Semantyka i weryfikacja programów

- Celem tych zajęć jest przedstawienie podstawowych technik formalnego opisu znaczenia programów, problemów w ich opracowaniu i stosowaniu, a przede wszystkim uzasadnienie potrzeby stosowania takich metod.
- Przedstawimy kolejno różne metody definiowania semantyki języków programowania wraz z ich matematycznymi podstawami i praktycznymi technikami opisu.
- Wprowadzimy podstawowe pojęcia poprawności programów wraz z formalizmami i metodami dowodzenia takich własności.
- Wprowadzimy (choć tylko bardzo szkicowo) idee systematycznego konstruowania poprawnych programów.

Programy

```
D207 0C78 F0CE 00078 010D0
D203 0048 F0D6 00048 01CD8
8000 F0EA F0B3 010EC 00ED7
9C00 000C F0DA 0000C ...
```

```
r := 0; q := 1;
while q <= n do
  begin r := r + 1;
        q := q + 2 * r + 1 end
```

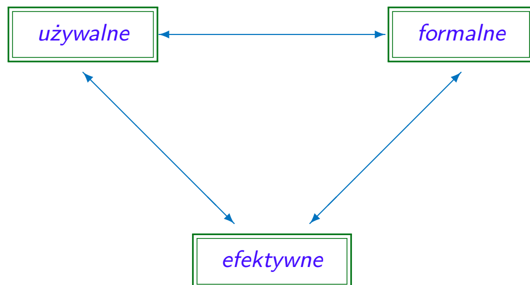
- ścisły, zrozumiały dla człowieka opis *algorytmu*
- ścisłe określenie, jakie *obliczenia* mają być wykonane przez komputer

Pożądane cechy programów:

- czytelne; efektywne; niezawodne; przyjazne dla użytkownika; dobrze udokumentowane; . . .
- ale nade wszystko, *POPRAWNE*
- no i nie zapominajmy: także, *wykonywalne*. . .

Sprzeczności

Oczekiwania wobec języków programowania:



Wielki Cel

Niezbędne aspekty dobrego języka programowania:

- Składnia
- Semantyka
- Logika
- Pragmatyka/metodyka
- Implementacja
- Środowisko programisty

Składnia

Służy do precyzyjnego zdefiniowania zbioru poprawnie zbudowanych konstrukcji języka.

– *składnia konkretna* (LL(1), LR(1), . . .)
– *składnia abstrakcyjna* (jednoznaczna gramatyka bezkontekstowa, notacja BNF itp.)
– *poprawność typów* (warunki kontekstowe, analiza statyczna)

Formalna prezentacja składni stała się obecnie standardem!

Dzięki temu są dostępne doskonałe narzędzia wspomagające analizę składniową.

Semantyka

Służy do zdefiniowania znaczenia programów i wszystkich fraz składniowych języka.

Opis nieformalny jest często niewystarczający

- semantyka operacyjna (małe kroki, duże kroki, maszynowa): wykorzystuje pojęcie *obliczenia*, ukazując *sposób*, w jaki dochodzi do wyliczenia wyników
- semantyka denotacyjna (bezpośrednia, kontynuacyjna): opisuje *znaczenie* całych konstrukcji językowych, definiuje wyniki bez wchodzenia w szczegóły związane z ich uzyskaniem
- semantyka aksjomatyczna: koncentruje się na *własnościach* konstrukcji językowych, pomijając niektóre aspekty związane z ich znaczeniem i wyliczaniem wyników

Pragmatyka

Jak należy używać języka, aby tworzyć *dobre* programy.

- prezentacja konstrukcji programistycznych przeznaczona dla użytkownika
- wskazówki dotyczące dobrego/złego stylu ich stosowania

Logika

Służy do formułowania własności programu i ich dowodzenia.

