Reprezentacja Wiedzy
Projekt nr 4

Realizacja efektywnych programów działań

Zespół:
Adam Komorowski
Krzysztof Adamiak
Krzysztof Kachniarz
Maciej Kachniarz
Mateusz Kilos
Tomasz Luśtyk
Paweł Paradowski
Katarzyna Żuk

16 kwietnia 2014

Spis treści

1 Przedstawienie problemu

Celem projektu było zaprojektowanie klasy systemów dynamicznych spełniających następujące warunki:

1. Prawo inercji.
2. Pełna informacja o wszystkich akcjach i wszystkich skutkach bezpośrednich.
3. Sekwencyjność i niedeterminizm działań.
4. Z każdą akcją związany jest
   - warunek wstępny,
   - warunek wynikowy,
   - koszt realizacji.
5. Skutki akcji zależą od
   - stanu początkowego jej wykonania,
   - kosztu jej realizacji.
6. W pewnych stanach akcja może być niewykonalna.
7. Warunki integralności wpływają na wykonalność akcji.
8. Dopuszczalny jest opis częściowy zarówno stanu początkowego, jak i pewnych stanów wynikających z wykonań sekwencji akcji.

2 Syntaktyka języka

Mamy dane następujące zbiory:

- Ac, oznaczający zbiór wszystkich akcji.
- F, oznaczający zbiór wszystkich fluentów.
- $F_I \subseteq F$, oznaczający zbiór wszystkich fluentów inertnych.
- P, zbiór możliwych stanów zasobu.
- Dla każdego $A_i \in Ac$ istnieje skończony zbiór $C_{A_i} \subseteq P$, oznaczający zbiór cen akcji A_i. Zbiór $\{C_{A_i} : A_i \in Ac\}$ będziemy oznaczać jako C.

Nasz język bedzie charakteryzowany za pomocą sygnatury: (Ac, F, P, C).

Definicja 2.1. *Formułą* nazywamy wyrażenie spełniające warunki:

- $\top$ i $\bot$ są formulami (pierwsza oznacza formułę zawsze prawdziwą, druga formułę zawsze fałszywą).
- Jeśli $f \in F$, to f jest formułą;
- Jeśli α jest formułą, to $\neg\alpha$ jest formułą;
- Jeśli α, β są formułami, to $\alpha \vee \beta$ jest formułą;
- Jeśli α, β są formułami, to $\alpha \wedge \beta$ jest formułą;
- Jeśli α, β są formułami, to $\alpha \rightarrow \beta$ jest formułą;
- Jeśli α, β są formułami, to $\alpha \leftrightarrow \beta$ jest formułą;

2.1 Język akcji

Wyrażenia naszego języka akcji są postaci:

1. α **after** $A_1, ..., A_n$, wyrażenie to oznacza, że wykonanie ciągu akcji: $A_1, ..., A_n$ ze stanu początkowego z początkowym stanem zasobu, prowadzi zawsze do osiągnięcia stanu spełniającego warunek logiczny α.
2. **initially** α, równoważne: α **after** $A_1, ..., A_n$ z $n = 0$, wyrażenie to oznacza że stan początkowy spełnia warunek logiczny α.

3. **obsevable** α **after** $A_1, ..., A_n$, wyrażenie to oznacza, że wykonanie ciągu akcji: $A_1, ..., A_n$ ze stanu początkowego z początkowym stanem zasobu, może prowadzić (prowadzi czasem) do osiągnięcia stanu spełniającego warunek logiczny α.

4. A **costs** p **if** π **with effect** α, wyrażenie to oznacza, że wykonanie akcji A ze stanu spełniającego warunek logiczny π przy koszcie akcji p prowadzi do osiągnięcia stanu spełniającego warunek logiczny α.

5. A **costs** p **with effect** α, wyrażenie to jest równoważne wyrażeniu A **costs** p **if** $\top$ **with effect** α, co oznacza że wykonanie akcji A z dowolnego stanu przy koszcie akcji p prowadzi do osiągnięcia stanu spełniającego warunek logiczny α.

6. A **costs** p **if** π, wyrażenie oznacza, że wykonanie akcji A ze stanu spełniającego warunek logiczny π jest wykonywane kosztem akcji co najmniej p.

7. A **costs** p, wyrażenie to jest równoważne: A **costs** p **if** $\top$, co oznacza, że wykonanie akcji A z dowolnego stanu jest wykonywane kosztem akcji co najmniej p.

8. A **if** π **with effect** α, wyrażenie to oznacza, że wykonanie akcji A na stanie spełniającym warunek logiczny π, prowadzi do stanu spełniającego warunek: α przy zerowym koszcie.

9. A **with effect** α, wyrażenie to jest równoważne:A **causes** α **if** $\top$, co oznacza, że wykonanie akcji A w dowolnym stanie prowadzi do stanu spełniającego warunek: α przy zerowym koszcie.

10. **impossible** A **if** π, wyrażenie to jest równoważne: A **if** π **with effect** $\bot$, co oznacza, że wykonanie akcji A w stanie spełniającym π jest niemożliwe.

11. A **releases** f **if** π, wyrażenie to oznacza, że wykonanie akcji A na stanie spełniającym warunek logiczny π może (ale nie musi) doprowadzić do zmiany wartości fluentu inertnego f.

12. A **releases** f, wyrażenie to jest równoważne A **releases** f **if** $\top$, co oznacza, że wykonanie akcji A na dowolnym stanie może (ale nie musi) doprowadzić do zmiany wartości fluentu inertnego f.

13. **always** α, wyrażenie to oznacza, że dowlony stan ukladu spełnia warunek logiczny α.

14. **noninertial** f, wyrażenie to oznacza, że fluent f jest nieinertny.

15. A **preserves** f **if** π, wyrażenie to oznacza, że wykonanie akcji A w stanie spełniającym warunek logiczny π nie może zmienić wartości fluentu f.

16. **initialy** p **value**, wyrażenie to oznacza, że początkowy stan zasobu wynosi p.

3 Semantyka języka

Definicja 3.1. *Stanem* nazywamy funkcję: $\sigma : F \to \{0, 1\}$.

Uwaga 3.1. Definicję stanu można rozszerzyć na zbiór wszystkich formuł nad zbiorem fluentów F do funkcji σ^* w następujący sposób:

- $\sigma^*(\top) = 1$.
- $\sigma^*(\bot) = 0$.
- $\sigma^*(f) = \sigma(f)$.
- $\sigma^*(\neg\alpha) = 1 - \sigma^*(\alpha)$.
- $\sigma^*(\alpha \vee \beta) = max(\sigma^*(\alpha), \sigma^*(\beta))$.
- $\sigma^*(\alpha \wedge \beta) = min(\sigma^*(\alpha), \sigma^*(\beta))$.
- $\sigma^*(\alpha \to \beta) = \sigma^*(\neg\alpha \vee \beta)$.
- $\sigma^*(\alpha \leftrightarrow \beta) = \sigma^*((\alpha \to \beta) \wedge (\beta \to \alpha))$.

gdzie α, β są formułami nad F i $f \in F$.
Jeśli $\sigma^*(\alpha) = 1$, to piszemy: $\sigma \models \alpha$.

Definicja 3.2. *Strukturą* naszego języka jest piątka: $S = (\Sigma, P, \sigma_0, p_0, Res)$ taka, że:

1. $\Sigma \neq \emptyset$ jest podzbiorem wszystkich stanów.
2. σ_0 jest stanem początkowym układu.
3. p_0 jest początkowym stanem zasobu.
4. $Res : \Sigma \times P \times Ac \to 2^{\Sigma \times P}$ zwana funkcją tranzycji.

Definicja 3.3. Na strukturze $S = (\Sigma, P, \sigma_0, p_0, Res)$ definiujemy częściową funkcję $\Psi_S : Ac* \times \Sigma \times P \to \Sigma \times P$ spełniającą następujące warunki:

- $\Psi_S(\epsilon, \sigma, p) = (\sigma, p)$.
- Jeśli dla $n \geq 1$, $\Psi_S((A_1, ..., A_n), \sigma, p)$ jest zdefiniowane,
- Jeżeli $Res((A_1, ..., A_n, A_{n-1}), \sigma, p) \neq \emptyset$, to
  $\Psi_S((A_1, ..., A_n, A_{n+1}), \sigma, p) \in Res(A_{n+1}, \Psi_S((A_1, ..., A_n), \sigma, p))$.

Definicja 3.4. Podamy pewne podstawowe definicje:

- Fluent f jest inertny w D wtedy i tylko wtedy gdy **(noninertial** f**)**$\notin D$
- $\sigma : F \to \{0,1\}$ jest stanem w D, jeśli dla każdego wyrażenia **(always** α**)**$\in D$ zachodzi $\sigma \models \alpha$.
- Wyrażenie α **after** $A_1, ..., A_n$ jest prawdziwe w S wtedy i tylko wtedy gdy, dla dowolnej funkcji Ψ_S zdefiniowanej wcześniej $\Psi_S((A_1, ..., A_n), \sigma_0, p_0) \models \alpha$.
- Wyrażenie **observalable** α **after** $A_1, ..., A_n$ jest prawdziwe w S wtedy i tylko wtedy gdy, dla pewnej funkcji Ψ_S zdefiniowanej wcześniej $\Psi_S((A_1, ..., A_n), \sigma_0, p_0) \models \alpha$.
- Wyrażenie **initialy** p **value** jest prawdziwe w S wtedy i tylko wtedy gdy, $p_0 = p$.

