Subgame Detection

Previous paper published developed for single player games

First introduce simple rules for **2-TicTacToe, later also** talk about **instances to generalize for other multiplayer games**

Search Space of Double-TicTacToe

Seen Double-TicTacToe, 2 games at the same time. Imagine **n-TicTacToe**

Search Space explodes**: 18 factorial, 18!**

Examples: Make moves on single field

Subgames

We have some **terms to introduce**

Explain

F' is a Subset of F, … all **not empty**!

If two subgames do **not intersect** in Fluents and Actions (no overlap) they are **independent**

Fluent is action independent all actions have the **same effect** on this fluent

Fluents **change** does **not depend** on **what** the **moves** are and **who** is playing.
For example: **control**() fluent

Some games have a **counter**() which decreases every move → independent

Incomplete dependency graph of 2-TicTacToe

Only for **visualization,** very **incomplete and superficial**!

Talked about **Independent subgames** but both interact with **control**() fluent

control()-fluent is **action independent** → **ignore** it!

Klick!

2 unconnected components: These are the **subgames**

Imagine Search Space: Smaller than 2-TicTacToe because search solutions only on a single field

Instantiated Fluents

Nim: Two Players, three or more heaps of objects

Alternating turns, take any number of objects from exactly one of the heaps

Goal: Be the last to take an object OR: Force opponent to take last object

GDL Rules c**onflict** with previous **definition,** we want **heaps** to be **subgames**

Important: **heap(b, 2)** every heap is an **Instance** of fluent **heap()**

Paper: Formal definition of **instantiated arguments of fluents**

Idea: **Instantiated** if its value does not **change** from one **state** to the **next** and if **instances** do not **interact**

Klick!

5

Instantiated Fluents 2

heap(X, _) is **instantiated** because the **rules** only refer to the **same heap** directly (line 13) or indirectly (check if move is **legal()** if player **does()** a move)

Not depicted: **Reduce(X)** operation influences only **heap(X, _)** and **control (we ignore control)**

As said: Because **control(W)** appears in **legal()** rules and would connect all actions and we would have no **unconnected subgames**

If we do this, we get the graph below

6

Decomposition into Subgames

To summarize it:

Some games contain **subgames** which can be clearly **separated** from each other

The **search tree** of **double**-TicTacToe from the beginning can be **reduced** because we find a winning strategy for one field (and reuse it for the other field as well)

Imagine an **n-TicTacToe**

Heuristic wrap up Find subgames by:

1. making a move on some fluent → nothing happens to some other fluent

2. Two fluents do not interact but if they do, they are in the same **subgame**

Now you may ask: Fine, we can win a **single TicTacToe**, but how do we win **both** at the **same time**? → Richard will talk about this now

Impartial Games

Short **remark** on **impartial games**: **Nim**, Quarto

Games, where the **effects** of each **move** are **independent** on **who** is making the move

The only **asymmetric** aspect is the **starting** player, afterwards both are equal

Not: **Chess**, Checkers, Go because each player can only move pieces of his **own color**

Poker, dice not because they rely on **chance** and are not deterministic

Hard to check every **state** of the game regarding impartiality. So: **Syntactic analysis** on the rules. Check legal and next rules for all players by **substitution** and checking if the **corresponding rule** exists, too

Iff independent **subgames** are **impartial** → **whole game** is impartial

Subgame Search: Only **compute** for **one player** (equivalent), saves effort

Global Game Search: No details, every impartial game can be seen as some instance of a **Nim** heap. So for an impartial game with **subgames** you can just use winning-strategies for regular **Nim**