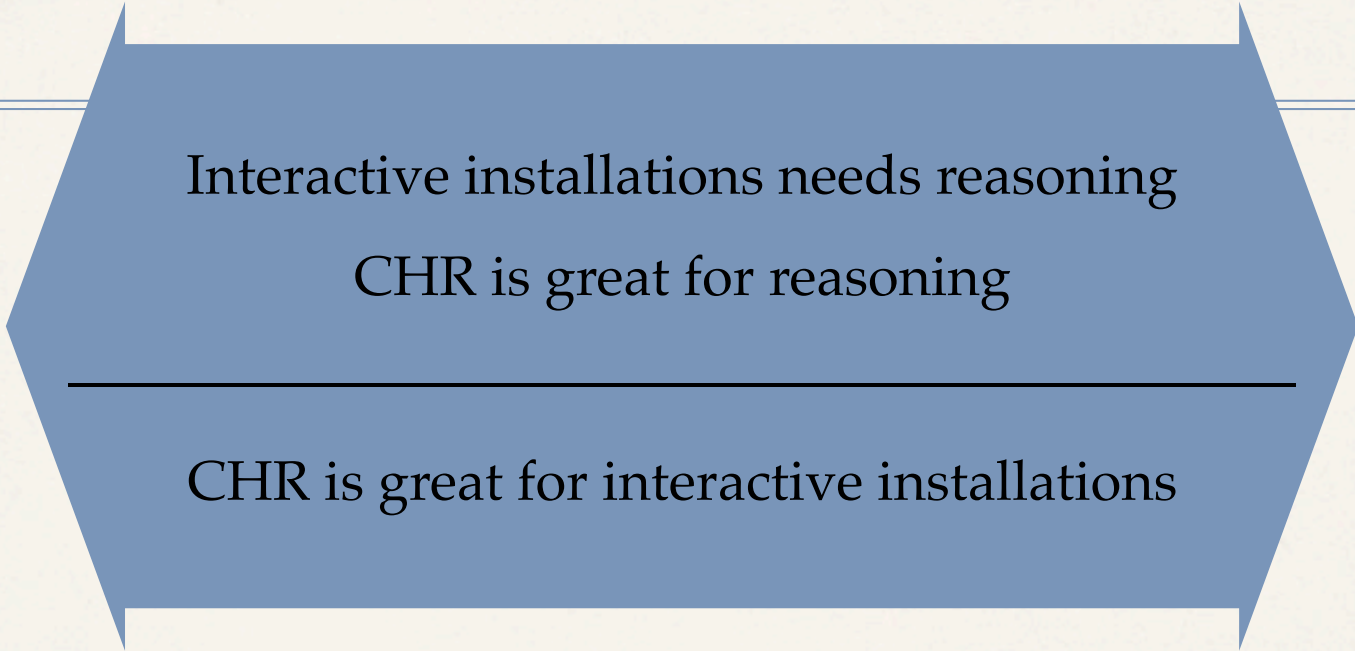
End of part III

CHR Grammars

Part IV

*ii*CHR: CHR with
persistent constraint stores
shared by different agents

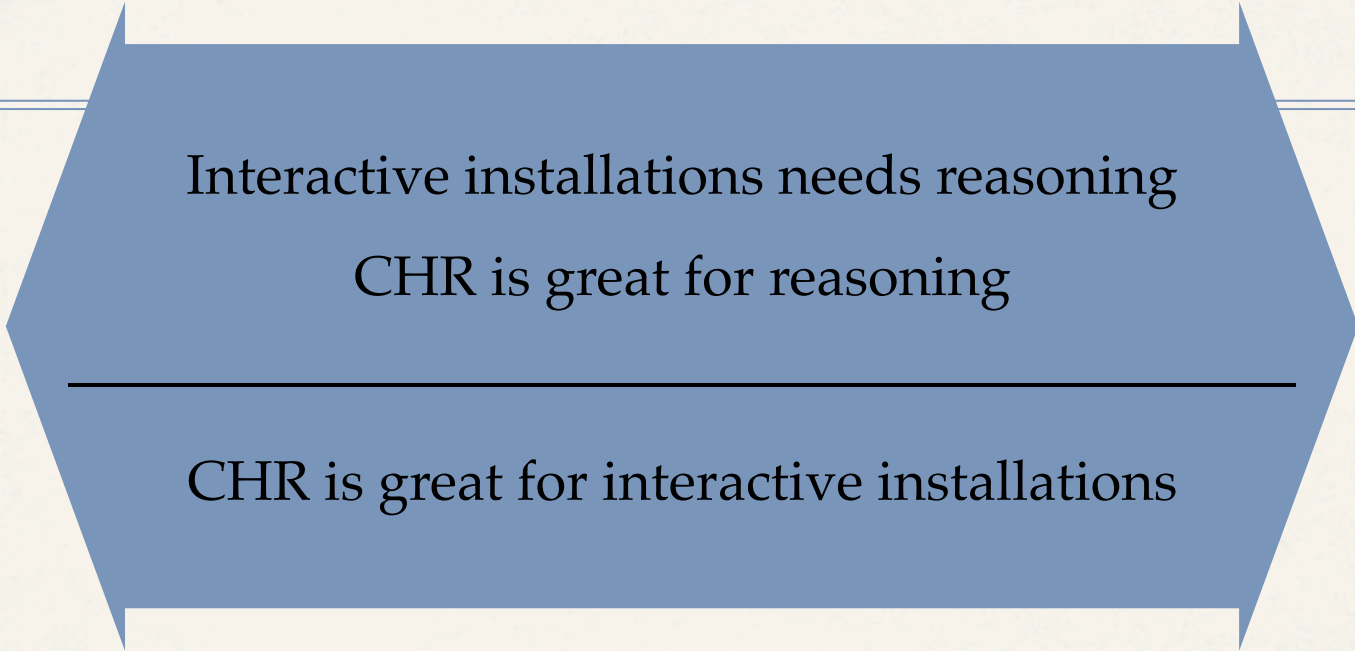
Intuition



Interactive installations needs reasoning
CHR is great for reasoning

CHR is great for interactive installations

Intuition



Interactive installations needs reasoning
CHR is great for reasoning

CHR is great for interactive installations

Problems with “Constraint store as knowledge base”:

- ❖ CHR’s constraint store disappears after each query
- ❖ Different processes need common knowledge and exchange of knowledge

iiCHR: An adaptation of CHR

Design criteria

- ❖ Programming iiCHR should be as close programming CHR as possible
- ❖ As little attention to low level details, synchronization and communication
- ❖ Interface with other programming languages and systems must be straightforward

Basic principles

- ❖ An illusion of a common and persistent constraint store using files, continually read and written by querying mechanisms
- ❖ Files formatted as *text files of Prolog facts* (that any programmer easily can access)

*ii*CHR, a first proposal

Constraint declarations define file association and behaviour

```
:- iiCHR_constraint ConstraintDecl, ... , ConstraintDecl.
```

```
ConstraintDecl ::= ConstraintPredicate / Arity * [Option, ...].
