Wnioskowanie o działaniach z wykorzystaniem semantyki Dijkstry dla języków programowania – SDPL

Ewa Madalińska

na podstawie pracy:

- W. Łukaszewicz, E. Madalińska-Bugaj. Reasoning about Action and Change Using Dijkstra's Semantics for Programming Languages: Preliminary Report. *Proceedings of 14th International Joint Conference on Artificial Intelligence*, 1995, 1950-1955.

Idea:

1. Działania mają charakter imperatywny, należy więc do opisu działaniu użyć formalizmu o charakterze imperatywnym – język programowania;
2. Wnioskowanie wymaga logiki;
3. Należy zastosować mechanizm, który połączy te dwa formalizmy.

Rozwiązanie:

1. Jako bazę logiczną przyjmujemy logikę zdań. Stałe zdaniowe występujące w języku logiki będziemy nazywać fluentami.

2. Język opisu działań – język programowania Dijkstry.

3. Semantykę działań będziemy wyrażać za pomocą transformatorów formuł Dijkstry dla instrukcji reprezentującej działanie.

Język definiowania działań SDPL i semantyka Dijkstry

Założenia: tylko jeden typ zmiennych – zmienne logiczne (Boolean).

V – zbiór zmiennych logicznych.

Stan nad zbiorem V – funkcja σ określona na zbiorze V o wartościach w zbiorze $\{0, 1\}$

$$\sigma : V \to \{0, 1\}$$

Język asercji $\mathcal{L}(V)$ nad zbiorem zmiennych logicznych V – zbiór formuł skonstruowanych w zwykły sposób z elementów zbioru V, spójników logicznych $(\neg, \Rightarrow, \wedge, \vee, \equiv)$ i stałych logicznych T (prawda) i F (fałsz).

Terminologia

β – formuła,

$\alpha_1, \ldots, \alpha_n$ – wyrażenia boolowskie,

$x_1, \ldots, x_n$ – zmienne

$\beta[x_1 \leftarrow \alpha_1, \ldots, x_n \leftarrow \alpha_n]$ oznacza formułę otrzymaną z β w wyniku jednoczesnego zastąpienia wszystkich wolnych wystąpień $x_1, \ldots, x_n$ odpowiednio przez $\alpha_1, \ldots, \alpha_n$.

Kwantyfikatory $(\forall, \exists)^*$:

$$\forall x.\alpha \stackrel{\text{def}}{=} \alpha[x \leftarrow T] \wedge \alpha[x \leftarrow F],$$

$$\exists x.\alpha \stackrel{\text{def}}{=} \alpha[x \leftarrow T] \vee \alpha[x \leftarrow F].$$

*Zauważmy, że kwantyfikatory mogą być zastosowane jedynie do zmiennych logicznych.

Wartość formuły $\alpha \in \mathcal{L}(\mathcal{V})$ *w stanie* σ, $\| \alpha \|_\sigma$ jest to element zbioru $\{0, 1\}$ wyznaczony według reguł klasycznej logiki zdań.

Stan σ *spełnia* formułę α wtw $\| \alpha \|_\sigma = 1$. Jeżeli σ spełnia α, to σ nazywamy *modelem* α. Mówiąc, że σ spełnia α, będziemy zawsze zakładać, że σ określa interpretację dla wszystkich zmiennych boolowskich występujących w α.

Transformatory formuł:

Dla dowolnej instrukcji S i formuły α:

- $wp(S, \alpha)$ jest formułą, której modelami są dokładnie wszystkie stany takie, że wykonanie S rozpoczęte w którymkolwiek z nich zakończy się w stanie spełniającym warunek α.

- $wlp(S, \alpha)$ jest formułą, której modelami są dokładnie wszystkie stany takie, że jeśli wykonanie S rozpocznie się w którymkolwiek z nich i zakończy, to stan końcowy spełnia warunek α.

- $sp(S, \alpha)$ jest formułą, której modelami są dokładnie te stany, które są osiągalne w wyniku wykonania S, jeśli S się skończy, gdy S rozpocznie się w stanie spełniającym warunek α.

Semantyka instrukcji

Lista instrukcji: *skip*, *przypisanie*, *sekwencyjne złożenie*, *alternatywa* (niedeterministyczna).