- Własności częściowej poprawności, formułowane na bazie logiki pierwszego rzędu
- Logika Hoare'a do ich dowodzenia
- Własności stopu (całkowita poprawność)

Także:

- logiki temporalne
- inne logiki modalne
- algebraiczne metody specyfikowania
- metody specyfikowania oparte na modelu abstrakcyjnym

weryfikacja programu a *tworzenie poprawnego programu*

Metodyka

- specyfikacje
- stopniowe uszczegóławianie
- projektowanie modułowej struktury programu
- kodowanie poszczególnych modułów

Implementacja

Kompilator/interpreter, z:

- analizatorem składniowym
- analizatorem statycznym i modułem optymalizującym
- generatorem kodu

Środowisko programisty

Niezbędne dla budowy i wykonywania programów:

- edytor tekstów dedykowany danemu językowi programowania
- kompilator/interpreter
- debugger
- biblioteki standardowych modułów

A TAKŻE:

- wsparcie do tworzenia specyfikacji
- narzędzia do weryfikacji
- ...

Po co nam formalna semantyka?

Abyśmy mogli spać spokojnie wiedząc, że:

- wszystkie *konstrukcje* językowe i stojące u ich podstaw *pojęcia* są precyzyjnie, dokładnie i jednoznacznie rozumiane
- jesteśmy niezależni od konkretnej implementacji
- potrafimy łatwo tworzyć implementacje prototypowe
- mamy niezbędne podstawy do wiarygodnego wnioskowania

Przykład

Przypomnijmy:

```
r := 0; q := 1;
while q <= n do
  begin r := r + 1;
        q := q + 2 * r + 1
  end
```

Lub lepiej:

$$
\begin{array}{l}
rt := 0; sqr := 1; \\
\textbf{while } sqr \leqslant n \textbf{ do } (rt := rt + 1; \\
\qquad\qquad\qquad\qquad sqr := sqr + 2 * rt + 1)
\end{array}
$$

Program oblicza część całkowitą pierwiastka kwadratowego, prawda?

$$
\begin{array}{l}
\{n \geqslant 0\} \\
rt := 0; sqr := 1; \\
\{n \geqslant 0 \land rt = 0 \land sqr = 1\} \\
\textbf{while } \{sqr = (rt+1)^2 \land rt^2 \leqslant n\} \; sqr \leqslant n \textbf{ do} \\
\qquad (rt := rt + 1; \\
\qquad \{sqr = rt^2 \land sqr \leqslant n\} \\
\qquad sqr := sqr + 2 * rt + 1) \\
\{rt^2 \leqslant n < (rt+1)^2\}
\end{array}
$$

Ale jak uzasadnić niejawne użycie asercji i reguł dowodzenia?

Przykładowa reguła dowodzenia

Na przykład:

$$\{sqr = rt^2 \wedge sqr \leqslant n\}\ sqr := sqr + 2 * rt + 1\ \{sqr = (rt + 1)^2 \wedge rt^2 \leqslant n\}$$

wynika z reguły:

$$\{\varphi[E/x]\}\ x := E\ \{\varphi\}$$

ALE: choć reguła ta jest *zasadniczo* poprawna, to jednak z wielu powodów nie działa w przypadku języka PASCAL (zakończenie z błędem, zapętlenie, efekty uboczne, przypisania na elementy tablicy, wskaźniki i aliasy itp.)

Bądź formalny i ścisły!

Uzasadnianie poprawności

- definicja semantyki programu
- definicja spełniania warunków poprawności
- reguły dowodowe dla warunków poprawności
- dowód poprawności logicznej wszystkich reguł
- analiza pełności systemu reguł

Plan zajęć

- Wstęp
- Semantyka operacyjna
- Semantyka denotacyjna
- Podstawy semantyki denotacyjnej
- Bardziej złożone konstrukcje
- Poprawność częściowa: logika Hoare'a
- Poprawność całkowita: dowodzenie stopu
- Systematyczne wyprowadzanie programów
- Semantyka: podejście algebraiczne
- Specyfikowanie programów i ich konstruowanie

Składnia

Są standardowe metody definicji składni języków programowania. Więcej na wykładzie:

Języki i automaty

Podstawowe pojęcia:

- *języki formalne*
- *gramatyki*: regularne (nieco zbyt słabe), *bezkontekstowe* (w sam raz), kontekstowe (zbyt mocne), . . .