Definicja 3.5. Niech D będzie domeną akcyjną i $S = (\Sigma, P, \sigma_0, p_0, Res)$ będzie strukturą naszego języka. Wtedy możemy zdefiniować przekształcenie: $Res_0 : Ac \times \Sigma \times P \to 2^{\Sigma \times P}$, takie że dla dowolnej funkcji $A \in Ac$, dla dowolnego $p \in P$ i dla dowolnego stanu $\sigma \in \Sigma$:

$$Res_0(A, \sigma, p) = \{(\sigma', p) \in \Sigma \times P : (\ (A \textbf{ causes } \alpha \textbf{ if } \pi)\in D \wedge \sigma \models \pi) \Rightarrow \sigma' \models \alpha\} \cup \{(\sigma', p - p_A) \in \Sigma \times P : ((\ (A \textbf{ cost } p \textbf{ if } \pi \textbf{ with effect } \alpha)\in D \wedge \sigma \models \pi) \Rightarrow \sigma' \models \alpha) \wedge p_A \in C_A\}.$$

Uwaga 3.2. Dodatkowo dla każdej pary stanów $\sigma, \sigma' \in \Sigma$ i dla dowolnej akcji $A \in Ac$ definiujemy zbiór

$$New(A, \sigma, \sigma') = \{\bar{f} : \sigma' \models \bar{f} \wedge ((f \in F_I \wedge \sigma(f) \neq \sigma'(f)) \vee (\text{Dla pewnego wyrażenia } (A \textbf{ releases } f \textbf{ if } \pi)\in D \text{ zachodzi } \sigma \models \pi))\}.$$

Definicja 3.6. Oznaczmy D^- jako D bez wszystkich wyrażeń postaci: A **preserves** f **if** π. Niech $S = (\Sigma, P, \sigma_0, p_0, Res_{RW})$ będzie strukturą naszego języka. Mówimy, że S jest modelem w D^- wtedy i tylko wtedy kiedy zachodzą warunki:

- Σ jest zbiorem wszystkich stanów w D^-.
- Każde wyrażenie postaci: **(**α **after** $A_1, ..., A_n$**)**, **(obsevable** α **after** $A_1, ..., A_n$**)** i **(initialy** p **value)** z D^- są prawdziwe w S.
- Dla każdego $A \in Ac$, dla każdego $\sigma \in \Sigma$ i dla każdego $p \in P$,

$$Res_{RW}(A, \sigma, p) = \{(\sigma', p') \in Res_0 : New(A, \sigma, \sigma') \text{ jest minimalny}\}.$$

Zbiór takich modeli oznaczamy symbolem: $Mod(D^-)$.

Definicja 3.7. Struktura $S = (\Sigma, P, \sigma_0, p_0, Res) \in Mod(D^-)$ jest modelem w D wtedy i tylko wtedy gdy dla dowolnej akcji $A \in Ac$, dla dowolnego stanu $\sigma \in \Sigma$ i dla dowolnego stanu zasobu $p \in P$:

$$Res(A, \sigma, p) = \{(\sigma', p') \in Res_{RW} : (A \textbf{ preserves } f \textbf{ if } \pi) \in D \wedge (\sigma \models \pi \Rightarrow (\sigma(f) = \sigma'(f)))\}.$$

4 Język kwerend

4.1 Q1

Kwerenda odpowiada na pytanie, czy podany program działań $A_1, \ldots, A_n$ jest realizowalny zawsze/kiedykolwiek z każdego stanu spełniającego warunek π przy podanym/dowolnym koszcie.

4.1.1 Czy program działań jest zawsze realizowalny?

- Jeżeli kwerenda Q jest postaci:

  always executable $A_1, \ldots, A_n$ **from** π **with** p **for** c,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$ i dla każdego odwzorowania Ψ_S, jeżeli $c \leq p$ to suma kosztów wykonania akcji $A_1, \ldots, A_n$ oraz $\sigma \vDash \pi$, to $\Psi_S((A_1, \ldots, A_n), \sigma, p)$ jest definiowalne.

  Innymi słowy – zawsze w każdym możliwym stanie spełniającym π i z zasobem wielkości p możemy wykonać ciąg akcji $A_1, \ldots, A_n$, którego koszt jest równy c.

- Jeżeli kwerenda Q jest postaci:

  always executable $A_1, \ldots, A_n$ **from** π **with** p,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$ i dla każdego odwzorowania Ψ_S, jeżeli $\sigma \vDash \pi$, to $\Psi_S((A_1, \ldots, A_n), \sigma, p)$ jest definiowalne.

  Innymi słowy – zawsze w każdym możliwym stanie spełniającym π i z zasobem wielkości p możemy wykonać ciąg akcji $A_1, \ldots, A_n$ o dowolnym koszcie.

- Jeżeli kwerenda Q jest postaci:

  always executable $A_1, \ldots, A_n$ **for** c,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego odwzorowania Ψ_S i dla $c \leq p$ równego sumie kosztów wykonania akcji $A_1, \ldots, A_n$ – $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0)$ jest definiowalne.

  Innymi słowy – zawsze ze stanu początkowego możemy wykonać ciąg akcji $A_1, \ldots, A_n$, którego koszt jest równy c.

- Jeżeli kwerenda Q jest postaci:

  always executable $A_1, \ldots, A_n$,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego odwzorowania Ψ_S – $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0)$ jest definiowalne.

  Innymi słowy – zawsze ze stanu początkowego możemy wykonać ciąg akcji $A_1, \ldots, A_n$ o dowolnym koszcie.

4.1.2 Czy program działań jest kiedykolwiek realizowalny?

- Jeżeli kwerenda Q jest postaci:

  sometimes executable $A_1, \ldots, A_n$ **from** π **with** p **for** c,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$ i dla pewnego odwzorowania Ψ_S, jeżeli $c \leq p$ to suma kosztów wykonania akcji $A_1, \ldots, A_n$ oraz $\sigma \vDash \pi$, to $\Psi_S((A_1, \ldots, A_n), \sigma, p)$ jest definiowalne.

  Innymi słowy – czasami możliwe jest wykonanie ciągu akcji $A_1, \ldots, A_n$ o koszcie równym c z dowolnego stanu spełniającego π i z zasobem równym p.

- Jeżeli kwerenda Q jest postaci:

  sometimes executable $A_1, \ldots, A_n$ **from** π **with** p,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$ i dla pewnego odwzorowania Ψ_S, jeżeli $\sigma \vDash \pi$, to $\Psi_S((A_1, \ldots, A_n), \sigma, p)$ jest definiowalne.

  Innymi słowy – czasami możliwe jest wykonanie ciągu akcji $A_1, \ldots, A_n$ o dowolnym koszcie z dowolnego stanu spełniającego π i z zasobem równym p.

- Jeżeli kwerenda Q jest postaci:

  sometimes executable $A_1, \ldots, A_n$ **for** c,

  wtedy $D \mid\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla pewnego odwzorowania Ψ_S i dla $c \leq p$ równego sumie kosztów wykonania akcji $A_1, \ldots, A_n$ – $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0)$ jest definiowalne.

  Innymi słowy – czasami możliwe jest wykonanie ciągu akcji $A_1, \ldots, A_n$ o koszcie równym c ze stanu początkowego.

- Jeżeli kwerenda Q jest postaci:

  sometimes executable $A_1, \ldots, A_n$,

  wtedy $D \mid\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla pewnego odwzorowania Ψ_S – $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0)$ jest definiowalne.

  Innymi słowy – czasami możliwe jest wykonanie ciągu akcji $A_1, \ldots, A_n$ o dowolnym koszcie ze stanu początkowego.

4.2 Q2

Kwerenda odpowiada na pytanie, czy podany program działań $A_1, \ldots, A_n$ prowadzi zawsze/kiedykolwiek do osiągnięcia celu γ ze stanu początkowego/spełniającego warunek π.

4.2.1 Czy program działań zawsze prowadzi do osiągnięcia celu γ?

- Jeżeli kwerenda Q jest postaci:

  necessary γ **after** $A_1, \ldots, A_n$ **from** π **with** p,

  wtedy $D \mid\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$, dla każdego odwzorowania Ψ_S, jeżeli $\sigma \models \pi$, to $\Psi_S((A_1, \ldots, A_n), \sigma, p) \models \gamma$.

  Innymi słowy – ciąg akcji $A_1, \ldots, A_n$ zawsze prowadzi z dowolnego stanu spełniającego π o zasobie p do stanu spełniającego γ. W przypadku $p = 0$ oznacza to, że podany ciąg prowadzi do osiągnięcia γ z dowolnym stanem zasobu.

- Jeżeli kwerenda Q jest postaci:

  $$\textbf{necessary } \gamma \textbf{ after } A_1, \ldots, A_n,$$

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D oraz dla każdego odwzorowania Ψ_S zachodzi:
  $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0) \vDash \gamma$.

  Oznacza to, że podany program działań zawsze prowadzi ze stanu początkowego do celu γ.

4.2.2 Czy program działań kiedykolwiek prowadzi do osiągnięcia celu γ?

- Jeżeli kwerenda Q jest postaci:

  $$\textbf{possibly } \gamma \textbf{ after } A_1, \ldots, A_n \textbf{ from } \pi \textbf{ with } p,$$

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$ i dla pewnego odwzorowania Ψ_S, jeżeli $\sigma \vDash \pi$, to $\Psi_S((A_1, \ldots, A_n), \sigma, p) \vDash \gamma$.

  Innymi słowy – ciąg akcji $A_1, \ldots, A_n$ czasami prowadzi z dowolnego stanu spełniającego π o zasobie p do stanu spełniającego γ. W przypadku $p = 0$ oznacza to, że podany ciąg prowadzi do osiągnięcia γ z dowolnym stanem zasobu.

