Politechnika Warszawska
Wydział Matematyki i Nauk Informacyjnych

Reprezentacja wiedzy

Raport z testów projektu grupy nr 4

Programy działań z akcjami współbieżnymi

Autorzy:

Dragan Łukasz
Flis Mateusz
Izert Piotr
Pielat Mateusz
Rząd Przemysław
Siry Roman
Waszkiewicz Piotr
Zawadzka Anna

14 czerwca 2016

1 Opis projektu

Tematem testowanego przez nas projektu są programy działań z akcjami współbieznymi. Rozpatrywana klasa systemów dynamicznych spełnia następujące warunki:

- Prawo inercji
- Niedeterminizm
- W języku kwerend występują akcje złożone (zbiory co najwyzej k akcji atomowych), w języku akcji jedynie akcje atomowe
- Pełna informacja o wszystkich akcjach atomowych i wszystkich ich skutkach bezpośrednich
- Z każdą akcją atomową zwiazany jest jej warunek wstepny (ew. TRUE) i końcowy (efekt akcji)
- Wykonywane są jedynie akcje bezkonfliktowe (żadne dwie akcje składowe nie mogą mieć wspólnych zmiennych, na które w jakimkolwiek stanie mają wpływ)
- Wynikiem akcji złożonej jest suma skutków wszystkich składowych akcji bezkonfliktowych
- Akcje mogą być niewykonalne w pewnych stanach; jeśli akcja jest niewykonalna, to każda akcja ją zawierająca jest niewykonalna
- Dopuszczalny jest opis częściowy zarówno stanu poczatkowego, jak i pewnych stanów wynikajacych z wykonań sekwencji akcji

Opracowywany jezyk kwerend ma za zadanie umożliwiać tworzenie zapytań, pozwalających na uzyskanie odpowiedzi na następujace pytania:

- Czy podany program P działań jest możliwy do realizacji zawsze/kiedykolwiek ze stanu początkowego?
- Czy wykonanie programu P działań w stanie początkowym prowadzi zawsze/kiedykolwiek do osiągniecia celu γ?
- Czy cel γ jest osiagalny ze stanu początkowego?

2 Przeprowadzone testy

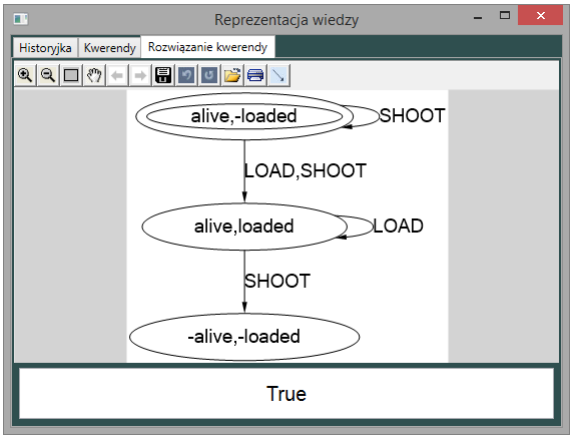
2.1 Test 1

Zdefiniowana dziedzina:

```
initially alive
initially ¬loaded
LOAD causes loaded
SHOOT causes ¬loaded if loaded
SHOOT causes ¬alive if loaded
```

Zadane kwerendy:

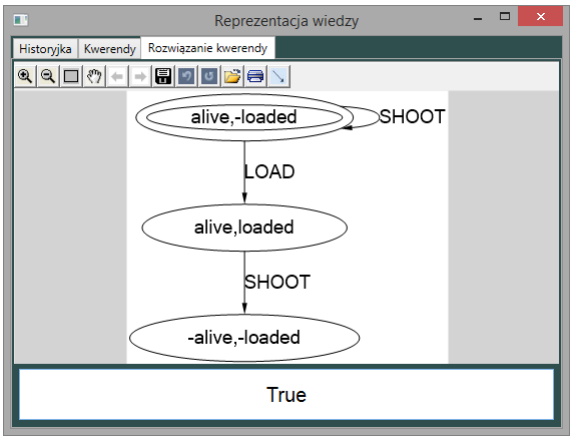
- `ever accessible ¬alive`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Jest to odpowiedź poprawna, ponieważ istnieje ciąg akcji, który prowadzi do stanu spełniającego podany cel, co ilustruje poniższy graf:



- `always accessible ¬alive`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Jest to prawda, ponieważ nie wszystkie ciągi akcji prowadzą do stanu, który spełnia podany warunek.

