Lecture 8: Introduction to Game Logic

Eric Pacuit

ILLC, University of Amsterdam

staff.science.uva.nl/~epacuit

epacuit@science.uva.nl

Lecture Date: April 6, 2006

Caput Logic, Language and Information: Social Software

staff.science.uva.nl/~epacuit/caputLLI.html

Overview

Assuming a background in Modal Logic (at the level of the Introduction to Modal Logic course)

- Proving Correctness of Programs: From Hoare Logic to **PDL**
- From **PDL** to Game Logic
- Example: Banach-Knaster Cake Cutting Procedure
- Semantics for Game Logic
  - Neighborhood Semantics
  - "Game Theoretic" Semantics
- Main Results

Background: Hoare Logic

Background: Hoare Logic

Motivation: Formally verify the "correctness" of a program via *partial correctness assertions*:

$$\{\phi\}\alpha\{\psi\}$$

Background: Hoare Logic

Motivation: Formally verify the "correctness" of a program via *partial correctness assertions*:

$$\{\phi\}\alpha\{\psi\}$$

Intended Interpretation: If the program α begins in a state in which ϕ is true, then after α terminates (!), ψ will be true.

Background: Hoare Logic

Motivation: Formally verify the "correctness" of a program via *partial correctness assertions*:

$$\{\phi\}\alpha\{\psi\}$$

Intended Interpretation: If the program α begins in a state in which ϕ is true, then after α terminates (!), ψ will be true.

C. A. R. Hoare. *An Axiomatic Basis for Computer Programming.*. Comm. Assoc. Comput. Mach. 1969.

Background: Hoare Logic

Main Rules:

Background: Hoare Logic

Main Rules:

Assignment Rule: $\{\phi[x/e]\}\ x := e\ \{\phi\}$

Background: Hoare Logic

Main Rules:

Assignment Rule: $\{\phi[x/e]\}\ x := e\ \{\phi\}$

Composition Rule: $$\frac{\{\phi\}\ \alpha\ \{\sigma\} \qquad \{\sigma\}\ \beta\ \{\psi\}}{\{\phi\}\ \alpha;\beta\ \{\psi\}}$$

Background: Hoare Logic

Main Rules:

Assignment Rule: $\{\phi[x/e]\}\ x := e\ \{\phi\}$

Composition Rule: $$\frac{\{\phi\}\ \alpha\ \{\sigma\} \qquad \{\sigma\}\ \beta\ \{\psi\}}{\{\phi\}\ \alpha;\beta\ \{\psi\}}$$

Conditional Rule: $$\frac{\{\phi \wedge \sigma\}\ \alpha\ \{\psi\} \qquad \{\phi \wedge \neg\sigma\}\ \beta\ \{\psi\}}{\{\phi\}\ \mathbf{if}\ \sigma\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta\ \{\psi\}}$$

Background: Hoare Logic

Main Rules:

Assignment Rule: $\{\phi[x/e]\}\ x := e\ \{\phi\}$

Composition Rule: $$\frac{\{\phi\}\ \alpha\ \{\sigma\} \qquad \{\sigma\}\ \beta\ \{\psi\}}{\{\phi\}\ \alpha;\beta\ \{\psi\}}$$

Conditional Rule: $$\frac{\{\phi \wedge \sigma\}\ \alpha\ \{\psi\} \qquad \{\phi \wedge \neg\sigma\}\ \beta\ \{\psi\}}{\{\phi\}\ \textbf{if}\ \sigma\ \textbf{then}\ \alpha\ \textbf{else}\ \beta\ \{\psi\}}$$

While Rule: $$\frac{\{\phi \wedge \sigma\}\ \alpha\ \{\phi\}}{\{\phi\}\ \textbf{while}\ \sigma\ \textbf{do}\ \alpha\ \{\phi \wedge \neg\sigma\}}$$

Example: Euclid's Algorithm

```
x := u;
y := v;
while x ≠ y do
    if x < y then
        y := y − x;
    else
        x := x − y;
```

Let $\phi := \gcd(x, y) = \gcd(u, v)$

Example: Euclid's Algorithm

```
x := u;
y := v;
while x ≠ y do
   if x < y then
      y := y − x;
   else
      x := x − y;
```

Let α be the inner `if` statement.

Example: Euclid's Algorithm

```
x := u;
y := v;
while x ≠ y do
    if x < y then
        y := y − x;
    else
        x := x − y;
```

Let α be the inner `if` statement.

