Lodě
Důkaz funkce whereNotToShoot

Karel Čemus

21. ledna 2012

Úvod

Pro důkaz byla přednášejícím navržena funkce vracející seznam polí, kam nemá smysl střílet. V kódu se tato funkce nazývá whereNotToShoot a je označena viditelným komentářem. Tato funkce vrací pouze ta pole, kam nemá smysl střílet, ale nezaručuje, že jsou to všechna pole, která lze vyloučit. Zaměřuje se pouze na vlastní zásahy do vody, zásahy do lodí, rozpoznaná okolí lodí a pole, kam nemá smysl střílet pro nedostatek prostoru. (více viz ilustrace v důkazu).

Chceme dokázat, že funkce definovaná pravidlem

```
op whereNotToShoot(_,_) : Boats Positions -> Positions .
```

vrací pouze taková pole, kam opravdu nemá smysl střílet. Bude proveden důkaz pro použité funkce. Každá funkce bude dekomponována na jednotlivé možnosti a dokázána vůči předpokladům a pravidlům hry, které definují axiomy. Některé důkazy nebudou prováděny do konce, neboť se jedná o zřejmý výsledek funkce.

Předpoklady

Předpokládejme axiomatickou platnost pravidel hry a správnost definování objektové reprezentace hry a jejích součástí. Jedná se o objekty Coords (souřadnice), CoordsTuple (sekvence souřadnic), Boat (loď) a Boats (sekvence lodí). Dále předpokládejme, že ve všech důkazech pracujeme se sekvencemi prvků.

1 Funkce sort()

Tvrzení: Metoda řadí přidělenou sekvenci souřadnic.
Přepisovací pravidlo:

```
op _->sort() : CoordsTuple -> CoordsTuple .
eq nul ->sort() = nul .
eq SEQ ->sort() = SEQ ->min() SEQ ->remove( SEQ ->min() )->sort() .
```

Důkaz: Předpokládejme, že funkce min() vrací nejmenší prvek sekvence a funkce remove() odebere prvek ze sekvence. Pak lze přepisovací relaci přepsat na:

$$\exists y \in Coords,\ \forall x \in SEQ:\ y \leq x\ \Rightarrow \mathrm{sort}(SEQ) = y \cup \mathrm{sort}(SEQ \setminus y)$$

Z tohoto zápisu je zřejmé, že se jedná o select sort, který vždy vybere nejmenší prvek sekvence a na zbytek zavolá rekurzivně sort(). Důkaz není nutný neboť se odvoláváme na známý algoritmus.

2 Funkce unique()

Tvrzení: Metoda odstraní duplicitní výskyty prvků.
Přepisovací pravidlo:

```
op _->unique() : CoordsTuple -> CoordsTuple .
eq ( nul )->unique() = nul .
eq ( C REST )->unique() =
      if ( REST )->contains( C ) then nul else C fi
      ( REST )->unique() .
```

Důkaz: Musíme dokázat 2 vlastnosti funkce. Za prvé, že je-li prvek obsažen v sekvenci X, pak je obsažen i v $unique(X)$. A za druhé, že funkce odstraňuje duplicity. Oba důkazy budou provedeny sporem. Přepisovací pravidla lze převést do následujícího tvaru:

- $\text{unique}(X_0 \ \ldots \ X_n) = \text{unique}(X_1 \ \ldots \ X_n)$ if $X_0 \in (X_1 \ \ldots \ X_n)$
- $\text{unique}(X_0 \ \ldots \ X_n) = X_0 \cup \text{unique}(X_1 \ \ldots \ X_n)$ if $X_0 \notin (X_1 \ \ldots \ X_n)$

1. Chceme dokázat sporem, že fce unique vrací všechny prvky, které obsahuje původní sekvence $X_0 \ \ldots \ X_n$. Předpokládejme tedy:

   $$\exists Y \in X_0 \ \ldots \ X_n \ \wedge \ Y \notin \text{unique}(X_0 \ \ldots \ X_n)$$

   vzhledem k rekurzivní povaze funkce platí:

   $$\exists Y_i \ \ldots \ Y_n, \ Y_i = X_i, \ Y_{i+1} = X_{i+1}, \ \ldots, \ Y_n = X_n \ \wedge \ Y_i = Y$$

   potom musí nastat jedna z těchto dvou situací

   $$Y_i \in Y_{i+1} \ldots Y_n \Rightarrow \text{unique}(Y_{i+1} \ \ldots \ Y_n) \ni Y$$

   $$Y_i \notin Y_{i+1} \ldots Y_n \Rightarrow (Y_i \cup \text{unique}(Y_{i+1} \ \ldots \ Y_n)) \ni Y$$

   Spor: Y je pokaždé výsledkem unique.