- `ever ¬alive after SHOOT, LOAD`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Jest to odpowiedź poprawna, ponieważ ze stanu początkowego, w którym zmienna *loaded* nie jest prawdziwa, akcje *SHOOT* i *LOAD* nie zmienią stanu zmiennej *alive*.

- `always executable SHOOT, LOAD, SHOOT`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest zgodna z prawdą - w dziedzinie nie istnieją zadania, które miałyby uniemożliwiać wykonanie ciągu podanych akcji. Wykonanie ciągu instrukcji przedstawia graf wynikowy:



2.2 Test 2

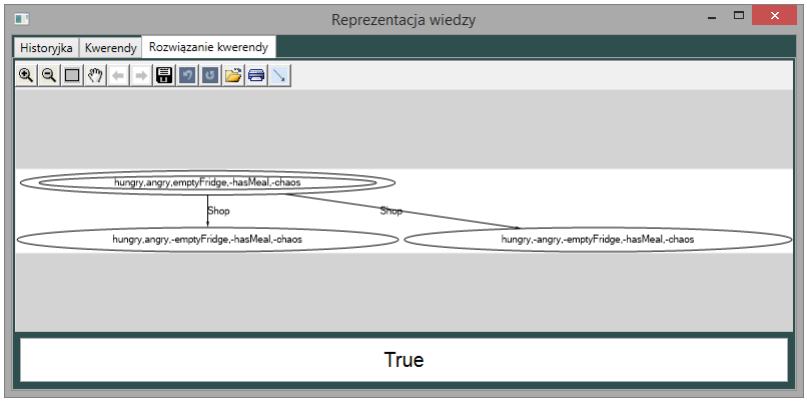
Zdefiniowana dziedzina:

```
initially hungry ∧ angry
initially emptyFridge ∧ ¬hasMeal ∧ ¬chaos
impossible Eat if ¬hasMeal
impossible Cook if emptyFridge
Shop causes ¬emptyFridge
Shop releases angry if angry
Cook causes chaos if angry
Cook causes hasMeal
Cook releases emptyFridge if emptyFridge
Eat causes ¬hasMeal ∧ ¬hungry ∧ ¬angry
```

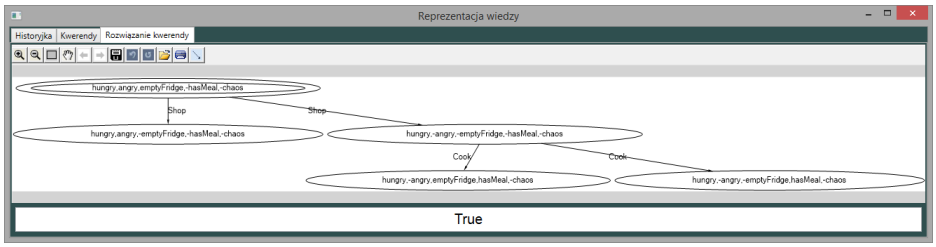
Zadane kwerendy:

- `always executable COOK, SHOP`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Odpowiedź jest poprawna, ponieważ zdanie impossible uniemożliwia wykonanie akcji $COOK$ w przypadku gdy zmienna $emptyFridge$ jest prawdziwa.