Then $\{\gcd(x, y) = \gcd(u, v)\}\ \alpha\ \{\gcd(x, y) = \gcd(u, v)\}$

Example: Euclid's Algorithm

```
x := u;
y := v;
while x ≠ y do
    if x < y then
        y := y − x;
    else
        x := x − y;
```

Hence by the `while`-rule (using a "weakening rule")

$$\frac{\{(\mathrm{gcd}(x,y) = \mathrm{gcd}(u,v)) \wedge (x \neq y)\}\ \alpha\ \{\mathrm{gcd}(x,y) = \mathrm{gcd}(u,v))\}}{\{\mathrm{gcd}(x,y) = \mathrm{gcd}(u,v)\}\ \textbf{while}\ \sigma\ \textbf{do}\ \alpha\ \{(\mathrm{gcd}(x,y) = \mathrm{gcd}(u,v)) \wedge \neg(x \neq y)\}}$$

Background: Propositional Dynamic Logic

Let P be a set of atomic programs and At a set of atomic propositions.

Formulas of **PDL** have the following syntactic form:

$$\phi := p \mid \bot \mid \neg\phi \mid \phi \vee \psi \mid [\alpha]\phi$$

$$\alpha := a \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^* \mid \phi?$$

where $p \in \mathsf{At}$ and $a \in \mathsf{P}$.

Background: Propositional Dynamic Logic

Let P be a set of atomic programs and At a set of atomic propositions.

Formulas of **PDL** have the following syntactic form:

$$\phi := p \mid \bot \mid \neg\phi \mid \phi \vee \psi \mid [\alpha]\phi$$

$$\alpha := a \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^* \mid \phi?$$

where $p \in \mathsf{At}$ and $a \in \mathsf{P}$.

$\{\phi\}\ \alpha\ \{\psi\}$ **is replaced with** $\phi \rightarrow [\alpha]\psi$

Background: Propositional Dynamic Logic

Semantics: $\mathcal{M} = \langle W, \{R_a \mid a \in \mathsf{P}\}, V\rangle$ where for each $a \in \mathsf{P}$, $R_a \subseteq W \times W$ and $V : \mathsf{At} \rightarrow 2^W$

- $R_{\alpha\cup\beta} := R_\alpha \cup R_\beta$
- $R_{\alpha;\beta} := R_\alpha \circ R_\beta$
- $R_{\alpha^*} := \cup_{n\geq 0} R_\alpha^n$
- $R_{\phi?} = \{(w,w) \mid \mathcal{M}, w \models \phi\}$

$$\mathcal{M}, w \models [\alpha]\phi \text{ iff for each } v, \text{ if } wR_\alpha v \text{ then } \mathcal{M}, v \models \phi$$

Background: Propositional Dynamic Logic

Segerberg Axioms:

1. Axioms of propositional logic
2. $[\alpha](\phi \to \psi) \to ([\alpha]\phi \to [\alpha]\psi)$
3. $[\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$
4. $[\alpha; \beta]\phi \leftrightarrow [\alpha][\beta]\phi$
5. $[\psi?]\phi \leftrightarrow (\psi \to \phi)$
6. $\phi \wedge [\alpha][\alpha^*]\phi \leftrightarrow [\alpha^*]\phi$
7. $\phi \wedge [\alpha^*](\phi \to [\alpha]\phi) \to [\alpha^*]\phi$
8. Modus Ponens and Necessitation (for each program α)

Background: Propositional Dynamic Logic

Segerberg Axioms:

1. Axioms of propositional logic
2. $[\alpha](\phi \to \psi) \to ([\alpha]\phi \to [\alpha]\psi)$
3. $[\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$
4. $[\alpha;\beta]\phi \leftrightarrow [\alpha][\beta]\phi$
5. $[\psi?]\phi \leftrightarrow (\psi \to \phi)$
6. $\phi \wedge [\alpha][\alpha^*]\phi \leftrightarrow [\alpha^*]\phi$ (Fixed-Point Axiom)
7. $\phi \wedge [\alpha^*](\phi \to [\alpha]\phi) \to [\alpha^*]\phi$ (Induction Axiom)
8. Modus Ponens and Necessitation (for each program α)

Background: Propositional Dynamic Logic

Some Results

Theorem (Parikh, Kozen and Parikh) **PDL** is sound and weakly complete with respect to the Segerberg Axioms.