1. **Instrukcja *skip*.** Jest to instrukcja "pusta", tzn. jej wykonanie nie zmienia stanu obliczeń.

   $$wlp(skip, \alpha) = sp(skip, \alpha) = \alpha.$$

2. **Instrukcja *przypisania*.** Jest postaci $x := e$, gdzie x jest zmienną boolowską, a e jest wyrażeniem boolowskim. W wyniku wykonania instrukcji wartość x zostaje zastąpiona przez wartość wyrażenia e.

$$wlp(x := e, \alpha) = \alpha[x \leftarrow e].$$
$$sp(x := e, \alpha) = \exists y.((x \equiv e[x \leftarrow y]) \wedge \alpha[x \leftarrow y]), \qquad (1)$$

gdzie y jest nową zmienną.

Jeśli zmienna x nie występuje w wyrażeniu e, równanie (1) można uprościć do postaci

$$sp(x := e, \alpha) = (x \equiv e) \wedge \exists x.\alpha$$

a biorąc pod uwagę, że zmienne są tylko typu boolowskiego, jest to równoważne

$$sp(x := e, \alpha) = (x \equiv e) \wedge (\alpha[x \leftarrow T] \vee \alpha[x \leftarrow F]). \qquad (2)$$

W przykładach często będziemy mieć do czynienia z przypisaniami $x := e$, gdzie e jest T lub F. Wtedy (2) można zastąpić przez

$$\begin{aligned} sp(x := T, \alpha) &= x \wedge (\alpha[x \leftarrow T] \vee \alpha[x \leftarrow F]) \\ sp(x := F, \alpha) &= \neg x \wedge (\alpha[x \leftarrow T] \vee \alpha[x \leftarrow F]) \end{aligned}$$

3. **Instrukcja *sekwencyjnego złożenia*.** Jest ona postaci $S_1; S_2$, gdzie S_1 i S_2 są dowolnymi instrukcjami. Wykonanie jej polega na wykonaniu najpierw S_1, a następnie S_2.

$$\begin{aligned} wlp(S_1; S_2, \alpha) &= wlp(S_1, wlp(S_2, \alpha)); \\ sp(S_1; S_2, \alpha) &= sp(S_2, sp(S_1, \alpha)). \end{aligned}$$

4. **Instrukcja *alternatywy*.** Jest ona postaci

$$\textbf{if} \quad B_1 \to S_1 \quad [\!] \quad B_2 \to S_2 \quad [\!] \quad \cdots \quad [\!] \quad B_n \to S_n \quad \textbf{fi} \qquad (3)$$

gdzie $B_1, \ldots, B_n$ są wyrażeniami boolowskimi, a $S_1, \ldots, S_n$ dowolnymi instrukcjami.

$B_1, \ldots, B_n$ nazywają się *dozorami*, a wyrażenia $B_i \to S_i$ *instrukcjami dozorowanymi*.

Instrukcję (3) będziemy oznaczać **if** $[\!] \, {}^{n}_{i=1} B_i \rightarrow S_i$ **fi** lub w skrócie jako IF.

Wykonanie instrukcji przebiega następująco: jeśli wszystkie dozory są fałszywe, wykonanie się kończy*, w przeciwnym przypadku spośród tych instrukcji dozorowanych $B_i \rightarrow S_i$, dla których B_i jest prawdziwe zostaje *losowo* wybrana jedna i S_i zostaje wykonane.†

$$wlp(IF, \alpha) \;=\; \bigwedge_{i=1}^{n} (B_i \Rightarrow wlp(S_i, \alpha)) \wedge ((\bigwedge_{i=1}^{n} \neg B_i) \Rightarrow \alpha)$$

$$sp(IF, \alpha) \;=\; (\bigvee_{i=1}^{n} sp(S_i, B_i \wedge \alpha)) \vee ((\bigwedge_{i=1}^{n} \neg B_i) \wedge \alpha).$$

*W wersji oryginalnej Dijkstry przypadek ten sprowadza się do instrukcji *abort*, której nie włączyliśmy do naszego repertuaru. Podana tutaj wersja IF jest specjalnym przypadkiem IF Dijkstry, tj.: **if** $B_1 \rightarrow S_1 \; [\!] \; \cdots \; [\!] \; B_n \rightarrow S_n \; [\!] \; \neg B_1 \wedge \ldots \wedge \neg B_n \rightarrow skip$ **fi**.

†Zauważmy, że jeśli więcej niż jeden dozór jest prawdziwy, wybór instrukcji dozorowanej jest niedeterministyczny.

