

Initiation à la recherche

15/05/2020

Etudiants

Angela IPSEIZ

Maxime NICOLAS

Matthieu OLEJNICZAK

Encadrant

Stephan MERZ



0101100
0110111
0110010
0110101
0110001
0110100
0110111
0110010
0110101
1110001011
1110001011
00001011
111111

Loria
1/20

The background of the slide is a dark gray color with a repeating pattern of thin, gold-colored hexagonal outlines, resembling a honeycomb or cellular structure. The pattern is most dense at the top and bottom edges, with some hexagons missing or faded in the center to create a subtle gradient effect.

“Preuve mécanisée de l’algorithme de Tarjan”

Introduction

Algorithme de Tarjan

Preuve de l'algorithme de Tarjan

Problèmes et solutions apportées

Conclusion

Sommaire

Introduction

- Correction
 - Fonctionne correctement et finit
 - Être sûr de ce que l'on fait
 - Sécurité et économie

Algorithme de Tarjan

Description de l'algorithme


- Identifier les composantes fortement connexes
- Parcours en profondeur
- Complexité linéaire

Fonctionnement de l'algorithme

- Distinction des sommets
- Utilisation d'une pile
- Repérage des composantes

Exemple de fonctionnement

(Commentaire : Nous préparons une animation sur
un exemple concret.)

The background of the slide is a dark gray color with a repeating pattern of thin, gold-colored hexagonal outlines, resembling a honeycomb or cellular structure.

Preuve de l'algorithme de Tarjan

TLA+

- (Notre premier modèle : SCC)
- Utilisation de TLA+ proof system

Preuve de l'algorithme

- Invariant de boucle
- Méthodologie

Invariant de typage

```
TypeOK ==
/\ index \in Nat
/\ t_stack \in Seq(Nodes)
/\ num \in [Nodes -> Nat \cup {-1}]
/\ lowlink \in [Nodes -> Nat \cup {-1}]
/\ onStack \in [Nodes -> BOOLEAN]
/\ sccs \in SUBSET SUBSET Nodes
/\ toVisit \in SUBSET Nodes
/\ pc \in {"main", "Done", "start_visit", "explore_succ", "visit_recurse", "continue_visit", "check_root"}
/\ stack \in Seq(StackEntry)
/\ \A i \in 1 .. Len(stack) : stack[i].pc = "continue_visit" =>
  /\ i < Len(stack)
  /\ stack[i].v \in Nodes /\ num[stack[i].v] \in Nat
  /\ stack[i].w \in Nodes
/\ pc \in {"start_visit", "explore_succ", "visit_recurse", "continue_visit", "check_root"}
=> /\ stack # << >>
  /\ Head(stack).pc = "continue_visit" => Head(stack).v \in Nodes
  /\ Head(stack).pc = "continue_visit" => Head(stack).w \in Nodes
/\ succs \in SUBSET Nodes
/\ v \in Nodes \cup {defaultInitValue}
/\ pc \in {"start_visit", "explore_succ", "visit_recurse", "continue_visit", "check_root"} => v \in Nodes
/\ pc = "start_visit" => num[v] = -1
/\ pc = "visit_recurse" => num[w] = -1
/\ pc \in {"explore_succ", "visit_recurse", "continue_visit", "check_root"} => num[v] \in Nat
/\ pc \in {"visit_recurse", "continue_visit"} => w \in Nodes
/\ w \in Nodes \cup {defaultInitValue}
```

Invariant des sommets sur la pile

```
NumStackInv ==  
  /\ index <= Cardinality(Nodes)  
  /\ \A n \in Nodes : num[n] < index  
  /\ \A n \in Nodes : onStack[n] <=> \E i \in 1 .. Len(t_stack) : t_stack[i] = n  
  /\ \A n \in Nodes : num[n] \in Nat <=> (onStack[n] \vee n \in UNION sccs)  
  /\ \A i \in 1 .. Len(t_stack) : \A j \in 1 .. Len(t_stack) :  
    /\ i <= j <=> num[t_stack[j]] <= num[t_stack[i]]  
    /\ t_stack[i] = t_stack[j] => i = j  
  /\ index + Cardinality({n \in Nodes : num[n] = -1}) = Cardinality(Nodes)
```


Invariant de couleur

```
White == {n \in Nodes : num[n] \notin Nat}
Gray == {n \in Nodes : n = v \/\ \E i \in 1 .. Len(stack) : n = stack[i].v}
Black == Nodes \ (White \cup Gray)

ColorInv ==
  /\ (* Nodes that are no longer in toVisit aren't white *)
    White \subseteq toVisit \cup (IF pc = "start_visit" THEN {v} ELSE {})
  /\ (* Successors of a visited node that are no longer in succs aren't white *)
    pc \in {"explore_succ", "visit_recurse", "continue_visit", "check_root"}
    => White \cap Succs[v] \subseteq succs \cup (IF pc = "visit_recurse" THEN {w} ELSE {})
  /\ (* analogous condition for stack entries *)
    \A i \in 1 .. Len(stack) : stack[i].pc = "continue_visit" =>
      /\ White \cap Succs[stack[i].v]
        \subseteq stack[i].succs \cup (IF pc = "start_visit" /\ v = stack[i].w THEN {v} ELSE {})
  /\ (* black nodes do not have white successors *)
    \A n \in Black : Succs[n] \cap White = {}
```

Un prouveur capricieux

Quand le prouveur accepte :

```
<3>10. (∀A i \in 1 .. Len(stack) : stack[i].pc = "continue_visit" =>  
  /\ i < Len(stack)  
  /\ stack[i].v \in Nodes /\ num[stack[i].v] \in Nat  
  /\ stack[i].w \in Nodes)'
```

Quand le prouveur refuse :

```
<3>4. (∀A n \in Black : Succs[n] \cap White = {})"  
BY <2>5 DEF check_root
```




Problèmes et solutions apportées

Transcription de l'algorithme en TLA+

- Récursivité et TLA+
- Utilisation d'un transcripteur (PlusCal)

Preuve de l'algorithme

(Nous sommes en train de faire les preuves)

Conclusion

- Un invariant se travaille
- Preuve partielle
- Le monde de la preuve est en évolution

(Commentaire : On ajoutera encore des illustrations)

Merci de votre
attention

Fin