- Jeżeli kwerenda Q jest postaci:

  $$\textbf{possibly } \gamma \textbf{ after } A_1, \ldots, A_n,$$

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D i dla pewnego odwzorowania Ψ_S zachodzi:
  $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0) \vDash \gamma$.

  Oznacza to, że podany program działań czasami prowadzi ze stanu początkowego do celu γ.

4.3 Q3

Kwerenda odpowiada na pytanie, przy jakim koszcie cel γ jest zawsze lub kiedykolwiek osiągalny ze stanu początkowego lub spełniającego warunek π.

4.3.1 Przy jakim koszcie cel γ jest zawsze osiągalny?

- Jeżeli kwerenda Q jest postaci:

  always cost of γ **from** π,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$, dla każdego odwzorowania Ψ_S istnieje taki program działań $(A_1, \ldots, A_n)$, że jeżeli $\sigma \vDash \pi$, to:
  $\Psi_S((A_1, \ldots, A_n), \sigma, \infty)$ jest definiowalna i $\Psi_S((A_1, \ldots, A_n), \sigma, \infty) \vDash \gamma$.

  Odpowiedzią na powyższą kwerendę jest maksymalna możliwa suma kosztów wykonania znalezionych akcji $A_1, \ldots, A_n$ z każdego ze stanów spełniających π i prowadzących do γ.

- Jeżeli kwerenda Q jest postaci:

  always cost of γ,

  wtedy $D \models\!\!\approx Q$ wtw. dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0 = \infty, Res)$ dziedziny D i dla każdego odwzorowania Ψ_S istnieje taki program działań $(A_1, \ldots, A_n)$, że zachodzi: $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0 = \infty)$ jest definiowalna oraz $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0 = \infty) \vDash \gamma$.

  Odpowiedzią na powyższą kwerendę jest maksymalna możliwa suma kosztów wykonania ze stanu początkowego znalezionych akcji $A_1, \ldots, A_n$, które prowadzą do γ.

4.3.2 Przy jakim koszcie cel γ jest kiedykolwiek osiągalny?

- Jeżeli kwerenda Q jest postaci:

  sometimes cost of γ **from** π,

  wtedy $D \models\!\!\approx Q$ wtw. istnieje program działań $(A_1, \ldots, A_n)$ taki, że dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0, Res)$ dziedziny D, dla każdego stanu $\sigma \in \Sigma$, dla pewnego odwzorowania Ψ_S, jeżeli $\sigma \vDash \pi$, to $\Psi_S((A_1, \ldots, A_n), \sigma, \infty)$ jest definiowalna oraz zachodzi:
  $\Psi_S((A_1, \ldots, A_n), \sigma, \infty) \vDash \gamma$.

Odpowiedzią na powyższą kwerendę jest suma kosztów wykonania znalezionego ciągu akcji $A_1, \ldots, A_n$, który czasami jest możliwy do wykonania z dowolnego stanu spełniającego π.

- Jeżeli kwerenda Q jest postaci:

  sometimes cost of γ,

  wtedy $D \not\models Q$ wtw. istnieje program działań $(A_1, \ldots, A_n)$ taki, że dla każdego modelu $S = (\Sigma, P, \sigma_0, p_0 = \infty, Res)$ dziedziny D i dla pewnego odwzorowania Ψ_S zachodzi: $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0 = \infty)$ jest definiowalna oraz $\Psi_S((A_1, \ldots, A_n), \sigma_0, p_0 = \infty) \models \gamma$

  Odpowiedzią na powyższą kwerendę jest suma kosztów wykonania znalezionego ciągu akcji $A_1, \ldots, A_n$, który czasami jest możliwy do wykonania ze stanu początkowego.

5 Wygląd programu

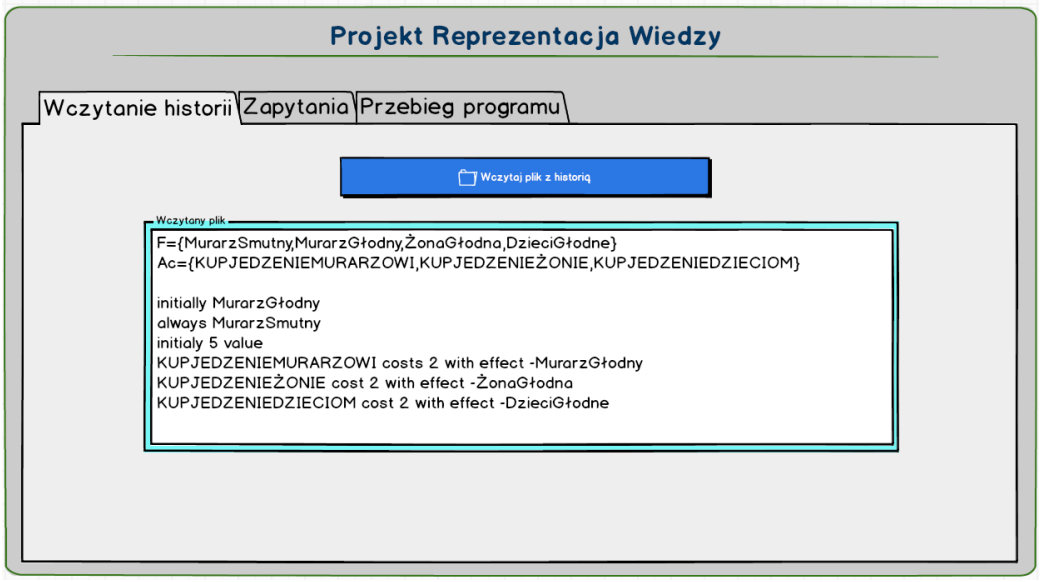
W przygotowanym przez nas programie będziemy mieć do dyspozycji trzy zakładki:

- Wczytanie historii
- Zapytania
- Przebieg programu

Każda z nich będzie udostępniała fragment funkcjonalności.

5.1 Zakładka *Wczytanie historii*

W tej zakładce będziemy mogli wczytać plik z zapisanymi danymi wejściowymi. Po naciśnięciu przycisku „Wczytaj plik z historią", otworzone zostanie okno dialogowe, które pozwoli nam na wybranie pliku z dysku. Plik wejściowy będzie zbudowany podobnie jak wczytany plik na przykładzie poniżej:



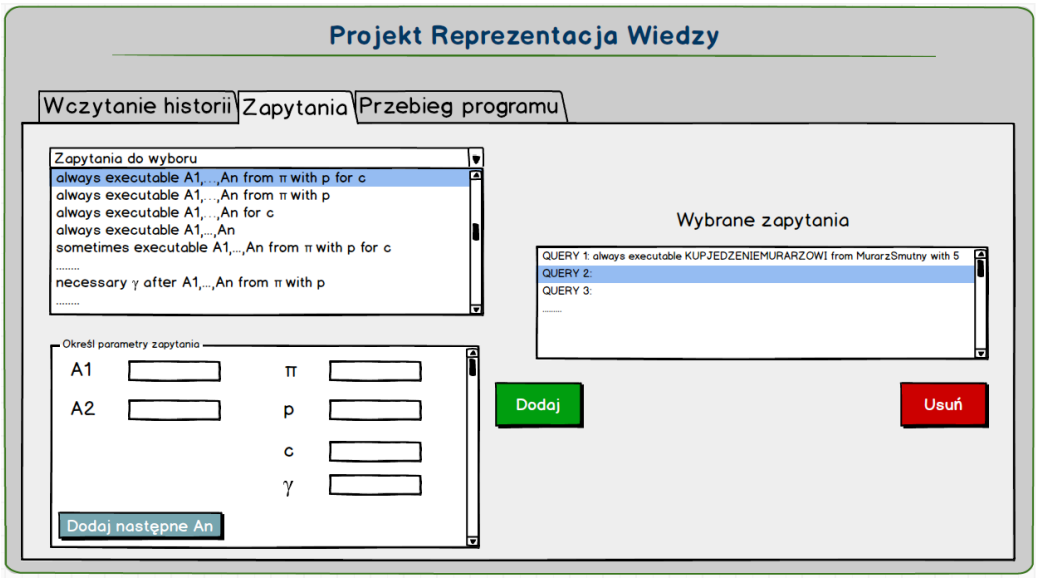
Pierwszy jego wiersz to zbiór fluentów. Fluenty oddzielone są między sobą przecinkami. Analogicznie, drugi wiersz zawiera zbiór akcji dostępnych do wykonania w programie. Trzeci wiersz – linia odstępu (dla zwiększenia czytelności pliku). Od czwartego wiersza zaczynają się inicjujące program instrukcje.

5.2 Zakładka *Zapytania*

W tej zakładce będziemy mogli ustalić kwerendy, na które będzie odpowiadał nasz program. Te kwerendy możemy wybrać z listy wszystkich dostępnych kwerend (dostępna w zakładce pod nazwą „Zapytania do wyboru"). Po wybraniu odpowiadającej nam kwerendy, musimy doprecyzować parametry w niej określone. Stąd poniżej znajduje się okno do wpisania wartości odpowiednich parametrów. Jeśli pole pozostaje puste, oznacza to, że dany symbol nie występuje w naszej kwerendzie. W przeciwnym wypadku powinien zostać wyświetlony komunikat, że nie wszystkie parametry zostały uzupełnione.

Dodatkowo, w oknie określania parametrów znajduje się przycisk „Dodaj następne A_n", który pozwoli na dodanie kolejnych pól na akcje A_2, A_3, A_4, ..., A_n. Jedno kliknięcie przycisku, powoduje pojawienie się kolejnego przycisku A_x gdzie A_{x-1} to ostatnie wyświetlane w oknie pole.