Na marginesie: istnieją mechanizmy wykorzystujące gramatyki do definiowania semantyki języków programowania: gramatyki atrybutywne, a także gramatyki dwupoziomowe, patrz:

Metody implementacji języków programowania

Składnia konkretna

Składnię konkretną języka programowania zazwyczaj definiuje się za pomocą gramatyki (bezkontekstowej), szczegółowo opisującej wszystkie „przecinki i średniki" niezbędne do tego, aby napisany ciąg znaków był poprawnie zbudowanym programem. Zazwyczaj towarzyszą jej dodatkowe warunki konktekstowe, które eliminują niektóre napisy dopuszczane przez gramatykę (np. „nie będziesz używać niezadeklarowanej zmiennej").
Opis języka formalnego za pomocą jednoznacznej gramatyki bezkontekstowej nadaje pewną *strukturę* napisom języka: pokazuje, w jaki sposób za pomocą lingwistycznych *konstrukcji* języka, utworzyć poprawnie zbudowany napis z jego bezpośrednich składowych.

Składnia abstrakcyjna

Składnia abstrakcyjna wyraża strukturę fraz programu przez podanie zastosowanych do ich tworzenia konstrukcji lingwistycznych języka. Z każdą frazą składniową związuje jej *bezpośrednie* składowe oraz *konstrukcję* zastosowaną do jej utworzenia.
Składnię abstrakcyjną można uważać za opis budowy każdej frazy języka wyrażony jako drzewo rozbioru: każdy wierzchołek jest etykietowany konstrukcją lingwistyczną, a poddrzewa reprezentują składowe.
Analiza składniowa odwzorowuje składnię konkretną na składnię abstrakcyjną poprzez zbudowanie drzewa rozbioru dla każdej frazy języka zgodnie z definicją zadaną przez składnię konkretną.
Powyższe pojęcia (i wiele innych) wyjaśniono na innych zajęciach.

Ten przedmiot

Nie będziemy wnikać w różnice między składnią konkretną a abstrakcyjną.

- składnię prezentujemy w postaci składni konkretnej
- frazy językowe używamy tak, jakby były zadane przez składnię abstrakcyjną
- w razie niejednoznaczności stosujemy nawiasy i wcięcia — uzyskując w ten sposób jednoznaczą interpretację frazy językowej w postaci pewnego drzewa rozbioru

To podejście nie jest właściwe w przypadku prawdziwych języków programowania, ale działa wystarczająco dobrze w przedstawianych w dalszym ciągu przykładach

Przykład wiodący

Jako podstawowy przykład dla prezentacji metod definiowania semantyki wykorzystamy prosty, iteracyjny język programowania:

- proste wyrażenia arytmetyczne
- proste wyrażenia logiczne
- proste instrukcje (przypisanie, instrukcja warunkowa, pętla)

Kategorie składniowe

- *stałe liczbowe*

$$\boxed{N \in \mathbf{Num}}$$

o następującej składni:

$N ::= 0 \mid 1 \mid 2 \mid \cdots$

- *zmienne*

$$\boxed{x \in \mathbf{Var}}$$

o następującej składni:

$x ::= \cdots$ *ciągi liter i cyfr rozpoczynające się od litery* $\cdots$

- *wyrażenia (arytmetyczne)*

$$\boxed{e \in \mathbf{Exp}}$$

o następującej składni:

$e ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2$

Kategorie składniowe, c.d.

