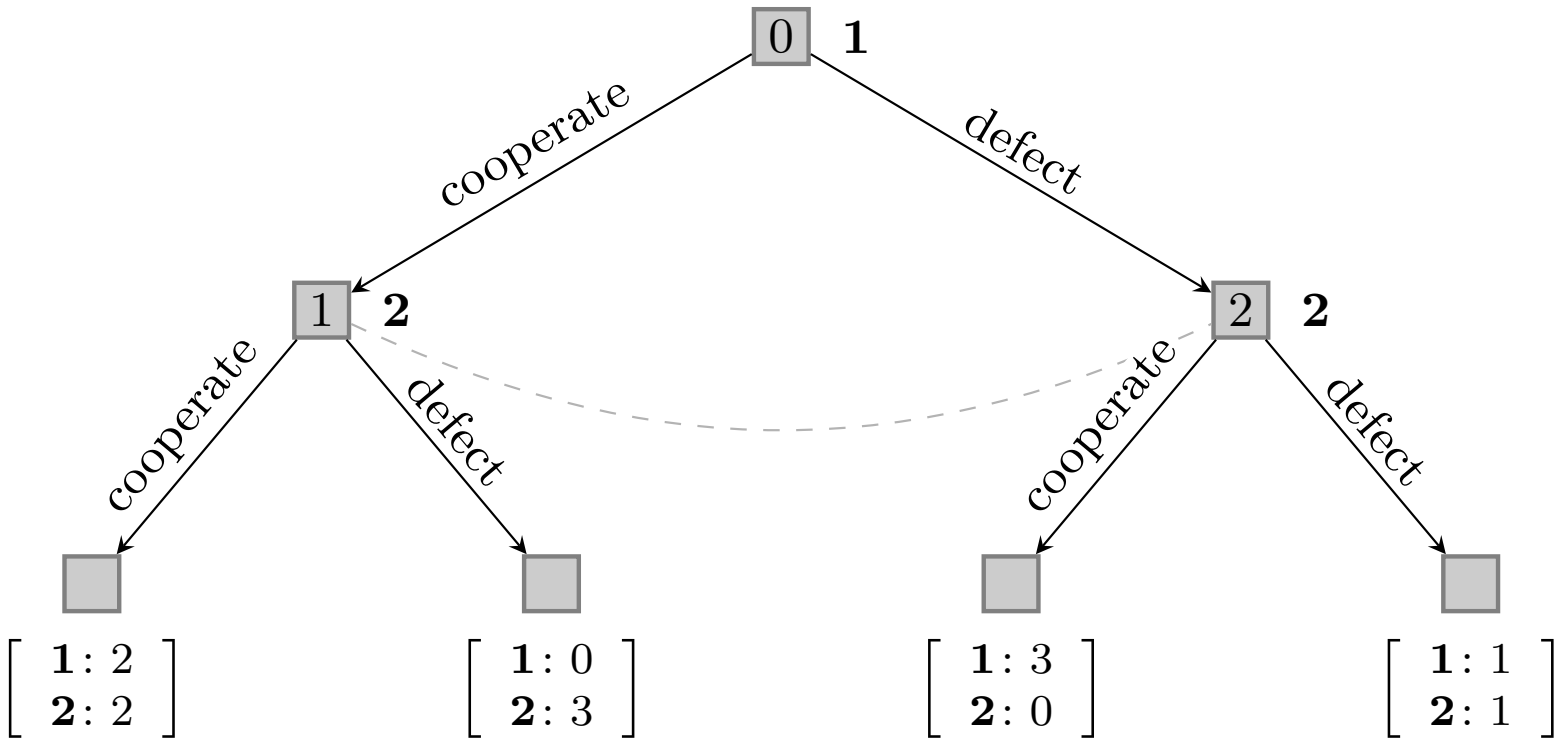
Entwurf und Implementierung von Analysealgorithmen für Spiele in erweiterter Extensivform

Vincent Göbel

16. Mai 2014

Gefangenendilemma



Strategien

Entscheidung:

- Tupel aus Informationsmenge und Aktion
- $\eta_1 = (\{0\}, \mathrm{cooperate})$
- $\eta_2 = (\{1, 2\}, \mathrm{defect})$
- Bei Variablen: $\eta_1^u = (\{0\}, (x, 2))$

Strategien

Entscheidung:

- Tupel aus Informationsmenge und Aktion
- $\eta_1 = (\{0\}, \mathrm{cooperate})$
- $\eta_2 = (\{1, 2\}, \mathrm{defect})$
- Bei Variablen: $\eta_1^u = (\{0\}, (x, 2))$

Strategie:

- Entscheidungen für jede Informationsmenge des Spielers
- $\sigma_2 = \{(\{1, 2\}, \mathrm{cooperate})\}$
- $\sigma_2^u = \{(\{1\}, \mathrm{reject}), (\{2\}, \mathrm{accept}), (\{3\}, \mathrm{accept})\}$

Strategien

Strategiekonfiguration:

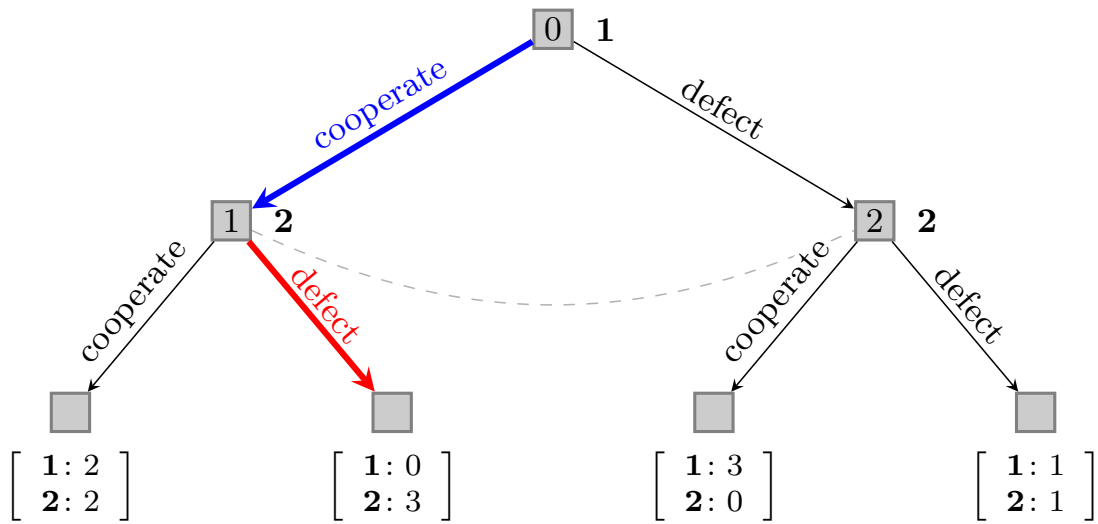
- Strategie für jeden Spieler
- $\sigma = (\{(\{0\}, \text{cooperate})\}, \{(\{1,2\}, \text{defect})\})$
- $\sigma^u = (\{(\{0\}, (x, 2))\}, \{(\{1\}, \text{reject}), (\{2\}, \text{accept}), (\{3\}, \text{accept})\})$

Strategien

Strategiekonfiguration:

- Strategie für jeden Spieler
- $\sigma = (\{(\{0\}, \text{cooperate})\}, \{(\{1,2\}, \text{defect})\})$
- $\sigma^u = (\{(\{0\}, (x,2))\}, \{(\{1\}, \text{reject}), (\{2\}, \text{accept}), (\{3\}, \text{accept})\})$

Auszahlung für Strategiekonfiguration σ:



Normalform

Auszahlungsmatrix für alle Strategiekonfigurationen

		2	
		cooperate	defect
1	cooperate	2 2	3 0
	defect	0 3	1 1

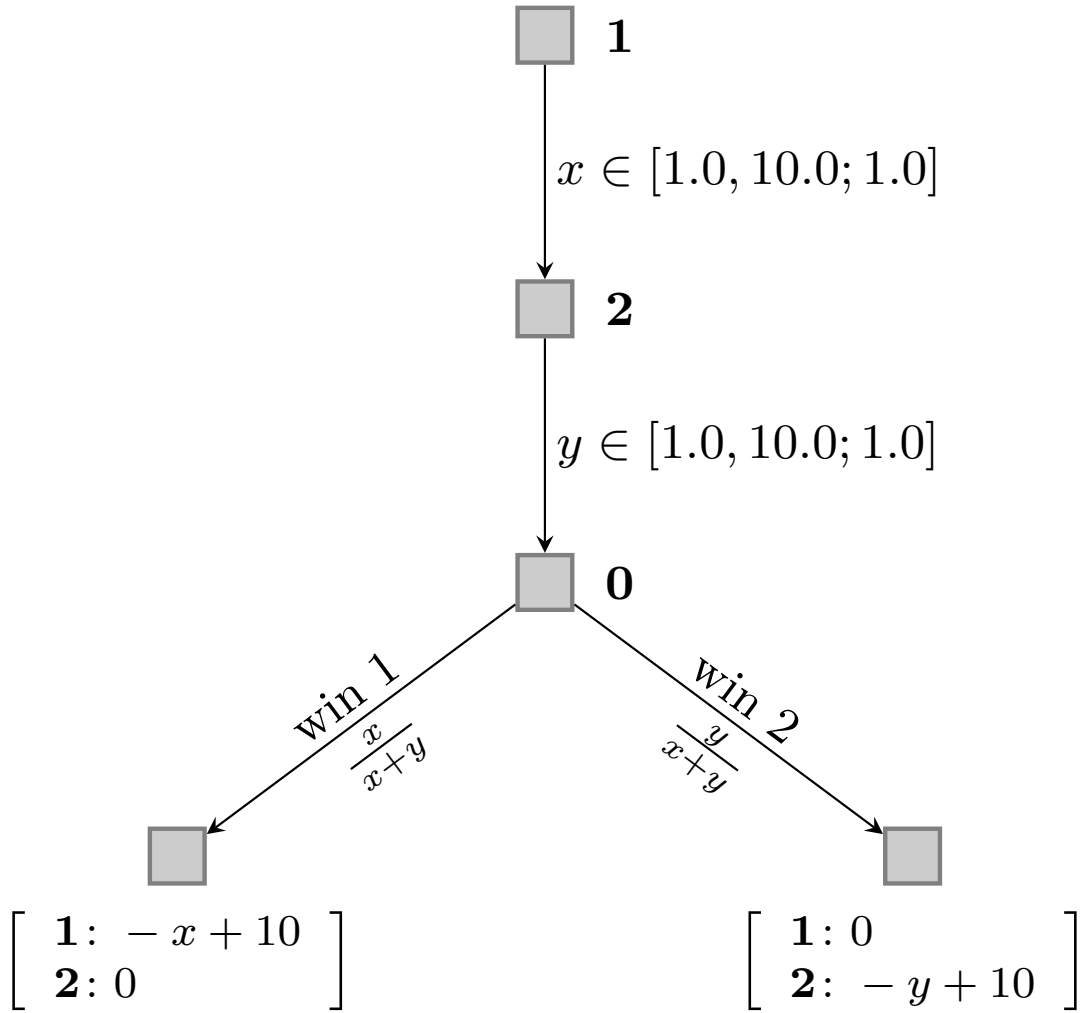
- Zeilennamen: Strategiemenge $\Sigma_1 = \{\text{cooperate}, \text{defect}\}$
- Spaltennamen: Strategiemenge $\Sigma_2 = \{\text{cooperate}, \text{defect}\}$

Erweiterte Extensivform $\rightarrow$ Normalform

Bei Gefangenendilemma simpel:

- Menge der Strategiekonfigurationen: $\Sigma = \Sigma_1 \times \Sigma_2$
- Für jedes $\sigma \in \Sigma$ Auszahlungen eintragen

Lotteriespiel



Erweiterte Extensivform $\rightarrow$ Normalform

Bei Gefangenendilemma simpel:

- Menge der Strategiekonfigurationen: $\Sigma = \Sigma_1 \times \Sigma_2$
- Für jedes $\sigma \in \Sigma$ Auszahlungen eintragen

Bei erweiterter Extensivform:

- Trennung von **Entscheidungs-** und **Variablenstrategien**
- Bestimme **Ausdrucksmatrix** für Konfiguration von E-Strategien
- Setze V-Strategien in Ausdrucksmatrix ein

Ausdrucksmatrix

- Keine E-Strategien im Ratespiel

		2
1	$\frac{x}{x+y} \cdot (10 - x)$	$\frac{y}{x+y} \cdot (10 - y)$

- Ausdrucksmatrix erspart „Simulation“ der V-Strategien
- Ein einziger Ausdruck pro Spieler
- Boolsche Ausdrücke werten zu 1 oder 0 aus

Verfügbare Analysemethoden

yTree berechnet

- Dominierte Strategien
- Nash-Gleichgewichte
- Minimax- und Maximax-Strategien

Dominierte Strategien

Strategie $\sigma_{1i} \in \Sigma_1$ ist strikt dominiert, g.d.w.

$$\exists \sigma_1^* \in \Delta(\Sigma_1) \forall \sigma_2 \in \Sigma_2 \colon H_1(\sigma_1^*, \sigma_2) > H_1(\sigma_{1i}, \sigma_2)$$

- H_1 ist die Auszahlungsfunktion von Spieler 1
- $\Delta(\Sigma_1)$: Wahrscheinlichkeitsverteilungen über Strategien in Σ_1
- $\Delta(\Sigma_1)$ ist Menge der *gemischten Strategien* von Spieler 1
- Gemischte Strategie $\sigma_1 = (p_{11}, \ldots, p_{1m_1}) \quad p_{1i} \in [0; 1]$

Neu formuliert: Strategie σ_{1i} ist strikt dominiert, g.d.w.

$$\exists \sigma_1^* \in \Delta(\Sigma_1) \forall \sigma_2 \in \Sigma_2 \colon \sum_{j=1}^{m_1} p_{1j}^* H_1(\sigma_{1j}, \sigma_2) > H_1(\sigma_{1i}, \sigma_2)$$

Dominierte Strategien

Formulierung als Lineares Programm:

- **x** ersetzt σ_1^*
- σ_{1i} ist strikt dominiert, g.d.w. . . .

$$\textbf{maximiere } y \tag{1}$$

$$\forall \sigma_2 \in \Sigma_2 : \sum_{j=1}^{m_1} x_j H_1(\sigma_{1j}, \sigma_2) - y \geq H_1(\sigma_{1i}, \sigma_2) \tag{2}$$

$$\sum_{j=1}^{m_1} x_j = 1 \tag{3}$$

$$x_i = 0 \tag{4}$$

$$y \geq 0 \tag{5}$$

- . . .eine Lösung mit $y > 0$ hat.
- y dient als Ersatz für strikte Ungleichungen

Analyzer

```
class StrictDominationAnalyzer(Analyzer):
  def __init__(self):
    Analyzer.__init__(self)
    self.required_types["game"] = Game
    self.required_types["matrix_analyzer"] = \
        AbstractPayoffMatrixAnalyzer

    self.return_types["dominated_strategies"] = dict
    self.return_types["dominated_edges"] = EdgeList


  def analyze(self):
    ...
```

- Erwartet Eingabewerte
- Berechnet Ergebnisse

DEMO

Mögliche Erweiterungen

- Super-Analyzer
- Konstanten, Loops
- Strategien im Editor
- weitere Algorithmen

Entwurf und Implementierung von Analysealgorithmen für Spiele in erweiterter Extensivform

Vincent Göbel

16. Mai 2014