Theorem The satisfiability problem for **PDL** is decidable (EXPTIME-Complete).

D. Kozen and R. Parikh. *A*. .

D. Harel, D. Kozen and Tiuryn. *Dynamic Logic*. .

K. Apt. *10 Years of Hoare Logic*. .

From PDL to Game Logic

Game Logic (**GL**) was introduced by Rohit Parikh in

R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

From PDL to Game Logic

Game Logic (GL) was introduced by Rohit Parikh in

R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

Main Idea:

In **PDL**: $w \models \langle \pi \rangle \phi$: there is a run of the program π starting in state w that ends in a state where ϕ is true.

The programs in **PDL** can be thought of as *single player games*.

From PDL to Game Logic

Game Logic (**GL**) was introduced by Rohit Parikh in

R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

Main Idea:

In **PDL**: $w \models \langle\pi\rangle\phi$: there is a run of the program π starting in state w that ends in a state where ϕ is true.

The programs in **PDL** can be thought of as *single player games*.

Game Logic generalized **PDL** by considering two players:

In **GL**: $w \models \langle\gamma\rangle\phi$: Angel has a **strategy** in the game γ to ensure that the game ends in a state where ϕ is true.

From PDL to Game Logic

Consequences of two players:

From PDL to Game Logic

Consequences of two players:

$\langle\gamma\rangle\phi$: Angel has a strategy in γ to ensure ϕ is true

$[\gamma]\phi$: Demon has a strategy in γ to ensure ϕ is true

From PDL to Game Logic

Consequences of two players:

$\langle\gamma\rangle\phi$: Angel has a strategy in γ to ensure ϕ is true

$[\gamma]\phi$: Demon has a strategy in γ to ensure ϕ is true

Either Angel or Demon can win: $\langle\gamma\rangle\phi \vee [\gamma]\neg\phi$

From PDL to Game Logic

Consequences of two players:

$\langle\gamma\rangle\phi$: Angel has a strategy in γ to ensure ϕ is true

$[\gamma]\phi$: Demon has a strategy in γ to ensure ϕ is true

Either Angel or Demon can win: $\langle\gamma\rangle\phi \vee [\gamma]\neg\phi$

But not both: $\neg(\langle\gamma\rangle\phi \wedge [\gamma]\neg\phi)$

From PDL to Game Logic

Consequences of two players:

$\langle\gamma\rangle\phi$: Angel has a strategy in γ to ensure ϕ is true

$[\gamma]\phi$: Demon has a strategy in γ to ensure ϕ is true

Either Angel or Demon can win: $\langle\gamma\rangle\phi \vee [\gamma]\neg\phi$

But not both: $\neg(\langle\gamma\rangle\phi \wedge [\gamma]\neg\phi)$

Thus, $[\gamma]\phi \leftrightarrow \neg\langle\gamma\rangle\neg\phi$ is a valid principle

From PDL to Game Logic

Consequences of two players:

$\langle\gamma\rangle\phi$: Angel has a strategy in γ to ensure ϕ is true

$[\gamma]\phi$: Demon has a strategy in γ to ensure ϕ is true

Either Angel or Demon can win: $\langle\gamma\rangle\phi \vee [\gamma]\neg\phi$

But not both: $\neg(\langle\gamma\rangle\phi \wedge [\gamma]\neg\phi)$

Thus, $[\gamma]\phi \leftrightarrow \neg\langle\gamma\rangle\neg\phi$ is a valid principle

However, $[\gamma]\phi \wedge [\gamma]\psi \rightarrow [\gamma](\phi \wedge \psi)$ is **not** a valid principle

From PDL to Game Logic

Reinterpret operations and invent new ones:

- $?\phi$: Check whether ϕ currently holds

From PDL to Game Logic

Reinterpret operations and invent new ones:

- $?\phi$: Check whether ϕ currently holds
- $\gamma_1; \gamma_2$: First play γ_1 then γ_2

From PDL to Game Logic

Reinterpret operations and invent new ones:

- $?\phi$: Check whether ϕ currently holds
- $\gamma_1;\gamma_2$: First play γ_1 then γ_2
- $\gamma_1 \cup \gamma_2$: Angel choose between γ_1 and γ_2

From PDL to Game Logic

Reinterpret operations and invent new ones:

- $?\phi$: Check whether ϕ currently holds
- $\gamma_1;\gamma_2$: First play γ_1 then γ_2
- $\gamma_1 \cup \gamma_2$: Angel choose between γ_1 and γ_2
- γ^*: Angel can choose how often to play γ (possibly not at all); each time she has played γ, she can decide whether to play it again or not.