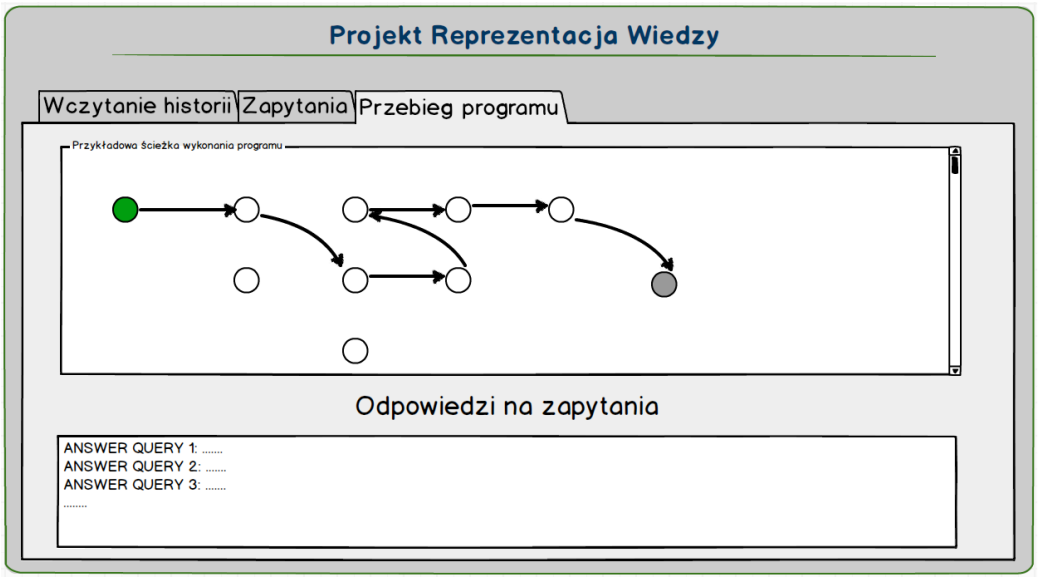
Po wypełnieniu parametru, możemy wcisnąć przycisk „Dodaj", co powoduje, że wypełniona kwerenda trafi na listę „Wybrane zapytania". W każdym momencie możemy usunąć z listy „Wybrane zapytania" kwerendę. Wystarczy ją nacisnąć i wcisnąć przycisk „Usuń".



5.3 Zakładka *Przebieg programu*

W tej zakładce możemy zobaczyć:

- Przykładową ścieżkę wywołania programu,
- Odpowiedzi na wybrane przez nas kwerendy (określone w zakładce „Zapytania").



6 Przykłady

6.1 Przykład 1

Rozważmy następującą historię:

W średniowiecznym królestwie mieszkał sobie głodny murarz. Miał żonę i dużo dzieci. Pewnego jesiennego, deszczowego dnia murarz stracił pracę i był z tego powodu bardzo smutny i był już smutny całą zimę. Zdał sobie wtedy sprawę, że całą zimę będą musieli (on i jego rodzina) przeżyć tylko za 5 złotych monet (wtedy było to trochę więcej niż dzisiaj, ale też mało). W tych czasach kupienie jedzenia na całą zimę dla murarza lub jego żony kosztowało 2 złote monety. Wykarmienie wszystkich dzieci w ciągu zimy też kosztowało 2 złote monety. Kupienie jedzenia sprawiało, że ten dla którego kupiono jedzenie nie byl głodny tej zimy Czy murarz zdołał wykarmić całą rodzinę tej zimy?

Ten problem można przedstawić w postaci następującego zagadnienia w naszym języku:

1. **initially** `MurarzGłodny`
2. **always** `MurarzSmutny`
3. **initialy** 5 **value**
4. KUP_JEDZENIE_MURARZOWI *costs* 2 *with effect* ¬ `MurarzGłodny`
5. KUP_JEDZENIE_ŻONIE *costs* 2 *with effect* ¬ `ŻonaGłodna`
6. KUP_JEDZENIE_DZIECIOM *costs* 2 *with effect* ¬ `DzieciGłodne`

gdzie jako:

1. zbiór fluentów przyjmujemy: $F = \{$ MurarzSmutny, MurarzGłodny, ŻonaGłodna, DzieciGłodne $\}$
2. zbiór akcji przyjmujemy: $Ac = \{$ KUP_JEDZENIE_MURARZOWI, KUP_JEDZENIE_ŻONIE, KUP_JEDZENIE_DZIECIOM $\}$
3. zbiorem możliwych stanów zasobu jest $P = \mathbb{N}$
4. zbiory $C_{\text{KUP\_JEDZENIE\_MURARZOWI}} = C_{\text{KUP\_JEDZENIE\_ZONIE}} = C_{\text{KUP\_JEDZENIE\_DZIECIOM}} = \{2\}$

Łatwo zauważyć, że wszystkich stanów jakie uda się utworzyć ze zbioru fluentów jest 2^4, przy czym w naszym modelu dopuszczalnych jest tylko 2^3 możliwości, co jest spowodowane tym, że w zapisie naszego problemu występuje: *always* `MurarzSmutny`.
Zastanówmy się nad wyglądem stanów początkowych. Otóż jest ich 2^2. Są to stany:

1. σ_1 =(`MurarzSmutny, MurarzGłodny, ŻonaGłodna, DzieciGłodne`)
2. σ_2 =(`MurarzSmutny, MurarzGłodny, ¬ŻonaGłodna, DzieciGłodne`)
3. σ_3 =(`MurarzSmutny, MurarzGłodny, ŻonaGłodna, ¬DzieciGłodne`)
4. σ_4 =(`MurarzSmutny, MurarzGłodny, ¬ŻonaGłodna, ¬DzieciGłodne`)

wynika to z faktu, że w naszym programie występują wyrażenia: *initially* `MurarzGłodny` i *always* `MurarzSmutny`.
Resztę stanów (stany nie będące stanami początkowymi i należące do zbioru wszystkich stanów dopuszczalnych Σ) oznaczmy jako:

1. σ_5 =(`MurarzSmutny, ¬MurarzGłodny, ŻonaGłodna, DzieciGłodne`)
2. σ_6 =(`MurarzSmutny, ¬MurarzGłodny, ¬ŻonaGłodna, DzieciGłodne`)
3. σ_7 =(`MurarzSmutny, ¬MurarzGłodny, ŻonaGłodna, ¬DzieciGłodne`)
4. σ_8 =(`MurarzSmutny, ¬MurarzGłodny, ¬ŻonaGłodna, ¬DzieciGłodne`)

Powstaje pytanie: dlaczego nie rozpartujemy stanów w których fluent MurarzSmutny jest zaprzeczony? Ponieważ stany te są niedopuszczalne (nie spełniają warunku *always* `MurarzSmutny`), który jest sprawdzany już na początku działania programu.

Zastanówmy się jakie stany możemy otrzymać wykonując w stanie oznaczonym przez nas jako σ_3 akcję KUP_JEDZENIE_DZIECIOM i akcję KUP_JEDZENIE_MURARZOWI. Zacznijmy od: KUP_JEDZENIE_MURARZOWI. Robimy to w następujący sposób:

- Liczymy:
  Res_0(KUP_JEDZENIE_MURARZOWI,σ_3,5) = $\{(\sigma_5, 3), (\sigma_6, 3), (\sigma_7, 3), (\sigma_8, 3)\}$
- Dla wszystkich stanów z Res_0 liczymy wartości zbiorów New otrzymujemy:
  - New(KUP_JEDZENIE_MURARZOWI, σ_3, σ_5) = {`MurarzGłodny`, `DzieciGłodne`}

- $New(\text{KUP\_JEDZENIE\_MURARZOWI}, \sigma_3, \sigma_6) = \{\texttt{MurarzGłodny}, \texttt{ŻonaGłodna}, \texttt{DzieciGłodne}\}$
- $New(\text{KUP\_JEDZENIE\_MURARZOWI}, \sigma_3, \sigma_7) = \{\texttt{MurarzGłodny}\}$
- $New(\text{KUP\_JEDZENIE\_MURARZOWI}, \sigma_3, \sigma_8) = \{\texttt{MurarzGłodny}, \texttt{ŻonaGłodna}\}$

- Szukamy który ze zbiorów New jest minimalny ze względu na zawieranie
- Liczymy $Res_{RW}(\text{KUP\_JEDZENIE\_MURARZOWI},\sigma_3,5) = \{(\sigma_7, 3)\}$
- Liczymy $Res(\text{KUP\_JEDZENIE\_MURARZOWI},\sigma_3,5) = \{(\sigma_7, 3)\}$

Stąd wnioskujemy, że po wykonaniu akcji KUP_JEDZENIE_MURARZOWI przy stanie zasobu 5 i ze stanu σ_3 osiągamy stan σ_7
W przypadku akcji KUP_JEDZENIE_DZIECIOM otrzymujemy:

- Liczymy:

  $Res_0(\text{KUP\_JEDZENIE\_DZIECIOM},\sigma_3,5) = \{(\sigma_3, 3), (\sigma_4, 3), (\sigma_7, 3), (\sigma_8, 3)\}$

- Dla wszystkich stanów z Res_0 liczymy wartości zbiorów New otrzymujemy:
  - $New(\text{KUP\_JEDZENIE\_DZIECIOM}, \sigma_3, \sigma_3) = \emptyset$
  - $New(\text{KUP\_JEDZENIE\_DZIECIOM}, \sigma_3, \sigma_4) = \{\texttt{ŻonaGłodna}\}$
  - $New(\text{KUP\_JEDZENIE\_DZIECIOM}, \sigma_3, \sigma_7) = \{\texttt{MurarzGłodny}\}$
  - $New(\text{KUP\_JEDZENIE\_DZIECIOM}, \sigma_3, \sigma_8) = \{\texttt{MurarzGłodny}, \texttt{ŻonaGłodna}\}$
- Szukamy który ze zbiorów New jest minimalny ze względu na zawieranie
- Liczymy $Res_{RW}(\text{KUP\_JEDZENIE\_DZIECIOM},\sigma_3,5) = \{(\sigma_3, 3)\}$
- Liczymy $Res(\text{KUP\_JEDZENIE\_DZIECIOM},\sigma_3,5) = \{(\sigma_3, 3)\}$

Stąd jeśli wykonamy akcję KUP_JEDZENIE_DZIECIOM na stanie σ_3 przy wartości portfela 5 to nie opuścimy tego stanu.

