WoodStock et le General Game Playing

Frédéric Koriche, Sylvain Lagrue, **Éric Piette**, Sébastien Tabary
Université Lille-Nord de France, CRIL, Lens

6 décembre 2016

General Game Playing (GGP)

- ➜ Développer un « programme-joueur » générique
- ➜ Compétition internationale annuelle et compétition en ligne continue
- ➜ Diverses approches proposées : Construction automatique de fonctions d'évaluations, Programmation logique, ASP (Answer Set Programming), Méthodes à base de Monte Carlo (UCT)
- ➜ GDL : Un langage logique pour la représentation des jeux à informations complètes

Un nouveau défi : GDLII

Les jeux à informations incomplètes

GDL : Game Description Language

GDL : Langage générique pour les jeux à informations complètes :

- ➜ Dérivé de la **programmation logique** avec négation et égalité
- ➜ Joueurs et Objets = Termes | Fluents et Actions = Predicats
- ➜ Ensemble de termes et de prédicats = Atome

Quelques mots-clés

Mots-clés GDL	**Description**
role(J)	J est un joueur
init(F)	le fluent F modélise l'état initial
true(F)	F modélise l'état courant
legal(J, A)	J peut réaliser l'action A
does(J, A)	l'action de J est A
next(F)	F modélise l'état suivant.
terminal	l'état courant est terminal
goal(J, N)	J a N points à l'état courant

Game Description Language with Incomplete Information

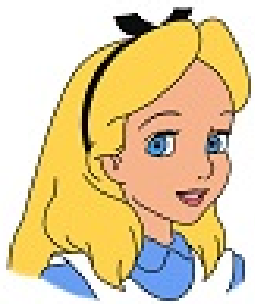
GDLII : extension de GDL pour les jeux à informations incomplètes :

- ➜ Ajout d'un joueur environnement modélisant le hasard
- ➜ Ajout de règles modélisant la perception d'un joueur à l'état courant

Quelques mots-clés supplémentaires

Mots-clés GDL	**Description**
`random`	représente l'environnement
`sees(J,P)`	J perçoit P à l'état courant

- ➜ Propriété 1 : Tout jeux GDL est modélisable en GDLII
- ➜ Propriété 2 : GDLII est universel



+

$Pièce_{alice}$ $Pièce_{vue}$, $Pièce_{cachée}$

2 pièces identiques

100 points

1 pièce identique

50 points

0 pièce identique

0 points

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1, _)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1, _)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1, _)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1,_)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1, _)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1, _)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1,_)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1,_)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1,_)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Jeu des Pièces cachées en GDLII

role(alice). **role**(random).

cote(pile). cote(face).

init($piece_a$(inconnue)).
init($piece_r$(inconnue,inconnue)).
init(control(random)).

legal(J,noop) ← **not**(**true**(control(J))).
legal(random,choisit(C1,C2)) ← **true**(control(random)), cote(C1), cote(C2).
legal(alice,retourne(C)) ← **true**(control(alice)), cote(C).

next(control(alice)) ← **true**(control(random)).
next($piece_r$(C1,C2)) ← **does**(random,choisit(C1,C2)).
next($piece_r$(C1,C2)) ← **true**($piece_r$(C1,C2)),**not**(**true**(control(random))).
next($piece_a$(C)) ← **does**(alice,retourne(C)).
next($piece_a$(C)) ← **true**($piece_a$(C)), **not**(**true**(control(alice))).

sees(alice,$piece_r$(C1,_)) ← **does**(random,choisit(C1,C2)).

terminal ← not(**true**($piece_a$(inconnue))), not(**true**($piece_r$(inconnue,inconnue))).

goal(alice,100) ← **true**($piece_a$(C)), **true**($piece_r$(C,C)).
goal(alice,50) ← **or**(**true**($piece_a$(C1)) **true**($piece_a$(C2))), **true**($piece_r$(C1,C2), **distinct**(C1,C2).
goal(alice,0) ← **true**($piece_a$(C1)), **true**($piece_r$(C2,C2)), distinct(C1,C2).

Travaux principaux de thèse

- ➜ Modélisation SCSP d'un jeu GDLII
- ➜ Identification d'un fragment de SCSP
- ➜ Algorithme MAC-UCB
- ➜ Détection efficace des symétries de jeux dans GGP
- ➜ Notre programme-joueur générique : WoodStock
  (With Our Own Developper STOchastic Constraint toolKit)

WoodStock : un programme-joueur dirigé par les contraintes

- Phase de traduction : GDL vers SCSP + prétraitement (SAC)
- Phase de résolution : MAC + SFC
- Phase de simulation : UCB
- Détection des symétries : Nauty

Mise en ligne de WoodStock sur le serveur Tiltyard.

WoodStock en compétition

Compétition continue : 1^{er} depuis mars 2016

Compétition annuelle 2016 (IGGPC) : 1^{er} - Champion GGP 2016

Perspectives

1. Étude poussée des techniques de résolution des SCSP générés

Perspectives

1. Étude poussée des techniques de résolution des SCSP générés
2. Paralléliser la phase de résolution et la phase de simulation

Perspectives

1. Étude poussée des techniques de résolution des SCSP générés
2. Paralléliser la phase de résolution et la phase de simulation
3. Détection de propriétés via SCSP et génération d'heuristiques

Perspectives

1. Étude poussée des techniques de résolution des SCSP générés
2. Paralléliser la phase de résolution et la phase de simulation
3. Détection de propriétés via SCSP et génération d'heuristiques
4. Détection automatique de jeux GDL valides (détection de cycles)

Perspectives

1. Étude poussée des techniques de résolution des SCSP générés
2. Paralléliser la phase de résolution et la phase de simulation
3. Détection de propriétés via SCSP et génération d'heuristiques
4. Détection automatique de jeux GDL valides (détection de cycles)
5. Adaptation des techniques de simulations de MAC-UCB

Perspectives

1. Étude poussée des techniques de résolution des SCSP générés
2. Paralléliser la phase de résolution et la phase de simulation
3. Détection de propriétés via SCSP et génération d'heuristiques
4. Détection automatique de jeux GDL valides (détection de cycles)
5. Adaptation des techniques de simulations de MAC-UCB
6. S'attaquer à d'autres modèles généraux (VGDL et GDL-III)