From PDL to Game Logic

Reinterpret operations and invent new ones:

- $?\phi$: Check whether ϕ currently holds
- $\gamma_1;\gamma_2$: First play γ_1 then γ_2
- $\gamma_1 \cup \gamma_2$: Angel choose between γ_1 and γ_2
- γ^*: Angel can choose how often to play γ (possibly not at all); each time she has played γ, she can decide whether to play it again or not.
- γ^d: Switch roles, then play γ

From PDL to Game Logic

Reinterpret operations and invent new ones:

- $?\phi$: Check whether ϕ currently holds
- $\gamma_1 ; \gamma_2$: First play γ_1 then γ_2
- $\gamma_1 \cup \gamma_2$: Angel choose between γ_1 and γ_2
- γ^*: Angel can choose how often to play γ (possibly not at all); each time she has played γ, she can decide whether to play it again or not.
- γ^d: Switch roles, then play γ
- $\gamma_1 \cap \gamma_2 := (\gamma_1^d \cup \gamma_2^d)^d$: Demon chooses between γ_1 and γ_2

From PDL to Game Logic

Reinterpret operations and invent new ones:

- $?\phi$: Check whether ϕ currently holds
- $\gamma_1;\gamma_2$: First play γ_1 then γ_2
- $\gamma_1 \cup \gamma_2$: Angel choose between γ_1 and γ_2
- γ^*: Angel can choose how often to play γ (possibly not at all); each time she has played γ, she can decide whether to play it again or not.
- γ^d: Switch roles, then play γ
- $\gamma_1 \cap \gamma_2 := (\gamma_1^d \cup \gamma_2^d)^d$: Demon chooses between γ_1 and γ_2
- $\gamma^x := ((\gamma^d)^*)^d$: Demon can choose how often to play γ (possibly not at all); each time he has played γ, he can decide whether to play it again or not.

Syntax

Let Γ_0 be a set of atomic games and At a set of atomic propositions. Then formulas of Game Logic are defined inductively as follows:

$$\begin{aligned} \gamma &:= g \mid \phi? \mid \gamma;\gamma \mid \gamma \cup \gamma \mid \gamma^* \mid \gamma^d \\ \phi &:= \bot \mid p \mid \neg\phi \mid \phi \vee \phi \mid \langle\gamma\rangle\phi \mid [\gamma]\phi \end{aligned}$$

where $p \in \mathsf{At}, g \in \Gamma_0$.

Game Logic: Semantics I

A **neighborhood game model** is a tuple
$\mathcal{M} = \langle W, \{E_g \mid g \in \Gamma_0\}, V\rangle$ where

W is a nonempty set of states

For each $g \in \Gamma_0$, $E_g : W \to 2^{2^W}$ is an **effectivity function** such that if $X \subseteq X'$ and $X \in E_g(w)$ then $X' \in E_g(w)$.

$X \in E_g(w)$ means in state s, Angel has a strategy to force the game to end in *some* state in X (we may write wE_gX)

$V : \mathsf{At} \to 2^W$ is a valuation function.

Game Logic: Semantics I

A **neighborhood game model** is a tuple
$\mathcal{M} = \langle W, \{E_g \mid g \in \Gamma_0\}, V\rangle$ where

Propositional letters and boolean connectives are as usual.

$\mathcal{M}, w \models \langle\gamma\rangle\phi$ iff $(\phi)^{\mathcal{M}} \in E_\gamma(w)$

Game Logic: Semantics I

A **neighborhood game model** is a tuple
$\mathcal{M} = \langle W, \{E_g \mid g \in \Gamma_0\}, V\rangle$ where

Propositional letters and boolean connectives are as usual.