Scenariusz

$$SC = \langle\alpha_0\rangle\ A_1; \langle\alpha_1\rangle\ A_2; \ldots; \langle\alpha_{n-1}\rangle\ A_n\ \langle\alpha_n\rangle$$

jest to pewne obliczenie należące do **klasy obliczeń**

$$[A_1; \ldots; A_n](\alpha_0, \ldots, \alpha_n). \tag{4}$$

Stany scenariusza

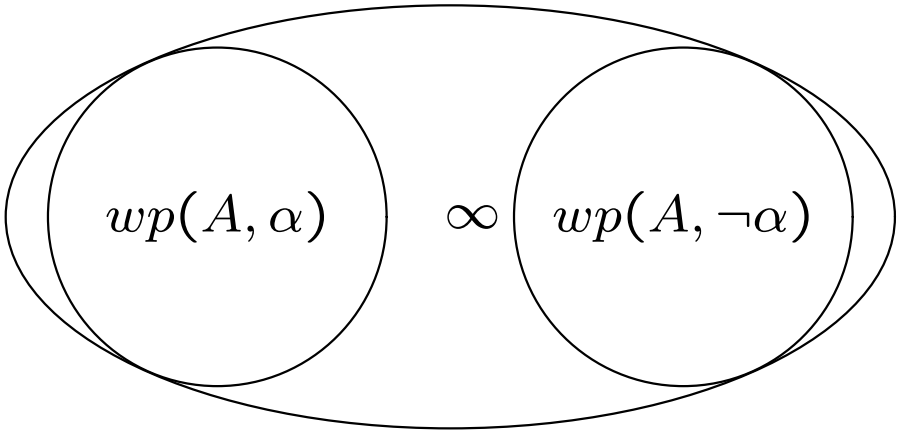
Niech $ST^{\mathcal{SC}}(i)$, $0 \leq i \leq n$, będzie zbiorem wszystkich stanów spełniających następujący warunek: dla każdego $\sigma \in ST^{\mathcal{SC}}(i)$, istnieje obliczenie $c \in [A_1; \ldots; A_n](\alpha_0, \ldots, \alpha_n)$ takie, że σ jest osiągany przez c po wykonaniu $A_1; \ldots; A_i$. Elementy $ST^{\mathcal{SC}}(i)$ będziemy nazywać *i-stanami* SC.

0-stany – stany początkowe

n-stany – stany końcowe.

Obliczanie i-stanów scenariusza

$\langle T \rangle A \langle \alpha \rangle$



$\mid wlp(A, \alpha) \mid = \mid wp(A, \alpha) \mid \cup \{\infty\}$, $\mid wlp(A, \neg\alpha) \mid = \mid wp(A, \neg\alpha) \mid \cup \{\infty\}$

0-stany$= \neg wlp(A, \neg\alpha)$

Niech WLP_i^{SC} i SP_i^{SC} będą formułami zdefiniowanymi indukcyjnie w sposób następujący:

$$
\begin{array}{l}
SP_0^{SC} = \alpha_0; \\
WLP_n^{SC} = \neg\alpha_n; \\
\quad \text{i dla } 0 < k \leq n \\
SP_k^{SC} = \alpha_k \wedge sp(A_k, SP_{k-1}^{SC}); \\
WLP_{n-k}^{SC} = \neg\alpha_{n-k} \vee wlp(A_{n-k+1}, WLP_{n-k+1}^{SC}).
\end{array}
$$

Twierdzenie 1
Niech $SC = \langle\alpha_0\rangle\ A_1; \langle\alpha_1\rangle\ A_2; \ldots; \langle\alpha_{n-1}\rangle\ A_n\ \langle\alpha_n\rangle$. Zbiór stanów początkowych scenariusza SC opisuje formuła

$$D_0^{SC} = \neg WLP_0^{SC} \wedge \alpha_0. \tag{5}$$

Formułę D_0^{SC} będziemy nazywać *opisem stanu początkowego* scenariusza SC. ∎

Twierdzenie 2
Niech $SC = \langle\alpha_0\rangle\, A_1; \langle\alpha_1\rangle\, A_2; \ldots; \langle\alpha_{n-1}\rangle\, A_n\, \langle\alpha_n\rangle$. Zbiór stanów końcowych scenariusza SC opisuje formuła

$$D_n^{SC} = SP_n^{SC}. \tag{6}$$

■

Twierdzenie 3
Zbiór $i-$stanów scenariusza SC opisuje formuła

$$D_i^{SC} = SP_i^{SC} \wedge \neg WLP_i^{SC}, \tag{7}$$

■

Przykłady

1. **Fluenty:**
   a (oznacza, że indyk żyje);
   l (strzelba jest naładowana)