2. Chceme dokázat sporem, že funkce odstraňuje duplicity. Předpokládejme existenci duplicit

$$\exists A \in X_0 \ \ldots \ X_n : \ (A = X_i \ \wedge \ A = X_j, \ i \neq j)$$

$$\wedge \ \Pi_k(\text{unique}(X_0 \ \ldots \ X_n)) = A \ \wedge \ \Pi_l(\text{unique}(X_0 \ \ldots \ X_n)) = A \ \wedge \ k \neq l$$

Pak musí existovat $X_i \ldots X_n$ takové, že $\text{unique}(X_i \ldots X_n) = X_i \cup \text{unique}(X_{i+1} \ldots X_n)$, ale pokud $X_i \in X_{i+1} \ \ldots \ X_n$, pak je to spor vůči předpisu funkce.

3 Funkce crop()

Tvrzení: Metoda odstraňuje z množiny pole, která nejsou na hrací ploše.
Přepisovací pravidlo:

```
op _->crop() : CoordsTuple -> CoordsTuple .
eq ( nul )->crop() = nul .
eq ( C REST )->crop() =if C->defined() then C else nul fi REST->crop()
```

Důkaz: Předpokládejme, že funkce defined() vrací TRUE, pokud je pole C na hrací ploše a FALSE, pokud je mimo hranice herní plochy. Pak můžeme přepisovací pravidlo převést na:

$$\text{crop}(X) = \{x \in X : \ \text{defined}(x) = \text{TRUE}\}$$

Z toho je zřejmé, že funkce vrací všechny souřadnice, které jsou na hrací ploše.

4 Funkce containsNeighbours()

Tvrzení: Metoda rozhoduje, zda-li sekvence obsahuje dvě sousední pole - horizontálně nebo vertikálně.
Přepisovací pravidlo:

```
op _->containsNeighbours() : CoordsTuple -> CoordsTuple .
eq ( nul )->containsNeighbours() = nul .
eq ( C REST ) ->containsNeighbours() = if REST ->contains(C + [0,1])
   or REST ->contains(C + [1,0]) or REST ->contains(C + [0,-1])
   or REST ->contains(C + [-1,0]) then true
else REST ->containsNeighbours() fi .
```

Důkaz:

$$\text{containsNeighbours}(X_1 \dots X_n) = (X_2 \dots X_n) \ni (X + [1,0]) \vee (X_2 \dots X_n) \ni (X + [-1,0]) \vee$$

$$(X_2 \dots X_n) \ni (X + [0,1]) \vee (X_2 \dots X_n) \ni (X + [0,-1]) \vee \text{containsNeighbours}(X_2 \dots X_n)$$

Z tohoto zápisu je zřejmé, že funkce rekurzivně pro všechna pole testuje, zda-li zbývající část sekvence obsahuje nějaké sousední pole.

5 Funkce isRotationDefined()

Tvrzení: Metoda vrací TRUE právě tehdy, když je definovaná orientace lodi. To je pouze v případě, že byla zasažena na dvou sousedních místech. Nesousední zásahy nezaručují, že se jedná o stejnou loď.

Přepisovací pravidlo:

```
op _->isRotationDefined() : CoordsTuple -> CoordsTuple .
eq [ TYPE, ROT, C, POSITIONS, {} ] ->isRotationDefined() = false .
eq [ TYPE, ROT, C, POSITIONS, { HITS } ] ->isRotationDefined() =
   HITS ->containsNeighbours() .
```

Důkaz:

$$\text{isRotationDefined(HITS)} = \lambda\text{HITS.containsNeighbours(HITS)}$$

Z tohoto zjednodušeného přepisu na lambda funkci je jasně patrné, že funkce vrací TRUE právě tehdy, když jsou detekovány dva zásahy vedle sebe. Vzhledem k tomu, že lodě nemohou být umisťovány vedle sebe (viz pravidlo 4), tak stačí testovat zásahy v každé lodi. Není potřeba množiny všech zásahů spojovat, neboť jejich spojení nemůže přidat žádné další sousedství.

