General Game Playing (GGP)
Winter term 2013/2014

3. Game Description Language
<table>
<thead>
<tr>
<th>Date</th>
<th>What will we do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.10.2013</td>
<td>Introduction, Repetition propositional logic and FOL</td>
</tr>
<tr>
<td>29.10.2013</td>
<td>Repetition FOL / Datalog and Prolog</td>
</tr>
<tr>
<td><strong>05.11.2013</strong></td>
<td><strong>Game Description Language</strong></td>
</tr>
<tr>
<td>12.11.2013</td>
<td>Design of GDL games</td>
</tr>
<tr>
<td>19.11.2013</td>
<td>Search Algorithms 1</td>
</tr>
<tr>
<td>26.11.2013</td>
<td>Search Algorithms 2</td>
</tr>
<tr>
<td>03.12.2013</td>
<td>Incomplete information</td>
</tr>
<tr>
<td>10.12.2013</td>
<td>Fluent Calculus and Fluxplayer</td>
</tr>
<tr>
<td>17.12.2013</td>
<td><strong>Midterm competition</strong></td>
</tr>
<tr>
<td>14.01.2014</td>
<td>Meta-Gaming</td>
</tr>
<tr>
<td>21.01.2014</td>
<td>Game Theory</td>
</tr>
<tr>
<td>28.01.2014</td>
<td>?</td>
</tr>
<tr>
<td>04.02.2014</td>
<td><strong>Final Competition</strong></td>
</tr>
<tr>
<td>11.02.2014</td>
<td><strong>Exam</strong></td>
</tr>
</tbody>
</table>
The setting: Finite Synchronous Games

Finite environment
- Environment with finitely many states
- One initial state and one or more terminal states

Finite Players
- Fixed finite number of players
- Each with finitely many “actions”
- Each with one or more goal states

Synchronous Update
- All players move on all steps (some no ops)
- Environment changes only in response to moves
Prerequisite for describing these games

- We need to be able to describe:
  - Legality of moves,
  - State transitions,
  - Termination, and
  - Scoring of states.
Possible descriptions

• Finite automata
  – Problem: Huge state space
    • around 5000 in Tic-Tac-Toe
    • around $4,5 \times 10^{12}$ in Vier Gewinnt
    • Around 16! in 15-Puzzle
  – Huge memory requirements
    → not applicable
Possible descriptions

• Lists and Tables
  – Roughly the same size as finite automata
  – Not applicable

• Programs
  – Which language? Java, Lisp, C, …?
  – Difficult for analyzing the structure of games

• Logic
  – existing interpreter / compiler
  – More easy to analyze than procedural languages
  – State-of-the-art (“Game Description Language”)
Game Description Language -
Formal Foundation:
Syntactic Subtleties
Terminology

A *literal* is either an atom or a negation of an atom.

\[ \text{p(a,b), } \neg \text{p(a,b)} \]

Rules:

\[ \text{p}(x_1, \ldots, x_n) : \neg p_1(x_{11}, \ldots, x_{1n}) \& \ldots \& p_k(x_{k1}, \ldots, x_{kn}) \]

\[ \text{head} \quad \quad \quad \quad \text{subgoal} \quad \ldots \quad \text{subgoal} \quad \quad \quad \quad \text{body} \]

An expression is *ground* if and only if it contains no variables.
Negation

There are various ways to compute the value of negative rules.

In *classical negation*, a negation is true only if the negated sentence is *known* to be false (i.e. there must be rules concluding negated sentences). This is the norm in computational logic systems. In GDL, we do not have such rules.

In *negation as failure*, a negation is true if and only if the negated sentence is *not known* to be true. This is the norm in database systems.
Negation as Failure Example

Definition:

\[
\text{childless}(x) : \neg\text{person}(x), \neg\text{father}(x), \neg\text{mother}(x)
\]

Value Computation:

\[
\{\text{person}(joe), \text{person}(bill), \text{father}(joe)\}
\]

\[
\{\text{childless}(bill)\}
\]
Safety

A rule is *safe* if and only if every variable in the head appears in some positive subgoal in the body.