2. **Działania:**
   $load$: $l := T$;
   $wait$: $skip$;
   $spin$: $\mathbf{if}\ T \rightarrow\ l := T\ \ [\!]\ \ T \rightarrow\ l := F\ \mathbf{fi}$;
   $shoot$: $\mathbf{if}\ l \rightarrow\ a := F; l := F\ [\!]\ \ \neg l \rightarrow\ skip\ \mathbf{fi}$;

Semantyka działań

- $wlp(load, \alpha) = \alpha[l \leftarrow T]$;
- $sp(load, \alpha) = l \wedge (\alpha[l \leftarrow T] \vee \alpha[l \leftarrow F])$;
- $wlp(wait, \alpha) = sp(wait, \alpha) = \alpha$;
- $wlp(spin, \alpha) = \alpha[l \leftarrow T] \wedge \alpha[l \leftarrow F]$;
- $sp(spin, \alpha) = \alpha[l \leftarrow T] \vee \alpha[l \leftarrow F]$;
- $wlp(shoot, \alpha) = (l \Rightarrow \alpha[a \leftarrow F, l \leftarrow F]) \wedge (\neg l \Rightarrow \alpha)$;
- $sp(shoot, \alpha) = (\neg a \wedge \neg l \wedge \alpha[a \leftarrow T, l \leftarrow T]) \vee (\neg a \wedge \neg l \wedge \alpha[a \leftarrow F, l \leftarrow T]) \vee (\neg l \wedge \alpha)$;

Przykład 1 (Yale Shooting Scenario)

$$YSS = \langle a \wedge \neg l \rangle load; wait; shoot \langle T \rangle.$$

Oczekiwaną konkluzją jest, że po wykonaniu działań $load; wait; shoot$ indyk nie żyje i strzelba jest nienaładowana.
Obliczamy D_3^{YSS}.

$$\begin{aligned} D_3^{YSS} &= sp(load; wait; shoot, a \wedge \neg l) \\ &= \neg a \wedge \neg l. \end{aligned}$$

Ponieważ $D_3^{YSS} \vdash \neg a \wedge \neg l$, więc w stanie końcowym indyk jest nieżywy i strzelba rozładowana. ∎

Przykład 2 (Russian Turkey Scenario)
(RTS) jest przykładem problemu predykcji, w którym występują działania niedeterministycze.

$$RTS = \langle a \wedge \neg l \rangle \; load; spin; shoot \; \langle T \rangle.$$

Oczekiwaną konkluzją jest, że nie możemy stwierdzić czy w stanie końcowym indyk żyje czy nie.

Obliczamy D_3^{RTS}.

$$\begin{aligned} D_3^{RTS} &= sp(load; spin; shoot, a \wedge \neg l) \\ &= \neg l. \end{aligned}$$

Ponieważ zarówno $D_3^{RTS} \not\vdash a$ jak i $D_3^{RTS} \not\vdash \neg a$, nie można wywnioskować czy w stanie końcowym indyk żyje, czy nie. ∎

Przykład 3 (Stanford Murder Mystery)
Rozważmy następujący problem: indyk żyje w stanie początkowym, a po wykonaniu działań $shoot$ i $wait$ jest nieżywy.

$$SMM = \langle a \rangle\ shoot; wait\ \langle \neg a \rangle.$$

Problemem, który nas interesuje w tym przypadku jest kiedy indyk został zabity i czy strzelba była naładowana w stanie początkowym (problem postdykcji). Obliczamy najpierw D_0^{SMM}.

$$\begin{aligned} D_0^{SMM} &= a \wedge \neg wlp(shoot; wait, a) \\ &= a \wedge \neg wlp(shoot, wlp(wait, a)) \\ &= a \wedge \neg wlp(shoot, a) \\ &= a \wedge \neg((l \Rightarrow F) \wedge (\neg l \Rightarrow a)) \\ &\equiv a \wedge l. \end{aligned}$$

Ponieważ $D_0^{SMM} \vdash l$, wnioskujemy, że strzelba była naładowana w stanie początkowym.