6 Funkce getFrontAndBack()

Tvrzení: Metoda vrací pole jedno před a jedno za lodí.

Přepisovací pravidlo:

```
op _->getFrontAndBack() : Boat -> Positions .
eq [ SMALL, HORIZONTAL, C, POS, HITS ] ->getFrontAndBack() =
      { ( C + [-1,0] ) ( C + [2,0] ) } .
```

```
eq [ SMALL, VERTICAL, C, POS, HITS ] ->getFrontAndBack() =
     { ( C + [0,-1] ) ( C + [0,2] ) } .
eq [ LARGE, HORIZONTAL, C, POS, HITS ] ->getFrontAndBack() =
     { ( C + [-1,0] ) ( C + [4,0] ) } .
eq [ LARGE, VERTICAL, C, POS, HITS ] ->getFrontAndBack() =
     { ( C + [0,-1] ) ( C + [0,4] ) } .
```

Důkaz:

Důkaz není třeba dělat. Z přepisovacích pravidel je zřejmé, že pro všechny definované typy lodí (viz pravidla) jsou vypočítána odpovídající pole.

7 Funkce sides()

Tvrzení: Metoda vrací sekvenci polí vedle lodi, kam nemá smysl střílet vzhledem k aktuálnímu stavu lodi.

Předpoklad: Metoda předpokládá, že loď obsahuje alespoň dva sousední zásahy, tzn. loď má pro soupeře definovanou orientaci.

Přepisovací pravidlo:

```
op _->getSides() : Boat -> Positions .
eq [ TYPE, HORIZONTAL, C, POS, { C' } ] ->getSides() =
     { C' ->getHorizontalSides() } .
eq [ TYPE, HORIZONTAL, C, POS, { C' REST } ] ->getSides() =
     { C' ->getHorizontalSides() }
     ( [ TYPE, HORIZONTAL, C, POS, { REST } ] ->getSides() ) .
eq [ TYPE, VERTICAL, C, POS, { C' } ] ->getSides() =
     { C' ->getVerticalSides() } .
eq [ TYPE, VERTICAL, C, POS, { C' REST } ] ->getSides() =
     { C' ->getVerticalSides() }
     ( [ TYPE, VERTICAL, C, POS, { REST } ] ->getSides() ) .
```

Důkaz:

Z pravidel hry vyplývá, že okolí lodi lze určit až ve chvíli, kdy má loď alespoň dva sousední zásahy (odkaz na pravidla 4,6,7,9). To je v předpokladech funkce. V takové situaci je definovaná orientace lodi a dle pravidla 4 je jasné, že pro každé pole lodi lze definovat 6 polí (rovnoběžně se směrem lodi), kde se nemůže žádná další loď nacházet. Přepisovací pravidla odlišují horizontálně a vertikálně orientované lodě a rekurzivně pro všechna pole

vrací seznam 6 sousedů pro každé z nich.

8 Funkce surround()

Tvrzení: Metoda vrací všechna pole okolo lodi, která lze vyloučit vzhledem k jejímu stavu.
Přepisovací pravidlo:

```
op _->getSurround() : Boat -> Positions .
eq B ->getSurround() =
      if B ->isRotationDefined() then ( B ->getSides()
         ( if B ->isSunk() then B ->getFrontAndBack() else {} fi )
      ) ->crop() ->unique() ->sort() else {} fi .
```

Důkaz: Přepisovací pravidlo lze rozdělit na 3 případy.

- je definována orientace lodi a loď je potopena: pak je zřejmé, že se jedná o celé okolí lodi, což jsou strany (getSides) a před a za (getFrontAndBack()). Tyto funkce již byly dokázány.
- je definována orientace, ale loď ještě není potopena. Pak lze vyloučit okolí kolem zásahů, neboť již známe orientaci, pouze neznáme začátek a konec lodi.
- není definována orientace - pak nelze vyloučit nic, neboť loď může pokračovat všemi směry.)