Safe Rule: \( r(X,Z) : - p(X,Y) \land q(Y,Z) \land \neg r(X,Y) \)

Unsafe Rule: \( r(X,Z) : - p(X,Y) \land q(Y,X) \)

Unsafe Rule: \( r(X,Z) : - p(X,Y) \land \neg q(Y,Z) \)

What could safety be good for?
Safety

A rule is *safe* if and only if every variable in the head appears in some positive subgoal in the body.

**Safe Rule:** \[ r(X,Z) : \neg p(X,Y) \land q(Y,Z) \land \neg r(X,Y) \]

**Unsafe Rule:** \[ r(X,Z) : p(X,Y) \land q(Y,X) \]

**Unsafe Rule:** \[ r(X,Z) : p(X,Y) \land \neg q(Y,Z) \]

If a nonrecursive Datalog program satisfies the safety conditions, then all the view relations defined in the program are finite.

In GDL, all rules are required to be safe!
Dependency Graph

The dependency graph for a set of rules is a directed graph in which (1) the nodes are the relations mentioned in the head and bodies of the rules and (2) there is an arc from a node \( p \) to a node \( q \) whenever \( p \) occurs with the body of a rule in which \( q \) is in the head.

\[
\begin{align*}
\text{r}(X,Y) & : \neg p(X,Y) \land q(X,Y) \\
\text{S}(X,Y) & : \neg r(X,Y) \\
\text{S}(X,Z) & : \neg r(X,Y) \land t(Y,Z) \\
\text{t}(X,Z) & : \neg s(X,Y) \land s(Y,X)
\end{align*}
\]

A set of rules is recursive if it contains a cycle. Otherwise, it is non-recursive.
Stratified Negation

The negation in a set of rules is said to be *stratified* if and only if there is no recursive cycle in the dependency graph involving a negation.

Negation that is not stratified:

\[
\begin{align*}
  r(X,Z) & : \neg p(X,Y,Z) \\
  r(X,Z) & : \neg p(X,Y,Z) \land \neg r(Y,Z)
\end{align*}
\]

Stratified Negation:

\[
\begin{align*}
  t(X,Y) & : q(X,Y) \land \neg r(X,Y) \\
  r(X,Z) & : p(X,Y) \\
  r(X,Z) & : r(X,Y) \land r(Y,Z)
\end{align*}
\]

In GDL, all negations are stratified.
Stratified Recursion

The recursion in a set of rules is said to be *stratified* if and only if every variable in a subgoal relation occurs in a subgoal with a relation at a lower stratum.

Stratified Recursion:

\[ r(X,Z) : \neg p(X,Y) \& q(Z) \& r(Y,Z) \]

Recursion that is not stratified:

\[ r(X,Z) : \neg r(X,Y) \& r(Y,Z) \]

In GDL, all recursions are stratified.
Game Description Language –
Formal Foundation: Semantics
Herbrand Universe

The *Herbrand universe* for a logic program is the set of all ground terms in the language.

Example 1:
Object Constants: \( a, b \)
Herbrand Universe: \( a, b \)

Example 2:
Object Constants: \( a \)
Unary Function Constants: \( f \)
Herbrand Universe: \( a, f(a), f(f(a)), \ldots \)
Herbrand Base

The *Herbrand base* for a logic program is the set of all ground atoms in the language.

Object Constants: a, b
Unary Relation Constant: p
Binary Relation Constant: q
Herbrand Universe: a, b

Herbrand Base:

\{p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b)\}
Herbrand Interpretations

An *interpretation* for a logic program is an arbitrary subset of the Herbrand base for the program.

Herbrand Base:
\{p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b)\}

Interpretation 1: \{p(a), q(a,b), q(b,a)\}

Interpretation 2: \{p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b)\}

Interpretation 3: {}
Herbrand Models

An interpretation $\Delta$ \textit{satisfies} a ground sentence (i.e. it is a model) under the following conditions:

$\Delta$ satisfies an atom $\varphi$ iff $\varphi \in \Delta$.

$\Delta$ satisfies $\neg \varphi$ iff $\Delta$ does \textit{not} satisfy $\varphi$.

$\Delta$ satisfies $\varphi_1 \land \ldots \land \varphi_n$ iff $\Delta$ satisfies \textit{every} $\varphi_i$.