```

```
Option ::= file(Path)
```

```
| read_only | write_only | append
```

```
| locking | nowait
```

```
| time_stamped | ... and a few more
```


Query predicates (cf. JDBC)

`executeQuery(Query)`

- read all constraints from files to form initial constraint store
- execute the query in the usual way

`executeUpdate(Query)`

- as above and
- rewrite or append to files

Small variations determined by options of decl. and extra arguments of query predicates

Adaptations for the file connection

Constraints associated with files are automatically made passive

- ❖ ensures linear loading time
- ❖ fits nicely with “design patterns” for updating (later)
- ❖ can be overridden by `#active`
 - ❖ (seldom needed; for completeness only)

Constraints associated with files must be ground

- ❖ exception generated at “dump time” otherwise

“Design patterns” for knowledge base maintenance

```
:- iiCHR_constraint c/2*[file(cFile)], newC/2, deleteC/2, replaceC/2.
```

```
newC(K,V) ... <=> ... .
```

```
newC(K,V) <=> c(K,V) .
```

```
deleteC(K,V) ... <=> ... .
```

```
deleteC(K,V), c(K,V) <=> true.
```

```
replaceC(K,V) ... <=> ... .
```

```
replaceC(K,V), c(K,_) <=> c(K,V) .
```

Example:

An interactive

- ❖ Keeps track of users' activity
- ❖ Paintings tagged by themes
- ❖ Advice for next painting to see based on