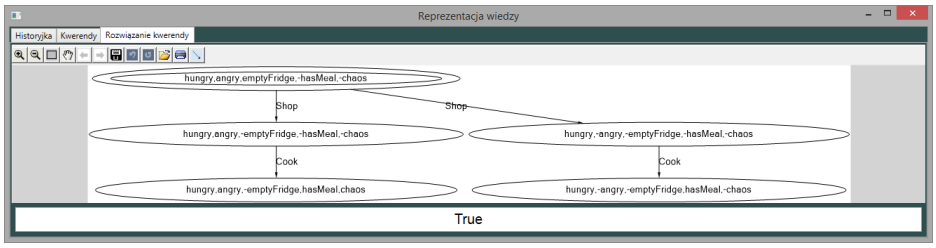
- `always executable {COOK, SHOP}`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest poprawna, ponieważ w tym wypadku zdefiniowana została akcja złożona z akcji $COOK$ i $SHOP$. Akcja $COOK$ nie jest wykonywalna ze stanu początkowego, więc pod uwagę brana jest tylko akcja $SHOP$, która jestw wykonywalna ze stanu początkowego.



- `ever accessible emptyFridge` $\wedge$ `¬angry` $\wedge$ `hungry` $\wedge$ `¬chaos`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Jest to odpowiedź poprawna, jednak nie jest zrozumiałe, dlaczego w grafie zwróconym przez program widnieją stany, które nie spełniają podanych założeń i nie należą do ścieżki, która prowadzi do takiego stanu.



- `ever ¬emptyFridge after SHOP, COOK`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest zgodna z prawdą, ponieważ akcja $COOK$ może, ale nie musi zmienić stan zmiennej $emptyFridge$.



2.3 Test 3 - kapelusznik w krainie czarów

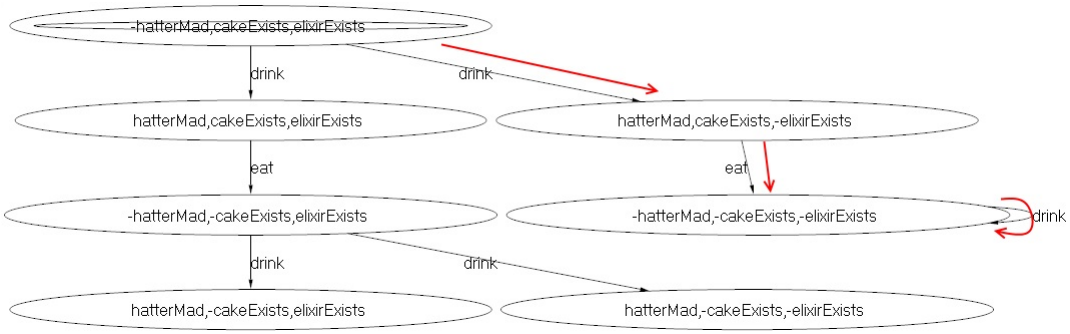
Zdefiniowana dziedzina:

```
initially ¬hatterMad ∧ cakeExists ∧ elixirExists
drink causes hatterMad if elixirExists
eat causes ¬hatterMad
impossible eat if ¬cakeExists
drink releases elixirExists if elixirExists
eat causes ¬cakeExists
```

Zadane kwerendy:

- `ever cakeExists after eat`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Odpowiedź jest poprawna, ponieważ jedzenie zawsze powoduje brak ciastka, a w dziedzinie nie ma możliwości przywrócenia ciastka.
- `always executable drink, drink`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest poprawna, ponieważ nawet jeśli eliksir się skończy, akcja picia jest wykonywalna.
- `always accessible hatterMad`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest poprawna.
- `ever  hatterMad after drink, eat, drink, eat`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Odpowiedź jest poprawna - druga próba zjedzenia ciastka jest niewykonalna.

- `ever ¬hatterMad after drink, eat, { drink, eat}`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest poprawna, ścieżkę można zobaczyć na poniższym obrazku.