$\mathcal{M}, w \models \langle\gamma\rangle\phi$ iff $(\phi)^{\mathcal{M}} \in E_\gamma(w)$

Suppose $E_\gamma(Y) = \{s \mid Y \in E_g(s)\}$

- $E_{\gamma_1;\gamma_2}(Y) \ := \ E_{\gamma_1}(E_{\gamma_2}(Y))$
- $E_{\gamma_1\cup\gamma_2}(Y) \ := \ E_{\gamma_1}(Y) \cup E_{\gamma_2}(Y)$
- $E_{\phi?}(Y) \ := \ (\phi)^{\mathcal{M}} \cap Y$
- $E_{\gamma^d}(Y) \ := \ \overline{E_\gamma(\overline{Y})}$
- $E_{\gamma^*}(Y) \ := \ \mu X.Y \cup E_\gamma(X)$

Game Logic: Axioms

1. All propositional tautologies
2. $\langle \alpha; \beta \rangle \phi \leftrightarrow \langle \alpha \rangle \langle \beta \rangle \phi$ Composition
3. $\langle \alpha \cup \beta \rangle \phi \leftrightarrow \langle \alpha \rangle \phi \vee \langle \beta \rangle \phi$ Union
4. $\langle \psi ? \rangle \phi \leftrightarrow (\psi \wedge \phi)$ Test
5. $\langle \alpha^d \rangle \phi \leftrightarrow \neg \langle \alpha \rangle \neg \phi$ Dual
6. $(\phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi) \rightarrow \langle \alpha^* \rangle \phi$ Mix

and the rules,

$$\frac{\phi \qquad \phi \rightarrow \psi}{\psi} \qquad \frac{\phi \rightarrow \psi}{\langle \alpha \rangle \phi \rightarrow \langle \alpha \rangle \psi} \qquad \frac{(\phi \vee \langle \alpha \rangle \psi) \rightarrow \psi}{\langle \alpha^* \rangle \phi \rightarrow \psi}$$

Some Results

- Game Logic is more expressive than **PDL**

Some Results

- Game Logic is more expressive than **PDL**

$$\langle (g^d)^* \rangle \bot$$

Some Results

- Game Logic is more expressive than **PDL**

$$\langle (g^d)^* \rangle \bot$$

- The induction axiom is not valid in GL.

R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

Some Results

- Game Logic is more expressive than **PDL**

$$\langle (g^d)^* \rangle \bot$$

- The induction axiom is not valid in GL.

R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

- All GL games are determined. This is not a trivial result since neither Zermelo's Theorem nor the Gale-Stewart Theorem can be applied.

M. Pauly. *Game Logic for Game Theorists.* Available at `http://www.stanford.edu/ pianoman/`.

Some Results

Theorem [1] Dual-free game logic is sound and complete with respect to the class of all game models.

Theorem [2] Iteration-free game logic is sound and complete with respect to the class of all game models.

Open Question Is (full) game logic complete with respect to the class of all game models?

[1] R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

[2] M. Pauly. *Logic for Social Software.* Ph.D. Thesis, University of Amsterdam (2001)..

Some Results

Theorem [2] Given a game logic formula ϕ and a finite game model $\mathcal{M}$, model checking can be done in time $O(|\mathcal{M}|^{ad(\phi)+1} \times |\phi|)$

Theorem [1,2] The satisfiability problem for game logic is in EXPTIME.

Theorem [1] Game logic can be translated into the modal μ-calculus

[1] R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

[2] M. Pauly. *Logic for Social Software.* Ph.D. Thesis, University of Amsterdam (2001)..

Some Results

Say two games γ_1 and γ_2 are equivalent provided $E_{\gamma_1} = E_{\gamma_2}$ iff $\langle\gamma_1\rangle p \leftrightarrow \langle\gamma_2\rangle p$ is valid for a p which occurs neither in γ_1 nor in γ_2.

Theorem [1,2]] Sound and complete axiomatizations of (iteration free) game logic

Theorem [3] No finite level of the modal μ-calculus hierarchy captures the expressive power of game logic.

[1] Y. Venema. *Representing Game Algebras.* Studia Logica **75** (2003)..

[2] V. Goranko. *The Basic Algebra of Game Equivalences.* Studia Logica **75** (2003)..

[3] D. Berwanger. *Game Logic is Strong Enough for Parity Games.* Studia Logica **75** (2003)..

More Information

Editors: M. Pauly and R. Parikh. *Special Issue on Game Logic*. Studia Logica **75**, 2003.

M. Pauly and R. Parikh. *Game Logic — An Overview*. Studia Logica **75**, 2003.

R. Parikh. *The Logic of Games and its Applications.*. Annals of Discrete Mathematics. (1985) .

M. Pauly. *Game Logic for Game Theorists*. Available at `http://www.stanford.edu/ pianoman/`.

Example: Banach-Knaster Cake Cutting Algorithm

The Algorithm:

Example: Banach-Knaster Cake Cutting Algorithm

The Algorithm:

- The first person cuts out a piece which he claims is his fair share.