$\Delta$ satisfies $\varphi_2 : - \varphi_1$ iff

$\Delta$ satisfies $\varphi_2$ whenever it satisfies $\varphi_1$.

An interpretation \textit{satisfies} a non-ground sentence if and only if it satisfies every ground instance of that sentence.
Multiple Models

In general, a logic program can have more than one model.

Logic Program:
\{p(a,b), q(X,Y):-p(X,Y)\}

Model 1: \{p(a,b), q(a,b)\}

Model 2: \{p(a,b), q(a,b), q(b,a)\}
Intuition for reasoning

- Start with the set $S_1$ of ground atoms (subset of Herbrand base)
- Compute $S_2$ from $S_1$, by applying all rules to the elements in $S_1$
- Repeat Step 2 until we reach a fixed point. The result is a Herbrand model

This is often called bottom-up-reasoning, in contrast to top-down reasoning
Game Description Language -

Finally!
Relevance to GDL

A GDL description is a logic program

Upshot is that each GDL program corresponds to one and only one state machine.

Initial state is the state in which all init facts are true.

Initial legal actions are defined in terms of this state.

Given initial facts and choice of legal actions, next tells us what is true in the next state.

And so forth.
Knowledge Interchange Format

Knowledge Interchange Format is a standard for programmatic exchange of knowledge represented in relational logic.

Syntax is prefix version of standard syntax.

Some operators are renamed: not, and, or.

Case-independent. Variables are prefixed with ?.

\[ r(X,Y) \leq p(X,Y) \land \neg q(Y) \]

\[ (\leq (r \ ?x \ ?y) \ (\text{and} \ (p \ ?x \ ?y) \ (\text{not} \ (q \ ?y)))) \]

Semantics is the same.
Game-Independent Vocabulary

Object Constants:

0, 1, 2, 3, ... - numbers

Relation Constants:

role(player)
init(proposition)
true(proposition)
next(proposition)
legal(player,action)
does(player,action)
goal(proposition)
terminal
Tic-Tac-Toe Vocabulary

Object constants:

white, black - players
x, o, b - marks

Function Constants:

mark(number,number) --> action
cell(number,number,mark) --> proposition
control(player) --> proposition

RelationConstants:

row(number,player)
column(number,player)
diagonal(player)
line(player)
open
Relations - role

- Defines the name of a player
- Example Tic-Tac-Toe
  - (role xplayer)
  - (role oplayer)
  - Two player: xplayer und oplayer
- Remember: In a message about moves, xplayer is always the first player and oplayer always the second player!
- role must not be head of a rule with a body
- Why?
Relations - role

- Defines the names of a player
- Example Tic-Tac-Toe
  - (role xplayer)
  - (role oplayer)
  - Two player: xplayer und oplayer
- Remember: In a message about moves, xplayer is always the first player and oplayer always the second player!
- role must not be head of a rule with a body
  - Players are fixed and cannot be changed
Relations - true

- Defines a fact that is true in the current state
- State = Set of all true facts
  - Not specified facts are false!
- Example Tic-Tac-Toe:
  - (true (control xplayer))
    - xplayer is active in current state
  - (true (cell 1 1 b)) ... (true (cell 3 1 b))
  - (true (cell 3 2 x))
  - (true (cell 3 3 o))
    - Only two rows have marks
      - (3, 2) with x
      - (3, 3) with o
- Must not be head of a rule
Relations - init

- Corresponds to true, but only for initial state
- Example Tic-Tac-Toe
  - (init (control xplayer))
    - xplayer starts
  - (init (cell 1 1 b)) ... (init (cell 3 3 b))
    - Board is empty (b = blank)
- Can only be head of a rule
- Must not depend of true, does, next, legal, goal or terminal
Relations - does

• Represents a move of a player
  – Exactly one kind of move for each player

• Necessary to compute successor state
  – Remember: Only moves are recorded/broadcasted, not whole states

• Example Tic-Tac-Toe:
  – (does xplayer (mark 1 2))
  – (does oplayer noop)
    • xplayer marks cell (1, 2) while oplayer doesn’t do anything