- *wyrażenia logiczne*

$$\boxed{b \in \mathbf{BExp}}$$

o następującej składni:

$$b ::= \mathbf{true} \mid \mathbf{false} \mid e_1 \leqslant e_2 \mid \neg b' \mid b_1 \wedge b_2$$

- *instrukcje*

$$\boxed{S \in \mathbf{Stmt}}$$

o następującej składni:

$$S ::= x := e \mid \mathbf{skip} \mid S_1; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ b\ \mathbf{do}\ S'$$

Zanim przejdziemy dalej

(do semantyki)

Użyta wcześniej definicja składni, np:

- *wyrażenia (arytmetyczne)*

  $e \in \mathbf{Exp}$

  o następującej składni:

  $e ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2$

oznacza, że wszystkie wyrażenia mają jedną z powyższych postaci, wszystkie te postacie są różne oraz każde wyrażenie można zbudować stosując w pewnej kolejności te konstrukcje.

Zatem do definiowania i dowodzenia można tu stosować INDUKCJĘ (STRUKTURALNĄ)

Indukcja strukturalna

Niech $P(_)$ będzie pewną własnością dotyczącą wyrażeń:
JEŚLI

- $P(N)$, dla każdego $N \in$ **Num**
- $P(x)$, dla każdego $x \in$ **Var**
- $P(e_1 + e_2)$ wynika z $P(e_1)$ i $P(e_2)$, dla każdego $e_1, e_2 \in$ **Exp**
- $P(e_1 * e_2)$ wynika z $P(e_1)$ i $P(e_2)$, dla każdego $e_1, e_2 \in$ **Exp**
- $P(e_1 - e_2)$ wynika z $P(e_1)$ i $P(e_2)$, dla każdego $e_1, e_2 \in$ **Exp**

TO

- $P(e)$ dla każdego $e \in$ **Exp**.

Zmienne wolne wyrażeń $FV(e) \subset \mathbf{Var}$:

$$
\begin{array}{ll}
FV(N) & = \emptyset \\
FV(x) & = \{x\} \\
FV(e_1 + e_2) & = FV(e_1) \cup FV(e_2) \\
FV(e_1 * e_2) & = FV(e_1) \cup FV(e_2) \\
FV(e_1 - e_2) & = FV(e_1) \cup FV(e_2)
\end{array}
$$

Fakt:

Dla każdego wyrażenia $e \in \mathbf{Exp}$, *zbiór* $FV(e)$ *jego zmiennych wolnych jest skończony.*

Kategorie semantyczne

Najpierw łatwe rzeczy:

- *wartości logiczne*

  $\mathbf{Bool} = \{\mathbf{tt}, \mathbf{ff}\}$

- *liczby całkowite*

  $\mathbf{Int} = \{0, 1, -1, 2, -2, \ldots\}$

  wraz z oczywistą funkcją semantyczną:

$$\mathcal{N} : \mathbf{Num} \to \mathbf{Int}$$

$$\begin{aligned} \mathcal{N}[\![0]\!] &= 0 \\ \mathcal{N}[\![1]\!] &= 1 \\ \mathcal{N}[\![2]\!] &= 2 \\ &\ldots \end{aligned}$$

Na marginesie: $_[\![_]\!]$ jest po prostu zastosowaniem funkcji semantycznej, przy czym $[\![\ ]\!]$ używa się, aby oddzielić frazę składniową od kontekstu semantycznego.

Wartościowania zmiennych

- *stany* (na razie: funkcje (całkowite) z **Var** w **Int**)

  $s \in \textbf{State} = \textbf{Var} \rightarrow \textbf{Int}$

  - wyszukanie (wartości zmiennej x w stanie s) jest po prostu zastosowaniem funkcji

    $s\ x$
  - uaktualnienie stanu: $s' = s[y \mapsto n]$

    $$s'\ x = \begin{cases} s\ x & \text{jeśli } x \neq y \\ n & \text{jeśli } x = y \end{cases}$$

$$\mathcal{E} : \textbf{Exp} \to (\textbf{State} \to \textbf{Int})$$

definiowana w oczywisty sposób:

$$\begin{aligned}
\mathcal{E}[\![N]\!]\, s &= \mathcal{N}[\![N]\!] \\
\mathcal{E}[\![x]\!]\, s &= s\, x \\
\mathcal{E}[\![e_1 + e_2]\!]\, s &= \mathcal{E}[\![e_1]\!]\, s + \mathcal{E}[\![e_2]\!]\, s \\
\mathcal{E}[\![e_1 * e_2]\!]\, s &= \mathcal{E}[\![e_1]\!]\, s * \mathcal{E}[\![e_2]\!]\, s \\
\mathcal{E}[\![e_1 - e_2]\!]\, s &= \mathcal{E}[\![e_1]\!]\, s - \mathcal{E}[\![e_2]\!]\, s
\end{aligned}$$

Na marginesie: Będziemy często używać funkcji wyższych rzędów!