Podane wyżej rozważania można przeprowadzić dla dowolnego stanu i dowolnej akcji, otrzymując jakie stany zostaną przyjęte po zastosowaniu akcji w jakimś stanie przy ustalonym stanie zasobu. Wyniki takiego postępowania

można by przedstawić na przykład na grafie o 2^3 wierzchołkach i krawędziach indeksowanych nazwami akcjami (i ewentualnie ceną tych akcji).

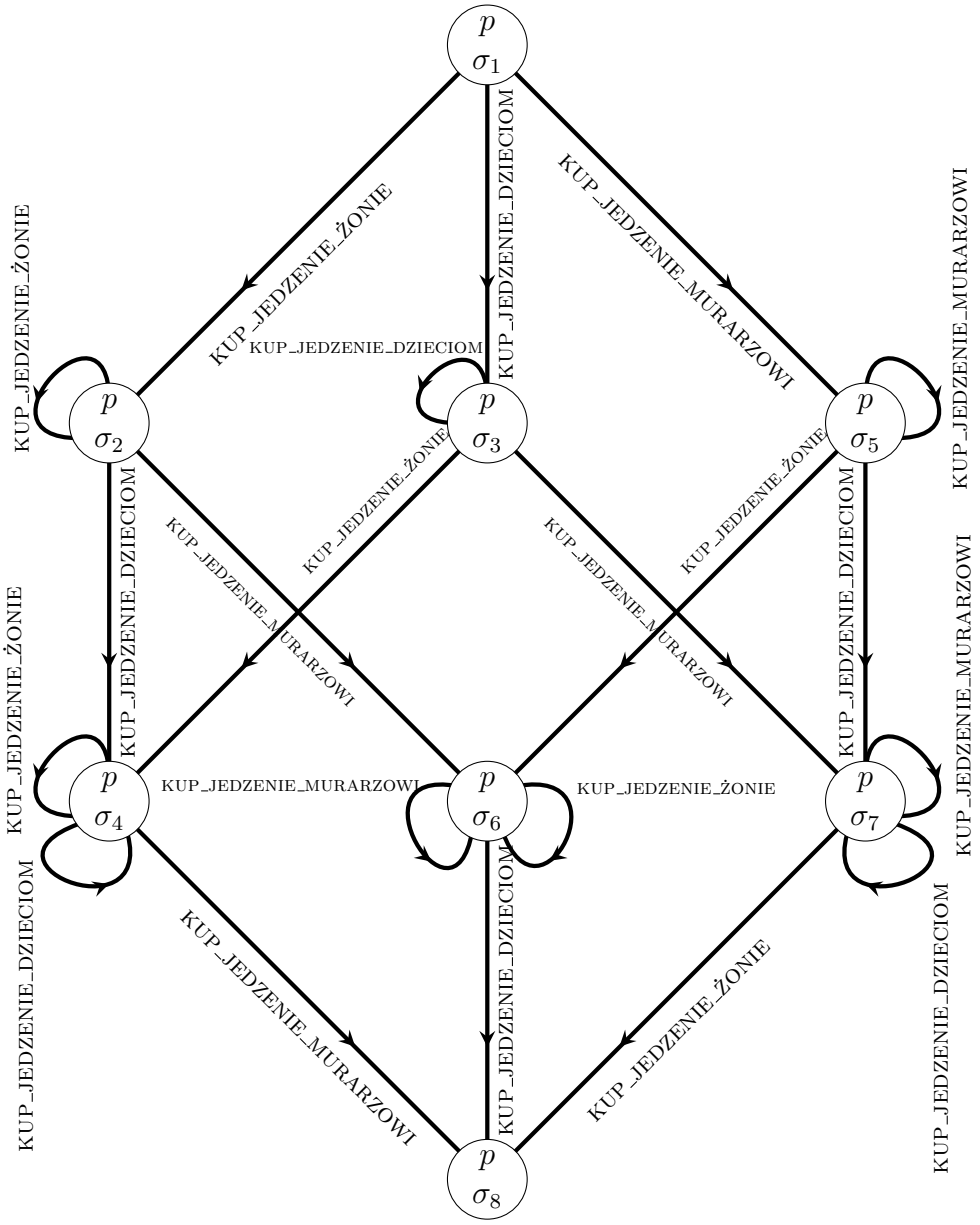
Zastanówmy się teraz nad rozwiązaniem głównego problemu naszego zadania, czyli tego czy murarz będzie w stanie wyżywić swoją rodzinę w czasie zimy. Zauważmy, że aby odpowiedzieć na to pytanie twierdząco musimy stwierdzić czy z dowolnego stanu początkowego (stany σ_1 – σ_4) jesteśmy w stanie wykonać pewien ciąg akcji, który prowadzi nas do osiągnięcia celu którym jest stan spełniający warunek:

$$\neg\texttt{MurarzGłodny}\wedge\neg\texttt{ŻonaGłodna}\wedge\neg\texttt{DzieciGłodne}.$$

Nietrudno zauważyć, że jedynym stanem spełniającym ten warunek jest stan σ_8. Zwróćmy też uwagę, że jedynym możliwym sposobem na otrzymanie σ_8 ze stanów σ_1 – σ_4 jest wykonanie w dowolnej kolejności akcji:

- $\{$KUP_JEDZENIE_MURARZOWI, KUP_JEDZENIE_ŻONIE, KUP_JEDZENIE_DZIECIOM$\}$ w przypadku σ_1
- $\{$KUP_JEDZENIE_MURARZOWI, KUP_JEDZENIE_DZIECIOM$\}$ w przypadku σ_2
- $\{$KUP_JEDZENIE_MURARZOWI, KUP_JEDZENIE_ŻONIE$\}$ w przypadku σ_3
- $\{$KUP_JEDZENIE_MURARZOWI$\}$ w przypadku σ_4

Każda z tych akcji kosztuje 2, przy początkowym stanie zasobu (złotych monet) 5, oznacza to że jesteśmy w stanie wykonać najwyżej dwie akcje kupna jedzenia. Oznacza to, że ze stanu σ_1 nie jesteśmy w stanie osiągnąć celu, w reszcie stanów początkowych jest to możliwe. Oznacza to, że nie możemy przyjąć, że rodzina murarza była niegłodna w czasie zimy.



Rysunek 1: Schemat pierwszego przykładu. Każda krawędź indeksowana jest nazwą akcji, przy czym istnieje tylko, jeśli zachodzi warunek: $2 \leq p$. Po wykonaniu każdej akcji aktualizujemy p w następujący sposób: $p = p - 2$.

6.2 Przykład 2

Kot Mruczek prowadzi bardzo systematyczny tryb życia. Głównie śpi, ale w czasie krótkich (120–minutowych) przerw między spaniem może pójść na myszy (tylko jeśli jest głodny, a po wykonaniu tej akcji przestaje być głodny). Jeśli Mruczek nie jest znudzony, to łapie mysz w ciągu 30 minut, w przeciwnym razie udaje mu się to dopiero po 90 minutach. Mruczek może również patrzeć w okno przez 10 minut z różnym skutkiem – czasami będzie po tym znudzony, a czasami nie. Mruczek może też wejść do pudełka na 30 minut (co sprawi, że nie będzie znudzony). Wiemy, że Mruczek budzi się znudzony i głodny. Czy jest szansa na to, że Mruczek tak zagospodaruje przerwę między spaniem, żeby na koniec nie być głodnym i znudzonym?

Dylemat Mruczka w naszym języku możemy zapisać tak, jak poniżej:

1. **initially** `głodny`, `znudzony`
2. **initially** 120 **value**
3. IDŹ_NA_MYSZY *causes* ¬`głodny`
4. IDŹ_NA_MYSZY *costs* 30 if ¬`znudzony`
5. IDŹ_NA_MYSZY *costs* 90 if `znudzony`
6. *impossible* IDŹ_NA_MYSZY if ¬`głodny`
7. PATRZ_PRZEZ_OKNO *releases* `znudzony`
8. PATRZ_PRZEZ_OKNO *costs* 10
9. WEJDŹ_DO_PUDEŁKA *costs* 30 *with* $effect$ ¬`znudzony`

gdzie przyjmujemy:

1. zbiór fluentów: $F = \{$`głodny`, `znudzony`$\}$
2. zbiór akcji: $Ac = \{$IDŹ_NA_MYSZY, PATRZ_PRZEZ_OKNO, WEJDŹ_DO_PUDEŁKA$\}$
3. zbiór możliwych stanów zasobu: $P = \mathbb{N}$
4. zbiory cen: $C_{\text{IDZ\_NA\_MYSZY}} = \{30, 90\}$, $C_{\text{PATRZ\_PRZEZ\_OKNO}} = \{10\}$, $C_{\text{WEJDZ\_DO\_PUDELKA}} = \{30\}$