• Must not be head of a rule

• Must not depend of legal, goal or terminal (and vice versa)
Relations - next

- Necessary to compute successor state
  - Depends (usually) on moves
- Defines what is true in the next state
- Example Tic-Tac-Toe
  - \((<= \text{(next (control xplayer))} \text{(true (control oplayer))})\)
  - Change of active player
    - \((<= \text{(next (cell ?x ?y x))} \text{(true (cell ?x ?y b))} \text{(does xplayer (mark ?x ?y))})\)
  - Set an x at position \((?x, ?y)\)
- Must be the head of a rule
Relations - next

• Important: Frame explicitly modeled
  – If next is not true for a relation, then the relation does not hold in the next state!
  – Unchanged has to be reported explicitly!

• Example Tic-Tac-Toe:
  – (<= (next (cell ?x ?y b))
    (true (cell ?x ?y b))
    (does ?player (mark ?m ?n))
    (or (distinct ?x ?m) (distinct ?y ?n)))
  – Cell (?x, ?y) remains empty, if
    • Another cell (?m ?n) is marked and
    • ?x ≠ ?m or ?y ≠ ?n holds
Relations - legal

- Defined legal moves
  - At least one move for each player in each (non-terminal) state

- Example Tic-Tac-Toe:
  - \((\leq (\text{legal} \ ?\text{player} \ (\text{mark} \ ?x \ ?y))
    \quad (\text{true} \ (\text{cell} \ ?x \ ?y b)))
    \quad (\text{true} \ (\text{control} \ ?\text{player})))\)
  - Player \(?\text{player}\) can mark cell \((?x, \ ?y)\) only if
    - Cell empty and
    - \(?\text{player}\) is active
Relations - legal

• Remember:
  – All moves are simultaneously
  – Moves in turn are explicitly modeled

• Example Tic-Tac-Toe
  – (<= (legal xplayer noop)
    (true (control oplayer))
    • xplayer cannot do anything if oplayer’s turn
    • noop could be any other term
Relations - terminal

• Defines the terminal states

• Example Tic-Tac-Toe
  - (<= terminal
    (role ?player)
    (line ?player))
  • Player ?player has a line of 3
  - (<= terminal
    (not open))
  • There is no more free cell
Relations - goal

- Defines goals for each player
  - Associates states with scores
  - Score is an integer between 0 (bad) and 100 (excellent)

- Example Tic-Tac-Toe:
  - \( \leq (\text{goal } \text{?player} 100) \)
    \( (\text{line } \text{?player}) \)
  - Best goal for each player: Line of 3 symbols

What could be another goal in Tic-Tac-Toe?
Relations - goal

• Binary scores are possible
  – only 0 or 100
• Scores can be inverse
  – 0-sum games: (0, 100), (100, 0), or (50, 50)
• Scores can be identical for several players
  – Cooperative games
• Any other distribution of scores is possible
Tic-Tac-Toe
Roles

role(white)
role(black)
Initial State

\texttt{init(cell(1,1,b))}
\texttt{init(cell(1,2,b))}
\texttt{init(cell(1,3,b))}
\texttt{init(cell(2,1,b))}
\texttt{init(cell(2,2,b))}
\texttt{init(cell(2,3,b))}
\texttt{init(cell(3,1,b))}
\texttt{init(cell(3,2,b))}
\texttt{init(cell(3,3,b))}
\texttt{init(control(black))}
Legality

\[\text{legal}(W, \text{mark}(X,Y)) : -
\text{true}(\text{cell}(X,Y,b)) \&
\text{true}(\text{control}(W))\]

\[\text{legal}(\text{white}, \text{noop}) : -
\text{true}(\text{cell}(X,Y,b)) \&
\text{true}(\text{control}(\text{black}))\]

\[\text{legal}(\text{black}, \text{noop}) : -
\text{true}(\text{cell}(X,Y,b)) \&
\text{true}(\text{control}(\text{white}))\]
Physics

\texttt{next(cell(M,N,x)) :- does(white,mark(M,N))}

\texttt{next(cell(M,N,o)) :- does(black,mark(M,N))}

\texttt{next(cell(M,N,Z)) :- does(W,mark(M,N)) \& true(cell(M,N,Z)) \& Z\#b}

\texttt{next(cell(M,N,b)) :- does(W,mark(J,K)) \& true(cell(M,N,b)) \& (M\#J \mid N\#K)}