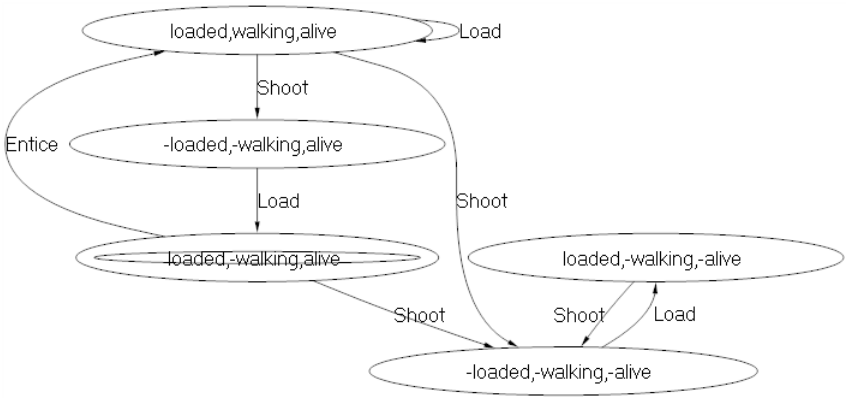
2.4 Test 4 - historia z indykiem i strzelającym Sebastianem

Zdefiniowana dziedzina:

```
initially loaded & ¬walking & alive
Shoot causes ¬loaded
Unload causes ¬loaded
Load causes loaded
Shoot causes ¬alive if ¬walking
Shoot releases alive if loaded & alive & walking
Shoot causes ¬walking if loaded
Entice causes walking
impossible Entice if ¬alive
```

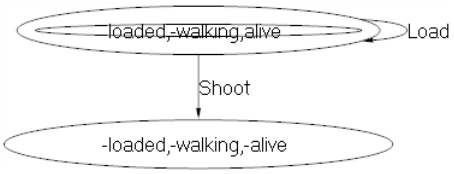
Zadane kwerendy:

- `ever alive after Entice, Shoot, Load, Entice, Load, Shoot, Load, Shoot`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Odpowiedź jest poprawna, ponieważ indyk zaczyna chodzić po pierwszej akcji Entice co powoduje, że strzał może się nie udać. Jednak po strzale oddanym z nabitej broni indyk przestaje się ruszać (boi się). W takim wypadku strzał zawsze uśmierci biedne zwierzę.

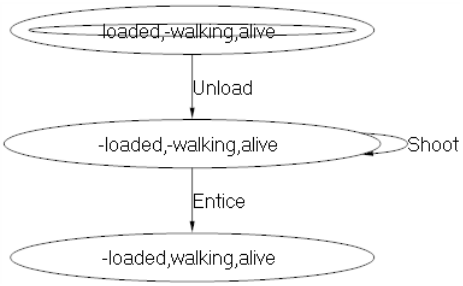


- `ever executable Load, Shoot, Entice`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Odpowiedź jest poprawna, ponieważ w przypadku naładowanej broni,

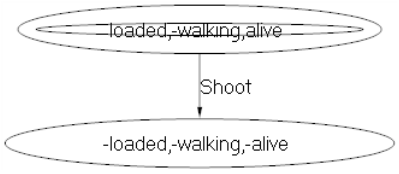
kiedy indyk się nie porusza, strzał zawsze go uśmierci. Nie można wykonać akcji Entice w przypadku uśmierconego indyka.



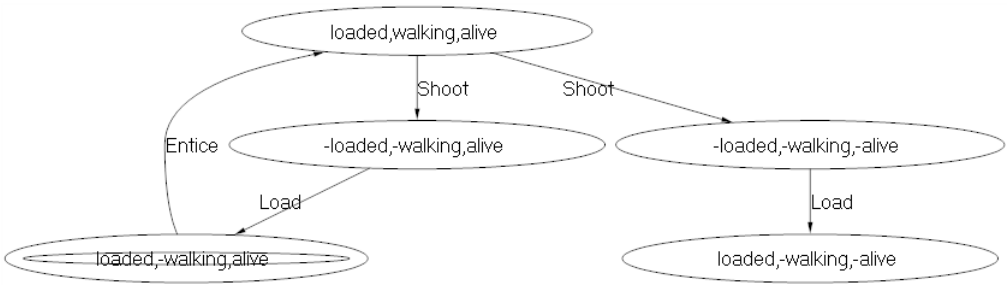
- `ever executable Unload, Shoot, Entice`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest poprawna. W przeciwieństwie do poprzedniej kwerendy, tutaj akcja Unload rozładowuje broń. Strzał z takiej broni nie uśmierci zwierzaka.