W naszym modelu dopuszczalnych jest 2^2 stanów. Stan początkowy, oznaczany jako σ_0 jest następujący:

$$\sigma_0 = (\texttt{głodny}, \texttt{znudzony}).$$

Pozostałe stany wyglądają jak poniżej:

$$\sigma_1 = (\neg\texttt{głodny}, \texttt{znudzony}),$$
$$\sigma_2 = (\texttt{głodny}, \neg\texttt{znudzony}),$$
$$\sigma_3 = (\neg\texttt{głodny}, \neg\texttt{znudzony}).$$

Zobaczmy teraz, jaki stan otrzymamy po wykonaniu akcji IDŹ_NA_MYSZY, jeśli Murczek jest aktualnie w stanie początkowym. Liczymy najpierw Res_0:

$$Res_0(\text{IDŹ\_NA\_MYSZY}, \sigma_0, 120) = \{(\sigma_1, 30), (\sigma_3, 30)\}.$$

Następnie szukamy minimalnego zbioru New (ze względu na zawieranie):

$$New(\text{IDŹ\_NA\_MYSZY}, \sigma_0, \sigma_1) = \{\texttt{głodny}\},$$
$$New(\text{IDŹ\_NA\_MYSZY}, \sigma_0, \sigma_3) = \{\texttt{głodny}, \texttt{znudzony}\}.$$

Ostatecznie otrzymujemy:

$$Res_{RW}(\text{IDŹ\_NA\_MYSZY}, \sigma_0, 120) = \{(\sigma_1, 30)\},$$
$$Res(\text{IDŹ\_NA\_MYSZY}, \sigma_0, 120) = \{(\sigma_1, 30)\}.$$

Zatem po wykonaniu akcji IDŹ_NA_MYSZY jesteśmy w stanie σ_1, a wartość naszego zasobu wynosi 30. Ze stanu σ_1 możemy już wykonać tylko akcje WEJDŹ_DO_PUDEŁKA i PATRZ_PRZEZ_OKNO, ponieważ akcja IDŹ_NA_MYSZY może być wykonana wyłącznie, jeśli kot jest głodny. Zastanówmy się zatem, w którym ze stanów znajdzie się Mruczek, jeśli wykona teraz akcję WEJDŹ_DO_PUDEŁKA.

$$Res_0(\text{WEJDŹ\_DO\_PUDEŁKA}, \sigma_1, 30) = \{(\sigma_2, 0), (\sigma_3, 0)\}.$$

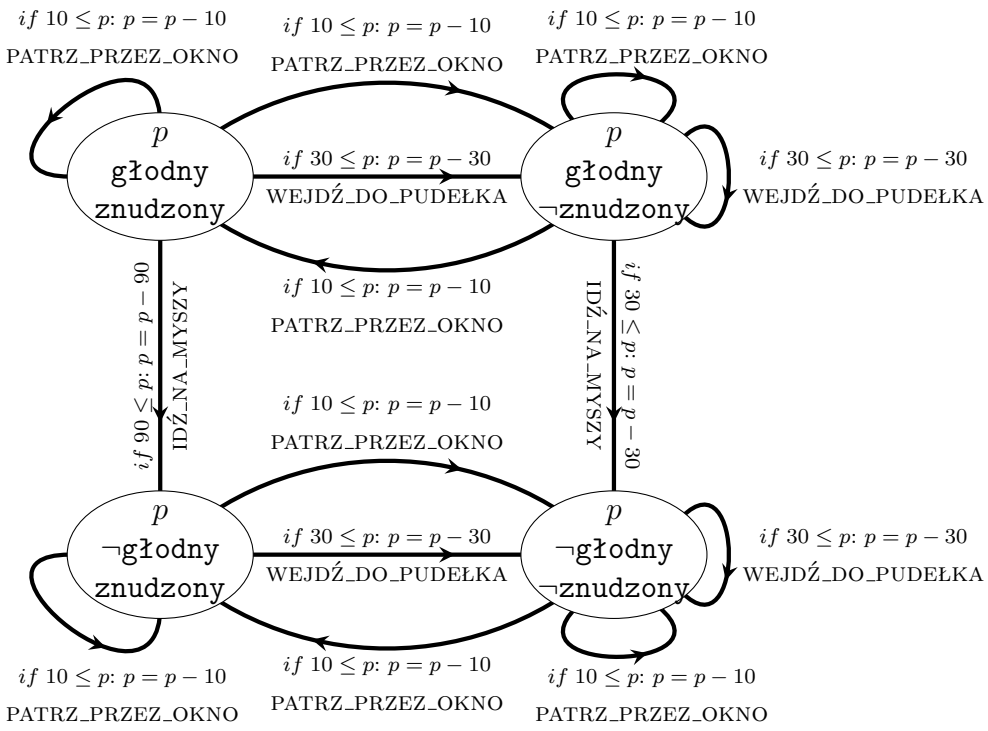
Szukamy minimalnego zbioru New (ze względu na zawieranie):

$$New(\text{WEJDŹ\_DO\_PUDEŁKA}, \sigma_1, \sigma_2) = \{\texttt{głodny}, \texttt{znudzony}\},$$
$$New(\text{WEJDŹ\_DO\_PUDEŁKA}, \sigma_1, \sigma_3) = \{\texttt{znudzony}\}.$$

Ostatecznie otrzymujemy:

$$Res_{RW}(\text{WEJDŹ\_DO\_PUDEŁKA}, \sigma_1, 30) = \{(\sigma_3, 0)\},$$
$$Res(\text{WEJDŹ\_DO\_PUDEŁKA}, \sigma_1, 30) = \{(\sigma_3, 0)\}.$$

Po tych dwóch akcjach stan zasobu został już całkowicie wyczerpany, ale Mruczek ostatecznie nie jest ani głodny, ani znudzony – co chcieliśmy osiągnąć.



Rysunek 2: Schemat drugiego przykładu.

6.3 Przykład 3

Przyjrzyjmy się następującej historii:

Zarządzamy pracą elektrowni. Zasoby oznaczają wkład pracy potrzebny do wykonania danej czynności. Wyłączanie zasilania i chłodzenia nie jest pracochłonne, dlatego nie wymaga zasobów. Włączenie zasilania lub chłodzenia, nie jest już tak łatwe i wymaga 1 zasobu. Jeśli elektrownia działa bez chłodzenia, to dochodzi do awarii. Chłodzenie nie może działać, ani być włączone jeśli zasilanie nie działa. Elektrownia ma prawie wszystkie najnowsze zabezpieczenia, dlatego nie można przerywać jej pracy bez włączonego układu chłodniczego. Włączenie lub wyłączenie elektrowni kosztuje 3 zasoby. Włączenie elektrowni wymaga aby zasilanie było włączone. Napawa elektrowni jest 'najdroższym' procesem, ponieważ kosztuje aż 10 zasobów. Naprawa odbywa się w przypadku odcięcia zasilania, co wcale nie oznacza, że elektrownia nie jest aktywna. Pytanie brzmi - czy mając 15 zasobów możemy zawsze usunąć awarię?

Podany problem można przedstawić w postaci następującego zagadnienia:

1. **initialy** 15 **value**

2. **initially** `Awaria`
3. **always** ¬ `Chłodzenie` ∧ `Aktywność` ⇒ `Awaria`
4. **always** ¬ `Zasilanie` ⇒ ¬ `Chłodzenie`
5. ZASILANIE-ON *costs* 1 *with effect* `Zasilanie`
6. ZASILANIE-OFF *causes* ¬ `Zasilanie`
7. CHŁODZENIE-ON *costs* 1 *if* `Zasilanie` *with effect* `Chłodzenie`
8. CHŁODZENIE-OFF *causes* ¬ `Chłodzenie`
9. AKTYWNOŚĆ-ON *costs* 3 *if* `Zasilanie` *with effect* `Aktywność`
10. AKTYWNOŚĆ-OFF *costs* 3 *if* `Chłodzenie` *with effect* ¬ `Aktywność`
11. NAPRAWA *costs* 10 *if* ¬ `Zasilanie` *with effect* ¬ `Awaria`

gdzie jako:

1. zbiór fluentów przyjmujemy: $F = \{$Chłodzenie, Zasilanie, Aktywność, Awaria$\}$
2. zbiór akcji przyjmujemy: $Ac = \{$ZASILANIE-ON, ZASILANIE-OFF, CHŁODZENIE-ON, CHŁODZENIE-OFF, AKTYWNOŚĆ-ON, AKTYWNOŚĆ-OFF, NAPRAWA$\}$
3. zbiorem możliwych stanów zasobu jest $P = \mathbb{N}$
4. zbiory cen:
   (a) $C_{ZASILANIE-OFF} = C_{CHLODZENIE-OFF} = \{0\}$,
   (b) $C_{ZASILANIE-ON} = C_{CHLODZENIE-ON} = \{1\}$,
   (c) $C_{AKTYWNOSC-ON} = C_{AKTYWNOSC-OFF} = \{3\}$,
   (d) $C_{NAPRAWA} = \{10\}$

Wszystkich stanów jakie można utworzyć ze zbioru fluentów jest 2^4, przy czym w naszym modelu dopuszczalnych jest tylko 10 możliwości, co jest spowodowane tym, że w zapisie naszego problemu występują ograniczenia:
always ¬ `Chłodzenie` ∧ `Aktywność` ⇒ `Awaria`
Co wyklucza stany:

1. σ_{11} =(¬`Chłodzenie`,¬`Zasilanie`, `Aktywność`,¬`Awaria`)

2. σ_{12} =(¬Chłodzenie, Zasilanie, Aktywność,¬Awaria)

always ¬ Zasilanie ⇒ ¬ Chłodzenie
Co wyklucza stany:

1. σ_{13} =(Chłodzenie,¬Zasilanie,Aktywność,Awaria)
2. σ_{14} =(Chłodzenie,¬Zasilanie,¬Aktywność,Awaria)
3. σ_{15} =(Chłodzenie,¬Zasilanie,Aktywność,¬Awaria)
4. σ_{16} =(Chłodzenie,¬Zasilanie,¬Aktywność,¬Awaria)

Zastanówmy się nad stanami początkowymi. Są to stany:

1. σ_1 =(¬Chłodzenie,¬Zasilanie,¬Aktywność,Awaria)
2. σ_2 =(¬Chłodzenie,¬Zasilanie,Aktywność,Awaria)
3. σ_3 =(¬Chłodzenie,Zasilanie,¬Aktywność,Awaria)
4. σ_4 =(¬Chłodzenie,Zasilanie,Aktywność,Awaria)
5. σ_5 =(Chłodzenie,Zasilanie,¬Aktywność,Awaria)
6. σ_6 =(Chłodzenie,Zasilanie,Aktywność,Awaria)

wynika to z faktu, że w naszym programie występuje wyrażenie: **initially** Awaria.
Resztę stanów (stany nie będące stanami początkowymi i należące do zbioru wszystkich stanów dopuszczalnych Σ) oznaczmy jako:

1. σ_7 =(¬Chłodzenie,¬Zasilanie,¬Aktywność,¬Awaria)
2. σ_8 =(¬Chłodzenie,Zasilanie,¬Aktywność,¬Awaria)
3. σ_9 =(Chłodzenie,Zasilanie,¬Aktywność,¬Awaria)
4. σ_{10} =(Chłodzenie,Zasilanie,Aktywność,¬Awaria)

Naszym celem jest próba przeprowadzenia stanów od σ_1 do σ_6, czyli stanów początkowych, w stany od σ_7 do σ_{10} z wykorzystaniem maksymalnie 15 zasobów. Pokażemy, że wykonanie sekwencji akcji ZASILANIE-ON, CHŁODZENIE-ON, AKTYWNOSC-OFF, ZASILANIE-OFF, NAPRAWA, których łączny koszt wynosi $C_{ZASILANIE-ON} + C_{CHLODZENIE-ON} + C_{AKTYWNOSC-OFF} + C_{ZASILANIE-OFF} + C_{NAPRWA} = 1+1+3+0+10 = 15$ zawsze zaprowadzi nas, bez naruszania ograniczeń, do stanu, w którym fluent Awaria jest niezaprzeczony. Zacznijmy od wykonania na stanach od σ_1 do σ_6 akcji ZASILANIE-ON. Po obliczeniach otrzymujemy:

- Res(ZASILANIE-ON,σ_1,15) = $\{(\sigma_3, 14)\}$
- Res(ZASILANIE-ON,σ_2,15) = $\{(\sigma_4, 14)\}$
- Res(ZASILANIE-ON,σ_3,15) = $\{(\sigma_3, 14)\}$
- Res(ZASILANIE-ON,σ_4,15) = $\{(\sigma_4, 14)\}$
- Res(ZASILANIE-ON,σ_5,15) = $\{(\sigma_5, 14)\}$
- Res(ZASILANIE-ON,σ_6,15) = $\{(\sigma_6, 14)\}$

Widzimy, że dla stanów od σ_3 do σ_6 wykonanie akcji ZASILANIE-ON nie jest konieczne, ponieważ stany nie zmienią się. Jeśli pokażemy, że ze stanu σ_x możemy dojść do stanu σ_y z wykorzystaniem k zasobów, to możemy również przejść ze stanu σ_x do stanu σ_y posiadając $k+i$ zasobów, gdzie $i \in \{1, 2, 3, ...\}$. Oczywiście przy założeniu, że wykonanie akcji nie może wymagać ujemnej liczby zasobów (co zakładamy). W dalszej części rozumowania nie musimy więc pokazywać, że stany σ_3 oraz σ_4 można przeprowadzić w stany od σ_7 do σ_{10} posiadając 15 zasobów, jeśli pokażemy, że można to zrobić z wykorzystaniem 14 zasobów. Wykonajmy teraz akcję CHŁODZENIE-ON.

- Res(CHŁODZENIE-ON,σ_3,14) = $\{(\sigma_5, 13)\}$
- Res(CHŁODZENIE-ON,σ_4,14) = $\{(\sigma_6, 13)\}$
- Res(CHŁODZENIE-ON,σ_5,15) = $\{(\sigma_5, 14)\}$
- Res(CHŁODZENIE-ON,σ_6,15) = $\{(\sigma_6, 14)\}$

Widać, że dla stanów od σ_5 oraz σ_6 wykonanie akcji CHŁODZENIE-ON nie jest konieczne. Wykonując akcję CHŁODZENIE-ON przeprowadziliśmy stany σ_3 i σ_4 w stany σ_5 i σ_6. W dalszych rozważaniach będziemy się więc zajmować już tylko stanami σ_5 oraz σ_6. Wykonajmy akcję AKTYWNOSC-OFF.

- Res(AKTYWNOSC-OFF,σ_5,13) = $\{(\sigma_5, 10)\}$
- Res(AKTYWNOSC-OFF,σ_6,13) = $\{(\sigma_5, 11)\}$

Doszliśmy do sytuacji, w której rozpatrujemy już tylko jeden stan σ_5. Jeśli pokażemy, że możemy go przeprowadzić w któryś ze stanów od σ_7 do σ_{10} posiadając 10 zasobów to udowodnimy, że wszystkie stany początkowe można przeprowadzić w stan, w którym fluent Awaria jest niezaprzeczony, z wykorzystaniem 15 zasobów. Wykonajmy akcję ZASILANIE-OFF. Zauważmy, że spowoduje ona zmianę nie tylko fluentu Zasilanie ale również Chłodzenie. Stanie się tak z powodu ograniczenia **always** $\neg$ `Zasilanie` $\Rightarrow \neg$ `Chłodzenie`.

- Res(ZASILANIE-OFF,σ_5,10) = $\{(\sigma_1, 10)\}$

Ostatnią wykonaną akcją będzie NAPRAWA.

- Res(ZASILANIE-OFF,σ_1,10) = $\{(\sigma_7, 0)\}$

Co kończy dowód. Pokazaliśmy, że z wykorzystaniem 15 zasobów rzeczywiście można z każdego stanu początkowego dojść do stanu od σ_7 do σ_{10}. Wyniki postępowania możnaby przedstawić na grafie o 10 wierzchołkach i krawędziach indeksowanych nazwami akcjami oraz ich ceną. Przejścia, które znaleźliśmy nie zawsze są optymalne (np. ze stanu σ_1 można było od razu przejść do stanu σ_7, zamiast do σ_3), ale spełniają wszytki postanowione ograniczenia. Odpowiedź na postawione w zadaniu pytanie brzmi więc, że mając 15 zasobów możemy zawsze usunąć awarię.

6.4 Przykład 4

Marysia mieszka w Bartoszycach. W przyszłą sobotę wybiera się do swojej szkoły na bal karnawałowy (również w Bartoszycach). Jednak zanim wybierze się na ten bal, zauważyła, że nie ma się w co ubrać. Marysia posiada 100 zł. W najbliższym sklepie "Bolek", Marysia może kupić sukienkę za 60 zł oraz buty za 50 zł. Może również wybrać się do sąsiedniego miasta Olsztyn, do którego jeździ autobus. Bilet do Olsztyna kosztuje 10 zł. Wieczorem Marysia powinna wrócić tym samym autobusem do domu (koszt również wynosi 10 zł). W znajdującym się w Olsztynie sklepie "Tola" może kupić sukienkę za 30 zł, oraz buty za 80 zł. Czy Marysi starczy pieniędzy na potrzebne zakupy?

Problem można przedstawić do następującego zagadnienia:

1. **initially** 100 **value**
2. **initially** ¬ `PosiadaSukienkę`
3. **initially** ¬ `PosiadaButy`
4. **initially** `WBartoszycach`
5. AUTOBUSOLSZTYN *costs* 10 *if* `WBartoszycach` *with effect* ¬ `WBartoszycach`
6. AUTOBUSBARTOSZYCE *costs* 10 *if* ¬ `WBartoszycach` *with effect* `WBartoszycach`
7. KUPBUTYBOLEK *costs* 50 *if* `WBartoszycach` *with effect* `PosiadaButy`
8. KUPBUTYTOLA *costs* 80 *if* ¬ `WBartoszycach` *with effect* `PosiadaButy`

9. KUPSUKIENKĘBOLEK *costs* 60 *if* `WBartoszycach` *with effect* `PosiadaSukienkę`
10. KUPSUKIENKĘTOLA *costs* 30 *if* ¬ `WBartoszycach` *with effect* `PosiadaSukienkę`

gdzie:

1. zbiór fluentów przyjmujemy: $F = \{$PosiadaSukienkę, PosiadaButy, WBartoszycach$\}$
2. zbiór akcji przyjmujemy: $Ac = \{$AUTOBUSOLSZTYN, AUTOBUSBARTOSZYCE, KUPBUTYBOLEK, KUPBUTYTOLA, KUPSUKIENKĘBOLEK, KUPSUKIENKĘTOLA$\}$
3. zbiorem możliwych stanów zasobu jest $P = \mathbb{N}$
4. zbiory cen:
   (a) $C_{AUTOBUSOLSZTYN} = C_{AUTOBUSBARTOSZYCE} = \{10\}$,
   (b) $C_{KUPSUKIENKETOLA} = \{30\}$,
   (c) $C_{KUPBUTYBOLEK} = \{50\}$,
   (d) $C_{KUPSUKIENKEBOLEK} = \{60\}$,
   (e) $C_{KUPBUTYTOLA} = \{80\}$