\texttt{next(control(white)) :- true(control(black))}

\texttt{next(control(black)) :- true(control(white))}
Supporting Concepts

row(M,W) :-
  true(cell(M,1,W)) &
  true(cell(M,2,W)) &
  true(cell(M,3,W))

diagonal(W) :-
  true(cell(1,1,W)) &
  true(cell(2,2,W)) &
  true(cell(3,3,W))

column(N,W) :-
  true(cell(1,N,W)) &
  true(cell(2,N,W)) &
  true(cell(3,N,W))

diagonal(W) :-
  true(cell(1,3,W)) &
  true(cell(2,2,W)) &
  true(cell(3,1,W))

line(W) :- row(M,W)
line(W) :- column(N,W)
line(W) :- diagonal(W)

open :- true(cell(M,N,b))
Goals and Termination

\[
\begin{align*}
goal(white, 100) & : - line(x) \\
goal(white, 50) & : ~line(x) \land ~line(o) \land ~open \\
goal(white, 0) & : - line(o) \\
\end{align*}
\]

\[
\begin{align*}
goal(black, 100) & : - line(o) \\
goal(white, 50) & : ~line(x) \land ~line(o) \land ~open \\
goal(white, 0) & : - line(x) \\
\end{align*}
\]

\[
\begin{align*}
terminal & : - line(W) \\
terminal & : ~open
\end{align*}
\]
IMPORTANT!
No Built-in Assumptions

What we see:

\[
\text{next}(\text{cell}(M,N,x)) : - \\
\text{does}(\text{white}, \text{mark}(M,N)) & \text{true}(\text{cell}(M,N,b))
\]

What the player sees:

\[
\text{next}(\text{welcoul}(M,N,himenoing)) : - \\
\text{does}(\text{himenoing}, \text{dukepse}(M,N)) & \text{true}(\text{welcoul}(M,N,lorenchise))
\]
Small Test
Buttons and Lights

- We have three buttons and three lights
- Pressing one button (e.g. Button a) => switches one light (e.g. Light p)
  - For simplicity assume that a/p, b/q, c/r
- Initial state: p off, q on, r off
- Final state: p on, q off, r on
Logical Encoding

role(robot)

init(q)

legal(robot,a)
legal(robot,b)
legal(robot,c)

next(p) :- does(robot,a) & true(p)
next(q) :- does(robot,a) & true(q)
next(r) :- does(robot,a) & true(r)

Goal(robot) :- true(p) & true(q) & true(r)
term :- true(p) & true(q) & true(r)
Logical Encoding – How about this one?

init(q)

legal(robot,a)
legal(robot,b)
legal(robot,c)

next(p) :- does(robot,a) & -true(p)
next(p) :- does(robot,b) & true(q)
next(p) :- does(robot,c) & true(p)
next(q) :- does(robot,a) & true(q)
next(q) :- does(robot,b) & true(p)
next(q) :- does(robot,c) & true(q)
next(r) :- does(robot,a) & true(r)
next(r) :- does(robot,b) & true(r)
next(r) :- does(robot,c) & true(q)

goal :- true(p) & -true(q) & true(r)
term :- true(p) & -true(q) & true(r)
GDL-II
Extension: GDL-II

- GDL does not allow describing elements of chance (e.g. rolling of a dice)
- GDL-II introduces two new keywords:
  - sees (?player, ?f)
  - random
- More information here:

- We will only use GDL, not GDL-II!
- However, a quick glance at GDL-II will not harm …
GDL-II game: What is it about?

role(jane). role(rick). role(random).

drac(7). . . . drac(eca).

ccus(7,8). . . . ccus(king,eca).

init(dnuoRgnilaed).