Example: Banach-Knaster Cake Cutting Algorithm

The Algorithm:

- The first person cuts out a piece which he claims is his fair share.

- The piece goes around being inspected by each agent.

Example: Banach-Knaster Cake Cutting Algorithm

The Algorithm:

- The first person cuts out a piece which he claims is his fair share.
- The piece goes around being inspected by each agent.
- Each agent, in turn, can either reduce the piece, putting some back to the main part, or just pass it.

Example: Banach-Knaster Cake Cutting Algorithm

The Algorithm:

- The first person cuts out a piece which he claims is his fair share.
- The piece goes around being inspected by each agent.
- Each agent, in turn, can either reduce the piece, putting some back to the main part, or just pass it.
- After the piece has been inspected by p_n, the last person who reduced the piece, takes it. If there is no such person, then the piece is taken by p_1.

Example: Banach-Knaster Cake Cutting Algorithm

The Algorithm:

- The first person cuts out a piece which he claims is his fair share.
- The piece goes around being inspected by each agent.
- Each agent, in turn, can either reduce the piece, putting some back to the main part, or just pass it.
- After the piece has been inspected by p_n, the last person who reduced the piece, takes it. If there is no such person, then the piece is taken by p_1.
- The algorithm continues with $n - 1$ participants.

Example: Banach-Knaster Cake Cutting Algorithm

Correctness: The algorithm is "correct" iff each player has a winning strategy for achieving a fair outcome ($1/n$ of the pie according to p_i's own valuation).

Towards a Formal Proof: A state will consist of the values of $n+2$ variables.

- The variable m has as its value the main part of the cake.
- The variable x is the piece under consideration.
- For $i = 1, \ldots, n$, the variable x_i has as its value the piece, if any, assigned to the person p_i.

Variables $m, x, x_1, \ldots, x_n$ range over subsets of the cake.

Example: Banach-Knaster Cake Cutting Algorithm

The algorithm uses three basic actions.

- c cuts a piece from m and assigns it to x. c works only if x is 0.
- r (reduce) transfers some (non-zero) portion from x back to m.
- a_i (assign) assigns the piece x to person p_i. Thus a_i is simply, $(x_i, x) := (x, 0)$.

Example: Banach-Knaster Cake Cutting Algorithm

The algorithm uses three basic actions.

- c cuts a piece from m and assigns it to x. c works only if x is 0.
- r (reduce) transfers some (non-zero) portion from x back to m.
- a_i (assign) assigns the piece x to person p_i. Thus a_i is simply, $(x_i, x) := (x, 0)$.

And predicates:

- $F(u, k)$: the piece u is big enough for k people.
- $F(u)$ abbreviates $F(u, 1)$ and F_i abbreviates $F(x_i)$.

Example: Banach-Knaster Cake Cutting Algorithm

Assume the following propositions

Example: Banach-Knaster Cake Cutting Algorithm

Assume the following propositions

1. $F(m, k) \rightarrow \langle c \rangle (F(m, k-1) \wedge F(x))$

Example: Banach-Knaster Cake Cutting Algorithm

Assume the following propositions

1. $F(m,k) \rightarrow \langle c \rangle (F(m,k-1) \wedge F(x))$

1′. $F(m,k) \rightarrow (c,i)(F(m,k-1) \wedge F(x))$

Example: **Banach-Knaster Cake Cutting Algorithm**

Assume the following propositions

1. $F(m,k) \rightarrow \langle c \rangle (F(m,k-1) \wedge F(x))$

1′. $F(m,k) \rightarrow (c,i)(F(m,k-1) \wedge F(x))$

2. $F(m,k) \rightarrow [r*]F(m,k)$

Example: **Banach-Knaster Cake Cutting Algorithm**

Assume the following propositions

1. $F(m,k) \rightarrow \langle c \rangle (F(m,k-1) \wedge F(x))$

1′. $F(m,k) \rightarrow (c,i)(F(m,k-1) \wedge F(x))$

2. $F(m,k) \rightarrow [r*]F(m,k)$

3. $F(m,k) \rightarrow [c][r*](F(m,k-1) \vee \langle r \rangle (F(m,k-1) \wedge F(x)))$