Teraz liczymy D_1^{SMM}.

$$\begin{aligned} D_1^{SMM} &= sp(shoot, a) \wedge \neg wlp(wait, a) \\ &= ((\neg a \wedge \neg l \wedge T) \vee (\neg a \wedge \neg l \wedge F) \vee (\neg l \wedge a)) \wedge \neg a \\ &\equiv \neg a \wedge \neg l. \end{aligned}$$

Ponieważ $D_1^{SMM} \vdash \neg a$ wnioskujemy, że indyk został zabity w wyniku wykonania $shoot$. ∎

Przykład 4

Rozważmy fluenty:

t (mam bilet),

r (jestem w czerwonym autobusie),

y (jestem w żółtym autobusie),

n (jestem w Nowym Jorku),

s (jestem w San Francisco)

oraz działania $getintobus$ i $gobybus$.

Działanie $getintobus$ jest niedeterministyczne. W wyniku wykonania tego działania jestem w czerwonym lub żółtym autobusie, o ile mam bilet.

W efekcie działania $gobybus$ jestem w Nowym Jorku, jeśli jadę czerwonym autobusem, a jestem w San Francisco, jeśli jadę żółtym autobusem.

Semantyka działań

- $wlp(getintobus, \alpha) = (t \Rightarrow (\alpha[r \leftarrow T] \wedge \alpha[y \leftarrow T])) \wedge (\neg t \Rightarrow \alpha)$
- $sp(getintobus, \alpha) = t \wedge (r \wedge \exists r.\alpha \vee y \wedge \exists y.\alpha) \vee \neg t \wedge \alpha$
- $wlp(gobybus, \alpha) = (r \Rightarrow \alpha[n \leftarrow T]) \wedge (y \Rightarrow \alpha[s \leftarrow T]) \wedge (\neg r \wedge \neg y \Rightarrow \alpha)$
- $sp(gobybus, \alpha) = n \wedge r \wedge \exists n.\alpha \vee s \wedge y \wedge \exists s.\alpha \vee \neg r \wedge \neg y \wedge \alpha$

Rozważmy scenariusz: w sytuacji początkowej nie jestem w Nowym Jorku, ani w San Francisco, ani w autobusie. Po wykonaniu działań $getintobus$ i $gobybus$ jestem w Nowym Jorku.

$$SC = \langle \neg n \wedge \neg s \wedge \neg r \wedge \neg y \rangle \ \ getintobus;\ gobybus \ \ \langle n \rangle.$$

Oczekiwaną konkluzją jest, że w stanie początkowym mam bilet (zachodzi t) oraz po wykonaniu $getintobus$ jestem w czerwonym autobusie.

$$\begin{aligned}
D_0^{SC} &= \neg n \wedge \neg s \wedge \neg r \wedge \neg y \wedge \neg wlp(getintobus; gobybus, \neg n) \\
&= \neg n \wedge \neg s \wedge \neg r \wedge \neg y \wedge \neg wlp(getintobus, wlp(gobybus, \neg n)) \\
&= \neg n \wedge \neg s \wedge \neg r \wedge \neg y \wedge \neg wlp(getintobus, \neg r \wedge (\neg y \vee \neg n) \\
&\quad \wedge (r \vee y \vee \neg n)) \\
&= \neg n \wedge \neg s \wedge \neg r \wedge \neg y \wedge \neg(\neg t \wedge \neg r \wedge (\neg y \vee \neg n) \wedge (r \vee y \vee \neg n)) \\
&\equiv \neg n \wedge \neg s \wedge \neg r \wedge \neg y \wedge t
\end{aligned}$$

Ponieważ $D_0^{SC} \vdash t$, więc w stanie początkowym tego scenariusza mam bilet.

Obliczmy teraz

$$\begin{array}{rcl}
D_1^{SC} & = & sp(getintobus, \neg n \wedge \neg s \wedge \neg r \wedge \neg y) \wedge \neg wlp(gobybus, \neg n) \\
& = & (t \wedge (r \wedge \neg n \wedge \neg s \wedge \neg y \vee y \wedge \neg r \wedge \neg n \wedge \neg s) \\
& \vee & \neg t \wedge \neg r \wedge \neg n \wedge \neg s \wedge \neg y) \wedge \neg(\neg r \wedge (\neg y \vee \neg n) \wedge (r \vee y \vee \neg n)) \\
& \equiv & \neg n \wedge \neg s \wedge (t \wedge (r \vee y) \vee (\neg t \wedge \neg r \wedge \neg y)) \\
& \wedge & (r \vee y \wedge n \vee \neg r \wedge \neg y \wedge n)) \\
& \equiv & \neg n \wedge \neg s \wedge t \wedge r
\end{array}$$

Ponieważ $D_1^{SC} \vdash r$, więc po wykonaniu $getintobus$ jestem w czerwonym autobusie.