Bez dalszych ostrzeżeń!

Semantyka wyrażeń logicznych

$$\mathcal{B} \colon \mathbf{BExp} \to (\mathbf{State} \to \mathbf{Bool})$$

definiowana w oczywisty sposób:

$$\mathcal{B}[\![\mathbf{true}]\!]\, s = \mathbf{tt}$$

$$\mathcal{B}[\![\mathbf{false}]\!]\, s = \mathbf{ff}$$

$$\mathcal{B}[\![e_1 \leqslant e_2]\!]\, s = \begin{cases} \mathbf{tt} & \text{jeśli } \mathcal{E}[\![e_1]\!]\, s \leqslant \mathcal{E}[\![e_2]\!]\, s \\ \mathbf{ff} & \text{jeśli } \mathcal{E}[\![e_1]\!]\, s \not\leqslant \mathcal{E}[\![e_2]\!]\, s \end{cases}$$

$$\mathcal{B}[\![\neg b]\!]\, s = \begin{cases} \mathbf{ff} & \text{jeśli } \mathcal{B}[\![b]\!]\, s = \mathbf{tt} \\ \mathbf{tt} & \text{jeśli } \mathcal{B}[\![b]\!]\, s = \mathbf{ff} \end{cases}$$

$$\mathcal{B}[\![b_1 \wedge b_2]\!]\, s = \begin{cases} \mathbf{tt} & \text{jeśli } \mathcal{B}[\![b_1]\!]\, s = \mathbf{tt} \text{ i } \mathcal{B}[\![b_2]\!]\, s = \mathbf{tt} \\ \mathbf{ff} & \text{jeśli } \mathcal{B}[\![b_1]\!]\, s = \mathbf{ff} \text{ lub } \mathcal{B}[\![b_2]\!]\, s = \mathbf{ff} \end{cases}$$

Semantyka instrukcji

Zdefiniujemy ją na kilka sposobów, aby zilustrować różne podejścia do semantyki formalnej.

Poprzednie definicje traktujemy jako pomocnicze

Zanim przejdziemy dalej

(do semantyki instrukcji)

Fakt:

Znaczenie wyrażenia zależy jedynie od wartości jego zmiennych wolnych: dla każdych $e \in \mathbf{Exp}$ *i* $s, s' \in \mathbf{State}$, *jeśli* $s\,x = s'\,x$ *dla każdego* $x \in FV(e)$ *to*

$$\mathcal{E}[\![e]\!]s = \mathcal{E}[\![e]\!]s'.$$

Dowód za chwilę. . .

Ćwiczenie:

Sformułuj (i udowodnij) analogiczną własność dla wyrażeń logicznych.

Dowód

Przez indukcję strukturalną:

- dla $N \in \textbf{Num}$, $\mathcal{E}[\![N]\!]\, s = \mathcal{N}[\![N]\!]$
  $= \mathcal{E}[\![e]\!]\, s'$
- dla $x \in \textbf{Var}$, $\mathcal{E}[\![x]\!]\, s = s\ x$
  $= s'\ x$ (gdyż $x \in FV(x)$)
  $= \mathcal{E}[\![x]\!]\, s'$
- dla $e_1, e_2 \in \textbf{Exp}$,
  $\mathcal{E}[\![e_1 + e_2]\!]\, s = \mathcal{E}[\![e_1]\!]\, s + \mathcal{E}[\![e_2]\!]\, s$
  $= \mathcal{E}[\![e_1]\!]\, s' + \mathcal{E}[\![e_2]\!]\, s'$ (z założenia indukcyjnego, gdyż $FV(e_1) \subseteq FV(e_1 + e_2)$, i podobnie dla e_2)
  $= \mathcal{E}[\![e_1 + e_2]\!]\, s'$
- ...

