Slide 1

Lecture 5

- Compact knowledge representation using rules
- Relation to propositional logic
- Constraint satisfaction problems
- Applications methodology
- Planning
- Extensions

Slide 2

Compact Knowledge Representation Using Rules

- Database operations
- Integrity constraints
- Rules with exceptions
- Choices

Slide 3

Database Operations

- Union:
  $u(X_1, \ldots, X_n) \leftarrow r_1(X_1, \ldots, X_n)$
  $u(X_1, \ldots, X_n) \leftarrow r_2(X_1, \ldots, X_n)$
- Intersection:
  $i(X_1, \ldots, X_n) \leftarrow r_1(X_1, \ldots, X_n), r_2(X_1, \ldots, X_n)$
- Projection:
  $father(X) \leftarrow father(X, Y)$
- Selection:
  $millionaire(X) \leftarrow assets(X, Y), (Y > 100000)$
- Join:
  $grandpa(X, Y, Z) \leftarrow father(X, Y), parent(Y, Z)$
- Complement:
  $poor(X) \leftarrow d(X), not\ millionaire(X)$

Slide 4

Integrity Constraints

- *Integrity constraints*: $\leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n$
- Can be seen as a shorthand for $f \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n$
  $f' \leftarrow not\ f', f$
- or embedded into the semantics:
  S is a stable model of a program P iff S is a stable model for the ordinary rules in P and satisfies the integrity constraints in P.

Example. $\leftarrow not\ p, s$
$\leftarrow r, not\ q, s$

A stable model $\{r, p\}$ satisfies the integrity constraints but $\{s, q\}$ does not satisfy the first integrity constraint.

Slide 5

Rules with Exceptions and Choices

- Rules with exceptions:
  $f(X) \leftarrow b(X), not\ ex\_f(X)$
  $ex\_f(X) \leftarrow \ldots$
- Choices:
  Boolean variable a: $a \leftarrow not\ \hat{a}$
  $\hat{a} \leftarrow not\ a$
  One out of $d(\cdot)$:
  $$v(X) \leftarrow d(X), not\ ov(X)$$
  $$ov(X) \leftarrow d(X), d(Y), v(Y), X \neq Y$$

Slide 6

Relation to Propositional Logic

- The built-in properties of rules (minimality and groundedness) facilitate compact knowledge representation.
- Standard constraint satisfaction tasks can be mapped to rules in a straightforward way and rules seem more expressive.

Example. Mapping propositional satisfiability to normal rules:

Consider clauses T	Translate T into a program P_T:	
$a \vee \neg b$	$\leftarrow \hat{a}, b$	$a \leftarrow not\ \hat{a}$
$\neg b \vee \neg a$	$\leftarrow a, b$	$\hat{a} \leftarrow not\ a$
$b \vee a$	$\leftarrow \hat{a}, \hat{b}$	$b \leftarrow not\ \hat{b}$
		$\hat{b} \leftarrow not\ b$

☞ T has a model iff P_T has a stable model.

Slide 7

☞ Normal rules are **more expressive** than propositional logic:

- Propositional satisfiability (**SAT**) can be mapped to normal rules in a modular way (local changes in the input lead to local changes in the output).
- There is no modular mapping from rules to **SAT** where a mapping $\mathrm{T}(\cdot)$ is modular if for each set of facts F, $P \cup F$ has a stable model iff $F \cup \mathrm{T}(P)$ is satisfiable.
  Proof: Assume there is a modular mapping $\mathrm{T}(\cdot)$.
  $P = \{q \leftarrow not\ q\}$ has no stable models.
  $\Rightarrow \mathrm{T}(P)$ is unsatisfiable.
  $\Rightarrow \{q\} \cup \mathrm{T}(P)$ is unsatisfiable
  $\Rightarrow P \cup \{q\}$ has no stable models, a contradiction.

Slide 8

Constraint Satisfaction Problems

- For each domain d: $d(c_1) \leftarrow$
  $\vdots$
  $d(c_n) \leftarrow$
- For each variable v with the domain d
  $$v(X) \leftarrow d(X), not\ ov(X)$$
  $$ov(X) \leftarrow d(X), d(Y), v(Y), X \neq Y$$
- For each constraint co: $constraint(co) \leftarrow$
  and for each allowed value combination $v_1 = c_1, \ldots v_j = c_j$ a rule
  $$sat(co) \leftarrow v_1(c_1), \ldots, v_j(c_j)$$
  and finally a rule: $\leftarrow constraint(C), not\ sat(C)$.

Slide 9

Example. Graph coloring

$$color(1) \leftarrow \ldots color(n) \leftarrow$$
$$colored(V,C) \leftarrow vertex(V), color(C), not\ othercolor(V,C)$$
$$othercolor(V,C) \leftarrow vertex(V), color(C), color(D), C \neq D, colored(V,D)$$
$$\leftarrow arc(V,U), color(C), colored(V,C), colored(U,C)$$

Slide 10

Application Methodology

- Rules express constraints on a solution set (stable model).
- This not Prolog programming (recursive data structures not used) nor constraint logic programming.
- Declarative constraint based paradigm (generate and test): Choices provide candidate solutions which are pruned by integrity constraints.
- Integrity constraints can only eliminate stable models.