Możliwości rozszerzenia przedstawionego formalizmu

1. Język logiki pierwszego rzędu
2. Kwalifikacja i ramifikacja
3. Działania z niepełna informacja o stanie końcowym: działania o nietypowych efektach.
4. Planowanie – ocena planów.
5. Systemy wieloagentowe – problemy z modelowaniem równoległości.

Rozszerzenie języka $SDPL$

Rozważmy przykład opisujący świat klocków.

Działanie:
Put(x,y), – efektem jest położenie klocka x na y, o ile y jest wolny i różny od x, x i y są zmiennymi.

On(x,y) oznacza: "klocek x jest na klocku y",

Clear(x) oznacza: "klocek x jest wolny".

Do opisanego formalizmu wprowadzamy działania sparametryzowane – schematy działań.
Definicja schematu działania zawiera zmienne, które w scenariuszu zostaną zastąpione odpowiednimi stałymi.

Reprezentacja predykatów *On(x,y)* i *Clear(x)* wymaga rozszerzenia języka $SDPL$ o zmienne tablicowe. Natomiast sama metoda postępowania pozostaje bez zmian.

Definicja schematu:

Put(x, y):

$$\begin{array}{rcl} \textbf{if } Clear(x) \wedge \neg(x = y) & \rightarrow & On(x, y) := T \ [\!] \\ \neg(Clear(x) \wedge \neg(x = y)) & \rightarrow & skip \ \textbf{fi} \end{array}$$

W scenariuszu wystąpi działanie np.
Put(a, b),
gdzie a, b – stałe odnoszące się do konkretnych obiektów (klocków).

Instrukcja definiująca działanie *Put*(a, b) powstaje z definicji *Put*(x, y) w wyniku tekstowego zastąpienia x przez a, a y przez b.

Ramifikacja

Przykład: Dodajmy do języka YSS dodatkowy fluent w oznaczający, że $indyk\ \ chodzi$ oraz aksjomat $IC = w \supset a$. Niech WSS będzie następującym scenariuszem

$[w \wedge l]\ shoot\ [T]$.

Licząc jak poprzednio i dodając IC

$$
\begin{aligned}
D_1 &= sp^{IC}(shoot, w \wedge l \wedge IC) \\
&= sp(shoot, w \wedge l \wedge (w \supset a)) \wedge (w \supset a) \\
&= (\neg a \wedge \neg l \wedge w \wedge T \wedge (w \supset T) \vee \neg a \wedge \neg l \wedge w \wedge T \\
&\quad \wedge (w \supset F) \vee (\neg l \wedge w \wedge l \wedge (w \supset a)) \wedge (w \supset a) \\
&\equiv ((\neg a \wedge \neg l \wedge w) \vee (\neg a \wedge \neg l \wedge w \wedge \neg w)) \\
&\quad \wedge (w \supset a) \\
&\equiv (\neg a \wedge \neg l \wedge w) \wedge (w \supset a) \\
&\equiv F
\end{aligned}
$$

dostajemy sprzeczność.
Co jest powodem tej sprzeczności?
Odp: kolizja pomiędzy inercją a ubocznym efektem względem fluentu w.
Rozwiązanie: pozbawić fluent w wartości.
Ogólnie

$$\begin{aligned} release(f_i) \quad = \quad & \textbf{if} \;\; T \rightarrow f_i := T \quad [\!] \\ & T \rightarrow f_i := F \;\; \textbf{fi} \end{aligned} \tag{8}$$

I zmodyfikować działanie A:

$$A; release(f_1); \ldots; release(f_n)$$

Złożenie $release(f_1); \ldots; release(f_n)$ zapiszemy w skrócie jako $release(f_1, \ldots, f_n)$.

Nowe obliczenia są następujące:

$$
\begin{aligned}
D_1^{IC} &= sp(shoot^*, w \wedge l \wedge IC) \wedge IC \\
&= sp(shoot; release(w), w \wedge l \wedge IC) \wedge IC \\
&\equiv sp(release(w), \neg a \wedge \neg l \wedge w) \wedge (w \supset a) \\
&\equiv \neg a \wedge \neg l \wedge (w \supset a).
\end{aligned}
$$

$D_1^{IC} \vdash \neg a \wedge \neg w.$