- `ever accessible  alive`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest poprawna - indyka zawsze można w pewien sposób uśmiercić.



- `ever walking after Entice, Shoot, Load, Entice`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Odpowiedź jest poprawna - jeśli indyk się porusza strzał może się nie udać. Ponowne wykonanie akcji Entice spowoduje dalsze chodzenie zwierzaka.


loaded,walking,alive
Shoot
Shoot
Entice
-loaded,-walking,alive
-loaded,-walking,-alive
Load
Load
loaded,-walking,alive
loaded,-walking,-alive

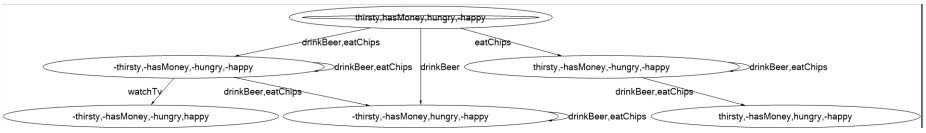
2.5 Test 5 - Piwo i czipsy

Stefan prowadzi mało zdrowy tryb życia: lubi jeść czipsy, pić piwo i oglądać telewizję. Z tego powodu ma mało pieniędzy. Do sklepu daleko, więc jak Stefan raz pojedzie na zakupy, to nie ma już pieniędzy na kolejny wypad do sklepu. Kiedy je czipsy, przestaje być głodny. Pragnienie może zaspokoić piwkiem, ale istnieje wtedy szansa, że znowu zgłodnieje. Stefan lubi również oglądać telewizję, ale robi to tylko wtedy, gdy nie jest ani głodny, ani spragniony. Szczęście Stefanowi zapewnia bezpośrednio tylko oglądanie telewizji. Obecnie Stefan jest i głodny, i spragniony, ma forsę ale nie jest szczęśliwy.Czy uśmiech ma szansę kiedykolwiek zagościć na jego twarzy?
Zdefiniowana dziedzina:

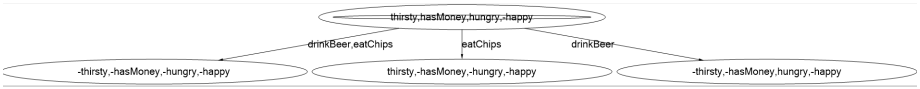
```
drinkBeer causes ¬thirsty & ¬hasMoney if hasMoney
drinkBeer releases hungry if ¬hungry
eatChips causes ¬hungry & ¬hasMoney if hasMoney

watchTv causes happy if ¬hungry & ¬thirsty
impossible watchTv if hungry | thirsty

initially hungry & thirsty & ¬happy & hasMoney
```

Zadane kwerendy:

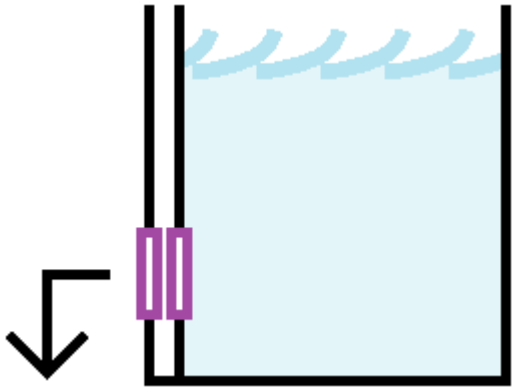
- `ever accessible happy`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Można dostać się do stanu *happy*, ale tylko, jeżeli wcześniej na raz wykona się akcje *drinkBeer* i *eatChips*.