Z dekompozice vyplývá, že pravidlo popisuje všechny možné případy v souladu s pravidly hry a zohledňuje předpoklad funkce getSides().

9 Funkce blocked()

Tvrzení: Metoda vrací všechna pole, na která nemá smysl střílet vůči aktuálnímu stavu dané lodi.
Přepisovací pravidlo:

```
op _->getBlocked() : Boat -> Positions .
eq [ TYPE, ROT, C, POS, HITS ] ->getBlocked() =
      ( HITS [ TYPE, ROT, C, POS, HITS ] ->getSurround() ) ->sort().
```

Důkaz: Z přepisovacího pravidla je zřejmé, že se jedná pouze o sjednocení okolí lodě a vlastních zásahů do lodě což odpovídá tvrzení.

10 Funkce detectCrosses()

Tvrzení: Metoda vrací všechna pole, kam nemá smysl střílet, protože se tam nevejde žádná loď. Vychází se z pravidla 9, které definuje minimální délku lodi rovnou 2.

Přepisovací pravidlo:

```
op checkFree(_,_) : Coords CoordsTuple -> Bool .
eq checkFree(C,POSITIONS) =
     not( POSITIONS ->contains(C) ) and C ->defined() .
op iterate(_,_) : Coords CoordsTuple -> CoordsTuple .
eq iterate(C, POSITIONS) =
     if C ->defined() then
        POSITIONS ->checkCross(C) iterate( C + [1,0], POSITIONS )
     else
        if (C + [-10,1])->defined() then
           iterate(C + [-10,1], POSITIONS)
        else
           nul
        fi
     fi .
op _->isCross(_) : CoordsTuple Coords -> Bool .
eq ( POSITIONS )->isCross( C ) =
     if checkFree(C +[1,0],POSITIONS) or checkFree(C +[-1,0],POSITIONS)
     or checkFree(C +[0,1],POSITIONS) or checkFree(C +[0,-1],POSITIONS)
     then false else true fi .
op _->checkCross(_) : CoordsTuple Coords -> Coords .
eq ( POSITIONS )->checkCross( C ) =
     if ( POSITIONS )->isCross( C ) then C else nul fi .
op _->detectCrosses() : CoordsTuple -> CoordsTuple .
eq POSITIONS ->detectCrosses() = iterate( [1,1], POSITIONS ) .
```

Důkaz: Výše uvedená přepisovací pravidla lze zjednodušit do následujícího zápisu.

$$\text{detectCrosses}(\text{POSITIONS}) = \{\forall [x,y] \in \text{defined}([x,y]) \wedge \text{isCross}([x,y], \text{POSITIONS})\}$$

Ten jasně definuje výslednou množinu pozic jako všechna definovaná pole ve hře, která mají vlastnost isCross(). Podíváme-li se na definici této vlastnosti, tak zjistíme, že kříž

je definován jako pole, které nemá v žádném ze základních 4 směrů dostupné (volné a definované) pole. Z toho je zřejmé, že platí-li předpoklad, že množina zablokovaných polí obsahuje pouze doopravdy vyloučená pole, pak na pole s vlastností isCross nelze umístit žádnou loď dle pravidla 9.

11 Funkce whereNotToShoot()

Tvrzení: Funkce vrací seznam polí, kam nemá smysl střílet. Na vstupu předpokládá aktuální stav soupeřových lodí a seznam zásahů do vody.

Přepisovací pravidlo:

```
op whereNotToShoot(_,_) : Boats Positions -> Positions .
eq whereNotToShoot( REST, MISSES ) = (
        REST ->getBlocked()
        ( REST ->getSurround() MISSES ) ->detectCrosses()
    ) ->unique() ->sort() .
```

Důkaz: Podíváme-li se na přepisovací pravidlo, které implementuje funkci, tak je z něj jasně zřejmé, že funkce vrací pole, která byla vyhodnocena jako zablokovaná nějakou lodí či nevhodná kvůli nedostatku prostoru. Všechny použité funkce byly dříve dokázány a toto jednoduché pravidlo je plně v souladu s pravidly hry.