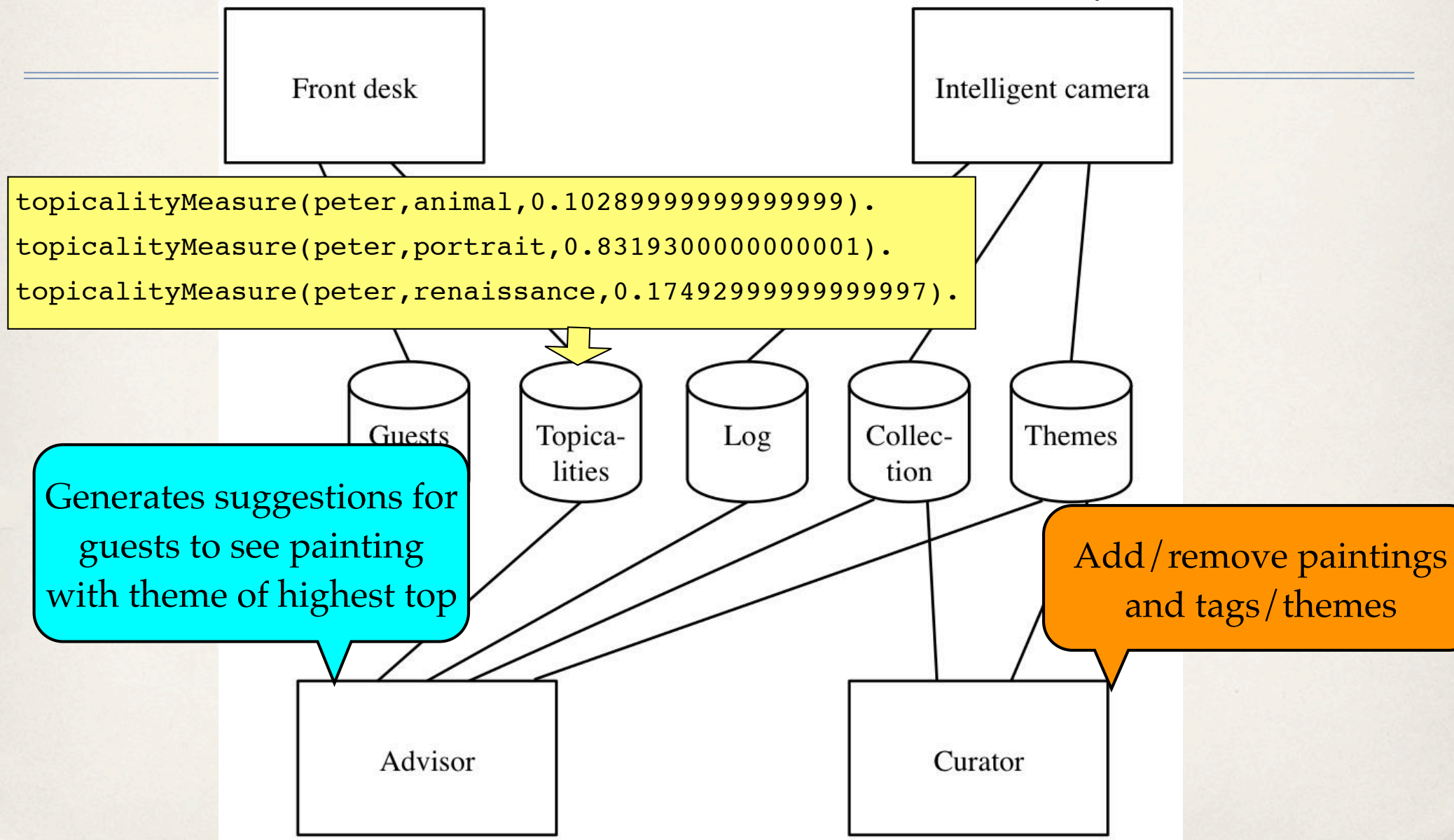
```
painting(leonardo, ladyWithEmine, animal).  
painting(leonardo, ladyWithEmine, portrait).  
painting(leonardo, ladyWithEmine, renaissance).
```



Receive guests;
clean files at depart.