- `always accessible happy & hungry`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `TRUE`
  Program daje dobry wynik, ale graf wcale tego nie potwierdza: żaden z wierzchołków grafu nie zawiera stanu spełniającego *happy&hungry*



2.6 Test 6



Rozważmy zbiornik z wodą. Posiada on dwie szeregowo zainstalowane śluzy umożliwiające spuszczanie z niego zawartości. Śluzy można dowolnie otwierać i zamykać, przy czym zbiornik opróżnia się tylko wtedy, gdy obie śluzy są otwarte jednocześnie. Nie przewidujemy napełniania raz opróżnionego zbiornika. Początkowo obie śluzy są zamknięte, a zbiornik wypełniony.

Zdefiniowana dziedzina:

initially $\neg openedA \wedge \neg openedB \wedge water$
OpenA **causes** $openedA$
OpenB **causes** $openedB$
OpenA **causes** $\neg water$ **if** $openedB$
OpenB **causes** $\neg water$ **if** $openedA$

(dziedzina początkowo miała wykorzystywać zdanie **nonintertial**, jednak brak obsługi przez system zdania integralności **always** uniemożliwia jego praktyczne zastosowanie)

Zadane kwerendy:

- *Czy sekwencyjne otwarcie śluz sprawia, że woda pozostaje w zbiorniku?*

  `ever` *water* `after` OpenA, OpenB
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`

- *Czy jednoczesne otwarcie śluz sprawia, że woda pozostaje w zbiorniku?*

  `ever` *water* `after` {OpenA, OpenB}
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `TRUE`

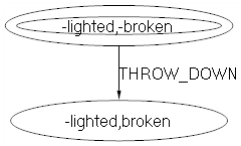
2.7 Test 7

Zdefiniowana dziedzina:

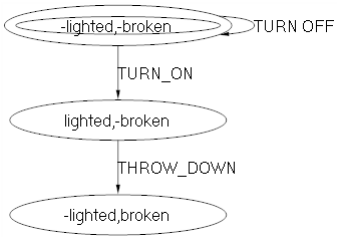
```
initially ¬lighted & ¬broken
THROW-DOWN causes broken & ¬lighted
TURN-OFF causes ¬lighted
TURN-ON causes lighted
impossible TURN-ON if broken
```

Zadane kwerendy:

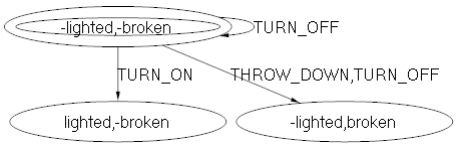
- `ever executable THROW-DOWN, TURN-ON`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Jest to odpowiedź poprawna, ponieważ po zrzuceniu lampki nie można już jej włączyć (bo jest rozbita), co ilustruje poniższy graf:



- `ever lighted after TURN-OFF, TURN-ON, THROW-DOWN`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `FALSE`
  Jest to prawda, ponieważ zrzucenie lampki, poza jej rozbiciem, powoduje również to, że przestaje się ona świecić.



- `always accessible lighted`
  Oczekiwana odpowiedź: `FALSE`
  Odpowiedź programu: `TRUE`
  Nie każdy ciąg akcji spełnia warunek, że lampka jest zapalona - po wykonaniu akcji `THROW-DOWN` system nigdy nie znajdzie się już w stanie spełniającym warunek `lighted`.


-lighted,-broken
TURN_OFF
TURN_ON
THROW_DOWN,TURN_OFF
lighted,-broken
-lighted,broken

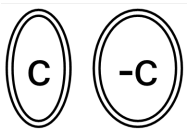
2.8 Test 8

Zdefiniowana dziedzina:

```
initially c | ¬c
```

Zadane kwerendy:

- `always accessible c`
  Oczekiwana odpowiedź: `TRUE`
  Odpowiedź programu: `FALSE`
  Program przyjmuje dziedzinę zawierającą 2 stany początkowe a następnie zwraca zły wynik kwerendy.



2.9 Test 9

3 Wnioski