Proposition. Let P be a program and IC a set of integrity constraints. Then if S is a stable model of $P \cup IC$, then S is a stable model of P.

Slide 11

- For "flat" constraints (CSP, SAT) structured representation and competitive performance.
- Particularly suitable for applications with **grounded sequences of choices:** planning, product configuration, verification of distributed systems.
- Can even challenge special purpose algorithms.

Slide 12

Example. Hamiltonian circuit

- Consider a graph G given as a database of two relations: $vertex(\cdot)$ $arc(\cdot,\cdot)$
- Program P_G^{hc} contains the two relations, the rules

$$hc(V1,V2) \leftarrow arc(V1,V2), not\ otherroute(V1,V2)$$
$$otherroute(V1,V2) \leftarrow arc(V1,V2), hc(V1,V3), not\ (V2 = V3)$$
$$otherroute(V1,V2) \leftarrow arc(V1,V2), hc(V3,V2), not\ (V1 = V3)$$
$$reached(V2) \leftarrow hc(V1,V2), reached(V1)$$
$$reached(V2) \leftarrow hc(V1,V2), initialnode(V1)$$
$$\leftarrow vertex(V), not\ reached(V)$$

  and for some vertex v the fact $initialnode(v)$.
- G has a Hamiltonian circuit iff P_G^{hc} has a stable model.

Slide 13

Applications

- Graph problems [Niemelä and Simons, JICSLP'96]
- Propositional satisfiability

  Comparison against ntab (Crawford) with hard 3-SAT problems (phase transition) [Simons, 1997]
- Planning

  Dimopoulos, Nebel, Koehler: Encoding Planning Problems in Nonmonotonic Logic Programs [ECP'97]
- Model checking

  Liu, Ramakrishnan, and Smolka: Fully Local and Efficient Evaluation of Alternating Fixed Points [TACAS'98]

Slide 14

- Reachability analysis

  Heljanko: Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets [TACAS'99]
- Product configuration

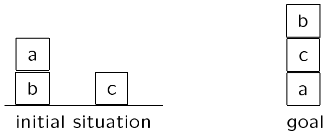
  Soininen and Niemelä: Developing a declarative rule language for applications in product configuration [PADL'99]
- Dynamic CSPs

  Soininen, Gelle and Niemelä: A Fixpoint Definition of Dynamic Constraint Satisfaction [CP'99]

Slide 15

Planning

```
(operator moveop
  (params (<X> OBJECT) (<Y> OBJECT) )
  (preconds  (clear <X>) (clear <Y>))
  (effects   (on <X> <Y>)  (clear <X>)))
```



solution:
moveop(a,table,0),
moveop(c,a,1),
moveop(b,c,2)

- Planning is **PSPACE**-complete.
- Planning with deterministic operators, with complete knowledge about the initial situation, and with an upper bound of the length of the plan is **NP**-complete.

Slide 16

Mapping Planning to Rules

- Length bound n: $time(t_0) \leftarrow$ $\quad nextstate(t_1, t_0) \leftarrow$

  $\vdots \qquad \vdots$

  $time(t_n) \leftarrow$ $\quad nextstate(t_n, t_{n-1}) \leftarrow$
- $on(X, Y, T)$: X is on Y in the situation T

  $moveop(X, Y, T)$: X is moved onto Y in the situation T
- Available blocks: $block(a) \leftarrow$

  $block(b) \leftarrow$

  $\vdots$
- The basic idea: map a planning problem to a logic program such that stable models correspond to valid plans where the goal has been reached.

Slide 17

- Initial conditions: $on(a,b,t_0) \leftarrow$
  $on(b,table,t_0) \leftarrow$
  $on(c,table,t_0) \leftarrow$
- The goal: $goal(T) \leftarrow time(T), on(b,c,T), on(c,a,T)$

  $goal \leftarrow time(T), goal(T)$

  $\leftarrow not\ goal$

  $goal(T_2) \leftarrow nextstate(T_2,T_1), goal(T_1)$
- Auxiliary defined predicates:

$on\_something(X,T) \leftarrow block(X), object(Z), time(T), on(X,Z,T)$

$covered(X,T) \leftarrow block(Z), block(X), time(T), on(Z,X,T)$

$available(table,T) \leftarrow time(T)$

$available(X,T) \leftarrow block(X), time(T), on\_something(X,T)$

$object(table) \leftarrow$

$object(X) \leftarrow block(X)$

Slide 18

- Operator preconditions:

$moveop(X,Y,T) \leftarrow time(T), block(X), object(Y), X \neq Y, on\_something(X,T), available(Y,T), not\ covered(X,T), not\ covered(Y,T), not\ blocked\_move(X,Y,T)$