legal(random,lead(C,D)) $\iff$ true(dnuoRgnilaed) $\land$ drac(C) $\land$ drac(D) $\land$ distinct(C,D)

legal(random,noop) $\iff$ true(dnuoRgnitteb)

legal(R,noop) $\iff$ true(dnuoRgnilaed) $\land$ role(R) $\land$ distinct(R,random)

legal(R,nIlla) $\iff$ true(dnuoRgnitteb) $\land$ role(R) $\land$ distinct(R,random)

legal(R,dlof) $\iff$ true(dnuoRgnitteb) $\land$ role(R) $\land$ distinct(R,random)

sees(jane,yourdrac(C)) $\iff$ does(random,lead(C,D))

sees(rick,yourdrac(D)) $\iff$ does(random,lead(C,D))

sees(jane,ricksBid(B)) $\iff$ does(rick,B) $\land$ true(dnuoRgnitteb)

sees(rick,janesBid(B)) $\iff$ does(jane,B) $\land$ true(dnuoRgnitteb)

sees(jane,ricksdrac(C)) $\iff$ does(jane,nIlla) $\land$ does(rick,nIlla) $\land$ true(hasdrac(rick,C))

sees(rick,janesdrac(C)) $\iff$ does(jane,nIlla) $\land$ does(rick,nIlla) $\land$ true(hasdrac(jane,C))

next(hasdrac(jane,C)) $\iff$ does(random,lead(C,D))

next(hasdrac(rick,D)) $\iff$ does(random,lead(C,D))

next(bet(R,C,nIlla)) $\iff$ does(R,nIlla) $\land$ true(hasdrac(R,C))

next(bet(R,C,dlof)) $\iff$ does(R,dlof) $\land$ true(hasdrac(R,C))

next(dnuoRgnitteb) $\iff$ true(dnuoRgnilaed)

terminal $\iff$ ¬true(dnuoRgnilaed) $\land$ ¬true(dnuoRgnitteb)

goal(R,100) $\iff$ true(bet(R,C,nIlla)) $\land$ true(bet(S,D,nIlla)) $\land$ staeb(C,D)

goal(R, 75) $\iff$ true(bet(R,C,nIlla)) $\land$ true(bet(S,D,dlof))

goal(R, 50) $\iff$ true(bet(R,C,dlof)) $\land$ true(bet(S,D,dlof)) $\land$ distinct(R,S)

goal(R, 25) $\iff$ true(bet(R,C,dlof)) $\land$ true(bet(S,D,nIlla))

goal(R, 0) $\iff$ true(bet(R,C,nIlla)) $\land$ true(bet(S,C,nIlla)) $\land$ staeb(D,C)

staeb(C,D) $\iff$ ccus(D,C)

staeb(C,D) $\iff$ ccus(X,C) $\land$ staeb(X,D)
GDL-II game: Ok, what is it about?

role(jane). role(rick). role(random).
card(7). ... card(ace).
succ(7,8). ... succ(king,ace).
init(dealingRound).

legal(random,deal(C,D)) ≡ true(dealingRound) ∧ card(C) ∧ card(D) ∧ distinct(C,D)
legal(random,noop) ≡ true(bettingRound)
legal(R,noop) ≡ true(dealingRound) ∧ role(R) ∧ distinct(R,random)
legal(R,allIn) ≡ true(bettingRound) ∧ role(R) ∧ distinct(R,random)
legal(R,fold) ≡ true(bettingRound) ∧ role(R) ∧ distinct(R,random)

sees(jane,yourCard(C)) ≡ does(random,deal(C,D))
sees(rick,yourCard(D)) ≡ does(random,deal(C,D))
sees(jane,ricksBid(B)) ≡ does(rick,B) ∧ true(bettingRound)
sees(rick,janesBid(B)) ≡ does(jane,B) ∧ true(bettingRound)
sees(jane,ricksCard(C)) ≡ does(jane,allIn) ∧ does(rick,allIn) ∧ true(hasCard(rick,C))
sees(rick,janesCard(C)) ≡ does(jane,allIn) ∧ does(rick,allIn) ∧ true(hasCard(jane,C))

next(hasCard(jane,C)) ≡ does(random,deal(C,D))
next(hasCard(rick,D)) ≡ does(random,deal(C,D))
next(bet(R,C,allIn)) ≡ does(R,allIn) ∧ true(hasCard(R,C))
next(bet(R,C,fold)) ≡ does(R,fold) ∧ true(hasCard(R,C))
next(bettingRound) ≡ true(dealingRound)