Wszystkich stanów jakie można utworzyć ze zbioru fluentów jest 2^3 Istnieje tylko jeden stan początkowy:

- $\sigma_1 =$(¬`PosiadaSukienkę`,¬`PosiadaButy`,`WBartoszycach`)

Resztę stanów (stany nie będące stanami początkowymi i należące do zbioru wszystkich stanów dopuszczalnych Σ) oznaczmy jako:

- $\sigma_2 =$(`PosiadaSukienkę`,`PosiadaButy`,`WBartoszycach`)
- $\sigma_3 =$(`PosiadaSukienkę`,`PosiadaButy`,¬`WBartoszycach`)
- $\sigma_4 =$(`PosiadaSukienkę`,¬`PosiadaButy`,`WBartoszycach`)
- $\sigma_5 =$(`PosiadaSukienkę`,¬`PosiadaButy`,¬`WBartoszycach`)
- $\sigma_6 =$(¬`PosiadaSukienkę`,`PosiadaButy`,`WBartoszycach`)
- $\sigma_7 =$(¬`PosiadaSukienkę`,`PosiadaButy`,¬`WBartoszycach`)

- σ_8 =(¬PosiadaSukienkę,¬PosiadaButy,¬WBartoszycach)

Naszym celem jest próba przeprowadzenia stanu początkowego σ_1 do stanu σ_2. Pokażemy, że wykonanie sekwencji akcji AUTOBUSOLSZTYN, KUPSUKIENKĘTOLA, AUTOBUSBARTOSZYCE, KUPBUTYBOLEK, których łączny koszt wynosi $C_{AUTOBUSOLSZTYN} + C_{KUPSUKIENKETOLA} + C_{AUTOBUSBARTOSZYCE} + C_{KUPBUTYBOLEK} = 10 + 30 + 10 + 50 = 100$, doprowadzi to takiego przejścia. Wykonajmy kolejne obliczenia:

1. Res(AUTOBUSOLSZTYN,σ_1,100) = $\{(\sigma_8, 90)\}$
2. Res(KUPSUKIENKĘTOLA,σ_8,90) = $\{(\sigma_5, 60)\}$
3. Res(AUTOBUSBARTOSZYCE,σ_5,60) = $\{(\sigma_4, 50)\}$
4. Res(KUPBUTYBOLEK,σ_4,50) = $\{(\sigma_2, 0)\}$

Uwaga - nie jest to jedyna sekwencja akcji o takim skutku. Wykonanie akcji KUPBUTYBOLEK, AUTOBUSOLSZTYN, KUPSUKIENKĘTOLA, AUTOBUSBARTOSZYCE również stanowi poprawne rozwiązanie problemu. Kolejne przejścia wyglądają wtedy następująco:

1. Res(KUPBUTYBOLEK,σ_1,100) = $\{(\sigma_6, 50)\}$
2. Res(AUTOBUSOLSZTYN,σ_6,50) = $\{(\sigma_7, 40)\}$
3. Res(KUPSUKIENKĘTOLA,σ_7,40) = $\{(\sigma_3, 10)\}$
4. Res(AUTOBUSBARTOSZYCE,σ_3,10) = $\{(\sigma_2, 0)\}$

W ten sposób pokazaliśmy, że z wykorzystaniem 100zł rzeczywiście można ze stanu początkowego σ_1 dojść do stanu od σ_2. Wyniki postępowania możnaby przedstawić na grafie o wierzchołkach indeksowanych stanami i krawędziach indeksowanych nazwami akcjami oraz ich ceną. Odpowiedź na postawione w zadaniu pytanie brzmi więc, że Marysi starczy pieniędzy na wydatki związane z balem.

6.5 Przykład 5

Robert jest nastolatkiem, który przepada za grą w piłkę i często spotyka się z gronem swoich znajomych. Pewnego dnia obudził się wypoczęty (30 jednostek) i snuł swoje plany odnośnie kolejnych meczów i spotkań towarzyskich. Każdy mecz wymaga, by Robert był sprawny, jest bardzo męczący (10 jednostek) i powoduje liczne kontuzje, tak że tylko maść Altacet jest w stanie przywrócić sprawność Roberta po meczu. Zastosowanie maści Altacet

to drobnostka, więc koszt aplikowania Altacetu to 0 jednostek. Dodatkowo, humor Roberta zmienia się zależnie od przebiegu meczu: raz Robert jest zadowolony po meczu, innym razem nie.

Spotkania towarzyskie natomiast nie kosztują Roberta wiele wysiłku, szczególnie, gdy jest zadowolony - wówczas koszt to 1 jednostka. Gdy Robert nie jest zadowolony, uczestniczenie w spotkaniu jest dla niego nieco męczące - 5 jednostek. Co więcej, spotkania towarzyskie mają różny efekt, w zależności od tego, w jakim humorze jest Robert. Gdy jest zadowolony, budzi wielką sympatię i po spotkaniu cieszy się popularnością - jest lubiany. Natomiast gdy humor mu nie dopisuje, zwykle zachowuje się kapryśnie i docina innym, tak że po spotkaniu nie jest lubiany. Jaki jest minimalny koszt akcji, które sprawią że Robert będzie lubiany jeśli obudził się niezadowolony i nie lubiany?

Ten problem można przedstawić następująco w naszym języku:

1. **initially** ¬ `Zadowolony`
2. **initially** ¬ `Lubiany`
3. **initially** 30 **value**
4. MECZ *costs 10* **if** `Sprawny` *with effect*¬ `Sprawny`
5. MECZ **releases** `Zadowolony`
6. ALTACET *costs 0 with effect* `Sprawny`
7. SPOTKANIE *costs 1* **if** `Zadowolony` *with effect* `Lubiany`
8. SPOTKANIE *costs 5* **if** ¬ `Zadowolony` *with effect* ¬ `Lubiany`

gdzie jako:

1. zbiór fluentów przyjmujemy: $F = \{$ Zadowolony, Sprawny, Lubiany $\}$
2. zbiór akcji przyjmujemy: $Ac = \{$ MECZ, ALTACET, SPOTKANIE $\}$
3. zbiorem możliwych stanów zasobu jest $P = \mathbb{N}$
4. zbiory cen:
   (a) $C_{ALTACET} = \{0\}$,
   (b) $C_{SPOTKANIE} = \{1, 5\}$,
   (c) $C_{MECZ} = \{10\}$,

Wszystkich możliwych stanów fluentów jest 2^3. Stanów początkowych jest 2^1. Są to stany:

1. σ_1 =(`Sprawny, ¬Zadowolony, ¬Lubiany`)
2. σ_2 =(`¬Sprawny, ¬Zadowolony, ¬Lubiany`)

wynika to z faktu, że w naszym programie występują wyrażenia:

1. **initially** ¬ `Zadowolony`
2. **initially** ¬ `Lubiany`

Stany należące do zbioru wszystkich stanów dopuszczalnych i nie będące stanami początkowymi oznaczmy jako:

1. σ_3 =(`Sprawny, Zadowolony, Lubiany`)
2. σ_4 =(`Sprawny, Zadowolony, ¬Lubiany`)
3. σ_5 =(`Sprawny, ¬Zadowolony, Lubiany`)
4. σ_6 =(`¬Sprawny, Zadowolony, Lubiany`)
5. σ_7 =(`¬Sprawny, Zadowolony, ¬Lubiany`)
6. σ_8 =(`¬Sprawny, ¬Zadowolony, Lubiany`)

Naszym celem jest prowadzenie stanów σ_1 i σ_2, czyli stanów początkowych, w stany σ_3, σ_5, σ_6 lub σ_8 , które spełniają warunek, że fluent Lubiany jest niezaprzeczony z wykorzystaniem minimalnej ilości zasobów. Przeanalizujmy wszystkie możliwe akcje, czyli MECZ, ALTACET oraz SPOTKANIE dla stanów σ_1 i σ_2. Załóżmy, że początkowa ilość jednostek wynosi x. Chcemy znaleźć taki ciąg akcji (osobny dla σ_1 i osobny dla σ_2), który doprowadzi do stanu w którym Robert jest Lubiany, a wartość x zmniejszy się o najmniejszą liczbę. Dla stanu σ_1:

- Res(MECZ,σ_1,x) = $\{(\sigma_2, x-10), (\sigma_7, x-10)\}$
- Res(ALTACET,σ_1,x) = $\{(\sigma_1, x)\}$
- Res(SPOTKANIE,σ_1,x) = $\{(\sigma_1, x-5)\}$

Dla stanu σ_2:

- w stanie σ_2 nie można wykonać akcji MECZ, ponieważ fluent Sprawny jest zaprzeczony.

- Res(ALTACET,σ_2,x) = $\{(\sigma_1, x)\}$
- Res(SPOTKANIE,σ_2,x) = $\{(\sigma_2, x - 5)\}$

Ze stanu σ_2 możemy przejść do stanów σ_1 oraz σ_2. Z kolei, ze stanu σ_1 możemy przejść do stanów σ_1, σ_2 oraz σ_7. Może się tak jednak zdarzyć, że ze stanu σ_1 będziemy przechodzili wyłącznie w stany σ_1 oraz σ_2. W pesymistycznym przypadku, nigdy nie wyjdziemy poza stany σ_1 oraz σ_2. Prowadzi to do konkluzji, że nie istnieje ciąg akcji, który zagwarantuje, osiągnięcie któregoś ze stanów w którym fluent Lubiany jest niezaprzeczony (stany σ_3, σ_5, σ_6, σ_8). Skoro nie istnieje taki ciąg akcji, to nie można podać jego minimalnego kosztu.