Maintains topicalities (+0.3; decay
0.7) and log files

Example: Files and processes



Sample code: Intelligent Camera

```
:- iiCHR constraint
    watchedPainting/3*[file('log'),time_stamped,append],
    topicalityMeasure/3*[file('topicalities')],
    painting/3*[file(collection),read_only],
    newWatchedPainting/3.
```


Sample code: Intelligent Camera

```
?- executeUpdate(newWatchedPainting(peter,leonardo,monalisa)).
```

```
newWatchedPainting(Guest,_,_) \ topicalityMeasure(Guest,Theme,W)  
    <=> W1 is W * 0.7, topicalityMeasure(Guest,Theme,W1).
```

```
newWatchedPainting(Guest,Painter,Title), painting(Painter,Title,Theme)  
    \ topicalityMeasure(Guest,Theme,W)  
    <=> W1 is W + 0.3, topicalityMeasure(Guest,Theme,W1).
```

```
newWatchedPainting(Guest,Painter,Title), painting(Painter,Title,Theme)  
    ==> not_exists topicalityMeasure(Guest,Theme,_)  
        | topicalityMeasure(Guest,Theme,0.3).
```

```
newWatchedPainting(Guest,Painter,Title)  
    <=> watchedPainting(Guest,Painter,Title).
```

Code size

	Rules	Lines
Front desk	4	4
Curator	7	10
Intelligent cam.	4	10
Advisor	2	4

NB: tested only through n terminal windows ($n = 4$)

*ii*CHR: Conclusions (1:2)

- ✧ Established fact: CHR is a powerful language for reasoning
- ✧ *ii*CHR adds facilities for
 - ✧ different programs / processes / agents to share constraint store
 - ✧ persistent constraint stores \approx survives from query to query
 - ✧ thus bringing rule-based and declarative programming into int. install.
- ✧ Are easily interfaced with other programming languages and software components

iiCHR: Conclusions (2:2)

- ✧ Until now tested only on single PC using n open terminal windows
- ✧ Next step: in an existing 360° screen installation with kinect (or other sensors)
 - ✧ Do OS and-or Prolog programming suffice to implement, e.g., sense-think-act loops?
 - ✧ Or does iiCHR need extensions for that?
 - ✧ Is it robust enough for streaming data from (abstract) sensors?

End of part IV

*ii*CHR

Summary of the course

- ❖ CHR is for more than numbers, inequalities and stuff like that
- ❖ CHR is a *powerful knowledge representation & manipulation language*
- ❖ I have showed methods for *abductive reasoning* and *language processing*, that are
 - ❖ executed directly by the underlying CHR and Prolog systems
 - ❖ thus efficient for the right kind of problems
- ❖ I have intended that, after this course and a bit of reading, *you can*
 - ❖ use the methods as described directly
 - ❖ invent your own ways to work with knowledge and experiment with in Prolog+CHR

References

- HC. An adaptation of Constraint Handling Rules for interactive and intelligent installations. CHR Workshop 2012.
- HC. An exposition of abductive reasoning through logic programming with constraints. SLS 2012, the Scandinavian Logic Symposium, 2012.
- M. van de Camp, HC. Resolving relative time expressions in Dutch text with Constraint Handling Rules. CSLP 2012. To appear in LNCS, 2013.
- HC, B. Li. Approaching the Chinese word segmentation problem with CHR grammars. CSLP 2011. To appear in a book 2013 or -14.
- HC, A.H. Saleh. Modeling dependent events with CHRiSM for probabilistic abduction. CHR 2011.
- HC, V. Dahl. Abductive logic grammars. Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009.
- HC. Executable specifications for hypothesis-based reasoning with Prolog and Constraint Handling Rules. Journal of Applied Logic, 2009
- HC. Implementing Probabilistic Abductive Logic Programming with Constraint Handling Rules. Constraint Handling Rules, LNCS 5388, 2008.
- HC. Prioritized abduction with CHR. CHR 2008.
- HC, C.T. Have, K. Tveitane. From use cases to UML class diagrams using logic grammars and constraints. RANLP 2007.
- HC, C.T. Have, K. Tveitane. Reasoning about use cases using logic grammars and constraints. CSLP 2007.
- HC, V. Dahl, HYPROLOG: A New Logic Programming Language with Assumptions and Abduction, ICLP 2005.
- HC, V. Dahl, Meaning in Context, CONTEXT 2005, LNCS 3554.
- HC, CHR Grammars, TPLP 2005.
- S. Abdennadher, HC, An Experimental {CLP} Platform for Integrity Constraints and Abduction, FQAS 2000.
- HC, Automated Reasoning with a Constraint-Based Metainterpreter, JPL 1998.