terminal ≡ ¬true(dealingRound) ∧ ¬true(bettingRound)
goal(R, 100) ≡ true(bet(R,C,allIn)) ∧ true(bet(S,D,allIn)) ∧ beats(C,D)
goal(R, 75) ≡ true(bet(R,C,allIn)) ∧ true(bet(S,D,fold))
goal(R, 50) ≡ true(bet(R,C,fold)) ∧ true(bet(S,D,fold)) ∧ distinct(R,S)
goal(R, 25) ≡ true(bet(R,C,fold)) ∧ true(bet(S,D,allIn))
goal(R, 0) ≡ true(bet(R,C,allIn)) ∧ true(bet(S,C,allIn)) ∧ beats(D,C)
beats(C,D) ≡ succ(D,C)
beats(C,D) ≡ succ(X,C) ∧ beats(X,D)
GDL-II game: What is it about?

- Two players: jane and rick (together with anonymous dealer)
- 1\textsuperscript{st} round: each player gets a random card
  - Each player only knows his own card!
- 2\textsuperscript{nd} round: betting
  - Four outcomes:
    - All-in/all in: player with higher card wins
    - All-in/fold: payout 75/25
    - Fold/fold: draw
The Game:
Agent Communication Language
The initial idea (roughly 2006)
Game Playing Protocol

Start
  Manager send Start message to players
  Start(match, role, description, startclock, playclock)

Play
  Manager sends Play messages to players
  Play(match, actions)
  Receives plays in response

Stop
  Manager sends Stop message to players
  Stop(match, actions)
Match Begins

• Start message sent to all players in match
• Format:

  (START
   <MATCH ID>
   <ROLE>
   <GAME DESCRIPTION>
   <STARTCLOCK>
   <PLAYCLOCK> )
Moves

- After the start clock has expired the game master sends a PLAY message to all players.

  (PLAY <MATCH ID> <PRIOR MOVES>)

  where <PRIOR MOVES> is of the form

  (<move> <move> <move>)

  for a 3-player game (for example).
Misplay

Player Errors

   Illegal move
   No move before clock runs out

Manager Action

   Selects a random legal move

Rationale

   Other players must work to win
   Other players do not lose because one player errs
What makes a General Game Player?
Ingredients

• A player typically consists of the following parts:

  – Communication
    • Each player is a basic HTTP server waiting for messages from the Game Master and sending its moves as a reply.

  – Reasoning
    • Since a game description is essentially a logic program, the player has to use automatic reasoning or a logic programming system (e.g., Prolog) to infer legal moves and successor states.

  – Strategy
    • You need the communication and reasoning parts to play games. To win you need a good strategy. There are reference players available (here or here) to get you started.
Communication

• Example for HTTP Start Message:

POST / HTTP/1.0
Accept: text/delim
Sender: GAMEMASTER
Receiver: GAMEPLAYER
Content-type: text/acl
Content-length: 1194

(START MATCH.3316980891 ROBOT
  ((ROLE ROBOT) (INIT (CELL A)) ... 
  (<= TERMINAL (TRUE (GOLD A))))
  60 30)
Reasoning

• In addition to formal reasoning,

you will also need “informal” reasoning:

- Identification of
  - symmetries
  - sub-games
  - ...
Strategy

• We will discuss search/game strategies in follow-up lectures
GGP Frameworks/Implementations
Frameworks for General Game Playing

• There are several frameworks/servers for GGP:
  – ...

• We will focus on GGP Galaxy (but the other servers should also work during coding/testing, if more convenient to you)

• **Our competitions will run inside the GGP Galaxy framework!**
The GGP Galaxy project (from their webpage)

• The "GGP Galaxy" Project is an effort to bring General Game Playing to a wider audience through the use of web technologies.

• GGP Galaxy will develop enjoyable federated web-based applications for making, distributing, playing, and viewing games, and will build the tools, libraries, and common infrastructure for supporting these applications.

• All of the applications developed in GGP Galaxy will be designed to be part of a decentralized, standards-based online GGP ecosystem, inhabited by applications and games created by many different groups around the world.
GGP Galaxy: vision

GGP Galaxy Federated Components (GGP-Fed0)

**Game Resources**
Servers that provide game resources (metadata, rulesheets, stylesheets, descriptions, user interfaces, etc.) to everyone else in the ecosystem. Can be queried in a consistent, standard manner. Game resources can be consumed in a consistent, standard manner.