- Operator effects:

$on(X,Y,T_2) \leftarrow block(X), object(Y), nextstate(T_2,T_1), moveop(X,Y,T_1)$

- Frame axioms:

$on(X,Y,T_2) \leftarrow nextstate(T_2,T_1), block(X), object(Y), on(X,Y,T_1), not\ moving(X,T_1)$

$moving(X,T) \leftarrow time(T), block(X), object(Y), moveop(X,Y,T)$

Slide 19

- Exceptions for $moveop$ I:

$blocked\_move(X,Y,T) \leftarrow block(X), object(Y), time(T), goal(T)$

$blocked\_move(X,Y,T) \leftarrow time(T), block(X), object(Y), not\ moveop(X,Y,T)$

- In addition rules for blocking conflicting operator instances are needed.
- This set depends on how much concurrency in the search of a plan is allowed.
- Computationally this seems advantageous to allow concurrency as it decreases search space explosion due to interleaving of independent operators in linear plans.

Slide 20

- Exceptions for $moveop$ II (with no concurrent actions)

(i) Only one object can be moving:

$blocked\_move(X,Y,T) \leftarrow block(X), object(Y), object(Z), time(T), moving(Z,T), X \neq Z$

(ii) Only one destination possible:

$blocked\_move(X,Y,T) \leftarrow block(X), object(Y), object(Z), time(T), destination(Z,T), Y \neq Z$

$destination(Y,T) \leftarrow time(T), block(X), object(Y), moveop(X,Y,T)$

Slide 21

- Exceptions for $moveop$ II (with concurrent actions)

  (i) A block cannot be moved to two destination:

  $$blocked\_move(X,Y,T) \leftarrow block(X), object(Y), object(Z), time(T), moveop(X,Z,T), Y \neq Z.$$

  (ii) The destination cannot be moving:

  $$blocked\_move(X,Y,T) \leftarrow block(X), object(Y), time(T), moving(Y,T)$$

  (iii) Two blocks cannot be moved onto the same block:

  $$blocked\_move(X,Y,T) \leftarrow block(X), block(Y), block(Z), time(T), moveop(Z,Y,T), X \neq Z$$

Slide 22

- Plans correspond to stable models: there is a stable model iff there is a sequence of moves that can be executed concurrently in at most n steps.
- A valid plan: facts $moveop(x,y,t)$ true in a stable model (order facts with the same situation argument in any linear order).
- Easy to integrate pruning rules:

  (i) Do not move a block from the table back to the table:

  $$\leftarrow block(X), time(T), moveop(X, table, T), on(X, table, T)$$

  (ii) Do not move a block on something and then to the table:

  $$\leftarrow nextstate(T_2, T_1), block(X), object(Y), moveop(X,Y,T_1), moveop(X, table, T_2)$$

Slide 23

Challenging blocks world planning problems solved using *Smodels* :

Problem	Number of steps	Number of ground rules	Time (s)
large.c	8	81682	(first solution) 23.2
(15 blocks)	7	72528	(no solutions) 6.0
large.d	9	128000	(first solution) 47.1
(17 blocks)	8	115110	(no solutions) 11.5
large.e	10	191622	(first solution) 101
(19 blocks)	9	174100	(no solutions) 17.3

Slide 24

New Smodels (versions 2.x)

- Choice rules: $k\{a_1, \ldots a_n\} \leftarrow Body$

  Example. SAT

  $a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_m \mapsto$
  $1\{a_1, \ldots, a_n\} \leftarrow b_1, \ldots, b_m$

- Constraint rules: $h \leftarrow k\{b_1, \ldots b_m\}$

  Example. Vertex cover of size less than K.

  $1\{v_i, v_j\} \leftarrow$ % for all edges (v_i, v_j)
  $\leftarrow K\{v_1, \ldots, v_n\}$

Slide 25

- Weight rules with weighted atoms

  Example. Bin packing

  $0\{i_1, \ldots, i_n\} \leftarrow$
  $\leftarrow \{i_1 = w_1, \ldots, i_n = w_n\} \geq W$
  $\leftarrow \{i_1 = v_1, \ldots, i_n = v_n\} < V$

- Optimization

  Maximize $\{i_1 = v_1, \ldots, i_n = v_n\}$

- Type definitions

  $1\{colored(V, C) : col(C)\}1 \leftarrow vertex(V)$

Slide 26

Example. Pigeonhole problem

$1\{in(P, H) : hole(H)\}1 \leftarrow pigeon(P)$

$\leftarrow 2\{in(P, H) : pigeon(P)\}, hole(H)$

pigeons/holes	smodels	sato	satz
8/7	0.3 s	1.4 s	0.1 s
9/8	2.4 s	45 s	0.8 s
10/9	22 s	6.7 s	7.4 s
11/10	225 s	57.8 s	77 s
12/11	2550 s	$>$ 1000min	880 s