Example: **Banach-Knaster Cake Cutting Algorithm**

Assume the following propositions

1. $F(m, k) \rightarrow \langle c \rangle (F(m, k-1) \wedge F(x))$

1′. $F(m, k) \rightarrow (c, i)(F(m, k-1) \wedge F(x))$

2. $F(m, k) \rightarrow [r*]F(m, k)$

3. $F(m, k) \rightarrow [c][r*](F(m, k-1) \vee \langle r \rangle (F(m, k-1) \wedge F(x)))$

4. $F(x) \rightarrow [a_i]F_i$

Example: **Banach-Knaster Cake Cutting Algorithm**

Assume the following propositions

1. $F(m, k) \rightarrow \langle c \rangle (F(m, k-1) \wedge F(x))$

1′. $F(m, k) \rightarrow (c, i)(F(m, k-1) \wedge F(x))$

2. $F(m, k) \rightarrow [r*]F(m, k)$

3. $F(m, k) \rightarrow [c][r*](F(m, k-1) \vee \langle r \rangle (F(m, k-1) \wedge F(x)))$

4. $F(x) \rightarrow [a_i]F_i$

There are tacit assumptions of relevance, e.g. that r and c can only affect statements in which m or x occurs.

We assume moreover that $F(m, n)$ is true at the beginning.

Example: **Banach-Knaster Cake Cutting Algorithm**

The (in)formal proof:

1. We show now that each person p_i has a winning strategy so that if, after the kth cycle, (s)he is still in the game then $F(m, n-k)$ and if (s)he is assigned a piece, then F_i is true.

Example: **Banach-Knaster Cake Cutting Algorithm**

The (in)formal proof:

1. We show now that each person p_i has a winning strategy so that if, after the kth cycle, (s)he is still in the game then $F(m, n-k)$ and if (s)he is assigned a piece, then F_i is true.

2. This is true at start since $k = 0$, $F(m, n)$ holds and no one yet has a piece.

Example: Banach-Knaster Cake Cutting Algorithm

The (in)formal proof:

1. We show now that each person p_i has a winning strategy so that if, after the kth cycle, (s)he is still in the game then $F(m, n-k)$ and if (s)he is assigned a piece, then F_i is true.

2. This is true at start since $k = 0$, $F(m, n)$ holds and no one yet has a piece.

3. We now consider the inductive step from k to $k+1$. We assume by induction hypothesis that $F(m, n-k)$ holds at this stage.

Example: **Banach-Knaster Cake Cutting Algorithm**

The (in)formal proof:

1. We show now that each person p_i has a winning strategy so that if, after the kth cycle, (s)he is still in the game then $F(m, n-k)$ and if (s)he is assigned a piece, then F_i is true.

2. This is true at start since $k = 0$, $F(m, n)$ holds and no one yet has a piece.

3. We now consider the inductive step from k to $k+1$. We assume by induction hypothesis that $F(m, n-k)$ holds at this stage.

4. If $i = 1$ then since p_1 (or whoever does the cutting) does the cutting, by (1) and (1') she can achieve $F(m, n-k-1) \wedge F(x)$.

Example: **Banach-Knaster Cake Cutting Algorithm**

5. If no one does an r, she gets x and F_1 will hold since x did not change. If someone does do an r, then by (2), $F(m, n-k-1)$ will still hold and this is OK since she will then be participating at the next stage.

Example: **Banach-Knaster Cake Cutting Algorithm**

5. If no one does an r, she gets x and F_1 will hold since x did not change. If someone does do an r, then by (2), $F(m, n-k-1)$ will still hold and this is **OK** since she will then be participating at the next stage.

6. Let us now consider just one of the other people. The last person p_i to do r (if there is someone who does r) could (by (3)) achieve $F(x)$ and therefore when x is assigned to him, F_1 will hold.

Example: **Banach-Knaster Cake Cutting Algorithm**

5. If no one does an r, she gets x and F_1 will hold since x did not change. If someone does do an r, then by (2), $F(m, n-k-1)$ will still hold and this is OK since she will then be participating at the next stage.

6. Let us now consider just one of the other people. The last person p_i to do r (if there is someone who does r) could (by (3)) achieve $F(x)$ and therefore when x is assigned to him, F_1 will hold.

7. All the other cases are quite analogous, and the induction step goes through. By taking $k = n$ we see that every p_i has the ability to achieve F_i.

Next Week: Coalitional Logic, Alternating Temporal Logic