**Game Players**
Systems that participate in matches as players. Can be humans or computers. Should be able to indicate which games they can play, and whether they’re currently busy playing a game.

**Match Hosts**
Servers that connect to players and run matches. They can publish this information in real-time to spectator servers, which can syndicate feeds for general consumption.

- Web-Kiosk
- Standalone Game Server
- Web-Player
- Standalone Player

**Spectators**
Servers that receive notifications as matches proceed. These can display the matches for spectators to watch, archive the match, perform analysis, record player statistics, etc.

- Spectator Server
GGP Galaxy: as of now

Application Suite for the General Game Playing Project;

- A GUI-based GameKiosk
  - for playing human-vs-computer matches
- A GUI-based GamePlayer
  - for running computer players
- A GUI-based GameServer
  - for hosting matches
- A GUI-based GDLValidator
  - for validating game rulesheets
Employing GGP-Galaxy

- Download source code from
- Run ant in folder ggp-base-master
- Compiled class files are in /bin
- External JARs are in bin/external (just add them to the classpath)
GameKiosk

java -cp "bin;bin/external/Guava/guava-14.0.1.jar;bin\external/JUnit/junit-4.11.jar;bin/external/FlyingSaucer/core-renderer.jar;bin/external/JTidy/Tidy.jar" org.ggp.base.apps.kiosk.Kiosk
GamePlayer

java -cp "bin;bin/external/Guava/guava-14.0.1.jar;bin\external/JUnit/junit-4.11.jar;bin/external/FlyingSaucer/core-renderer.jar;bin/external/JTidy/Tidy.jar" org.ggp.base.apps.player.Player
GameServer

```
java -cp "bin;bin/external/Guava/guava-14.0.1.jar;bin\external/JUnit/junit-4.11.jar;bin/external/FlyingSaucer/core-renderer.jar;bin/external/JTidy/Tidy.jar" org.ggp.base.apps.server.Server
```
GDLValidator

```
java -cp "bin;bin/external/Guava/guava-14.0.1.jar;bin\external/JUnit/junit-4.11.jar;bin/external/FlyingSaucer/core-renderer.jar;bin/external/JTidy/Tidy.jar"
org.ggp.base.apps.validator.Validator
```
Notes

• core-renderer.jar can be found in external/FlyingSaucer
• base/player/GamePlayer contains a command-line player, which takes a listening port as the first parameter
  – Might be convenient to build an easy-to-test/run player
What is left?
Completeness

Game descriptions are logically incomplete in that they do not uniquely specify the moves of the players.

Every game description contains complete definitions for legality, termination, goalhood, and update in terms of the primitive moves and the does relation.

The upshot is that in every state every player can determine legality, termination, goalhood and, given a joint move, can update the state.
Playability

A game is playable if and only if every player has at least one legal move in every non-terminal state.

Note that in chess, if a player cannot move, it is a stalemate. Fortunately, this is a terminal state.

In GGP, it is guaranteed that every game is playable.
A game is strongly winnable if and only if, for some player, there is a sequence of individual moves of that player that leads to a terminating goal state for that player.

A game is weakly winnable if and only if, for every player, there is a sequence of joint moves of the players that leads to a terminating goal state for that player.

In GGP, every game is weakly winnable, and all single player games are strongly winnable.
What is next?

- 12.11.2013: Describing games in GDL
  - We will look at different (toy) games and how they are modeled in GDL
- 19.11.2013: Search heuristics I
- 26.11.2013: No lecture! (Personalversammlung)
- 03.12.2013: Search heuristics II
- 10.12.2013: Presentation of existing game players and their strategies
- 17.12.2013: Midterm competition
Acknowledgements

- Michael Genesereth (Stanford University): “Lecture 2/3: Game Description”
- Peter Kissmann (Universität Bremen): „Game Description Language (GDL)“
- David M. Kaiser: “Games, Playing, Goblin”
- Wiki at http://code.google.com/p/gqp-galaxy/