Przezroczystość odwołań…

Podstawienie e' na x w e daje wyrażenie $e[e'/x]$:

$$
\begin{array}{ll}
N[e'/x] & = N \\
x'[e'/x] & = \begin{cases} e' & \text{jeśli } x = x' \\ x' & \text{jeśli } x \neq x' \end{cases} \\
(e_1 + e_2)[e'/x] & = e_1[e'/x] + e_2[e'/x] \\
(e_1 * e_2)[e'/x] & = e_1[e'/x] * e_2[e'/x] \\
(e_1 - e_2)[e'/x] & = e_1[e'/x] - e_2[e'/x]
\end{array}
$$

Udowodnij:

$$
\boxed{\mathcal{E}[\![e[e'/x]]\!]\; s = \mathcal{E}[\![e]\!]\; s[x \mapsto \mathcal{E}[\![e']\!]\; s]}
$$

semantyka małych kroków

Ogólna idea:

- definiujemy *konfiguracje*: $\gamma \in \Gamma$
- niektóre z nich oznaczamy jako *końcowe*: $\mathrm{T} \subseteq \Gamma$
- definiujemy *(jednokrokową) relację przejścia*: $\Rightarrow \subseteq \Gamma \times \Gamma$
  - dla $\gamma \in \mathrm{T}$, zazwyczaj $\gamma \not\Rightarrow$
- badamy *obliczenia*: (skończone bądź nieskończone) ciągi konfiguracji

$$\gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}, \ldots$$

takie że $\gamma_i \Rightarrow \gamma_{i+1}$. Zapisujemy je w postaci:

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_i \Rightarrow \gamma_{i+1} \Rightarrow \cdots$$

Obliczenia

Obliczenia mogą:

- kończyć się: $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \in \mathrm{T}$
- zakleszczać się: $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \notin \mathrm{T}$ i $\gamma_n \not\Rightarrow$
- być nieskończone (pętlić się): $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots$

Ponadto:

- $\gamma \Rightarrow^k \gamma'$ dla $k \geqslant 0$, jeśli istnieje obliczenie $\gamma = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_k = \gamma'$
- $\gamma \Rightarrow^* \gamma'$ jeśli $\gamma \Rightarrow^k \gamma'$ dla pewnego $k \geqslant 0$

Na marginesie: $\Rightarrow^* \subseteq \Gamma \times \Gamma$ jest najmniejszą relacją zwrotną i przechodnią zawierającą $\Rightarrow$.

Konfiguracje: $\Gamma = (\textbf{Stmt} \times \textbf{State}) \cup \textbf{State}$
Konfiguracje końcowe: $\mathrm{T} = \textbf{State}$
Relacja przejścia zawiera jedynie:

$$\langle x := e, s \rangle \Rightarrow s[x \mapsto (\mathcal{E}[\![e]\!]\ s)]$$
$$\langle \textbf{skip}, s \rangle \Rightarrow s$$
$$\langle S_1; S_2, s \rangle \Rightarrow \langle S_1'; S_2, s' \rangle \quad \text{jeśli } \langle S_1, s \rangle \Rightarrow \langle S_1', s' \rangle$$
$$\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle \quad \text{jeśli } \langle S_1, s \rangle \Rightarrow s'$$
$$\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle \quad \text{jeśli } \mathcal{B}[\![b]\!]\ s = \textbf{tt}$$
$$\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s \rangle \Rightarrow \langle S_2, s \rangle \quad \text{jeśli } \mathcal{B}[\![b]\!]\ s = \textbf{ff}$$
$$\langle \textbf{while } b \textbf{ do } S, s \rangle \Rightarrow \langle S; \textbf{while } b \textbf{ do } S, s \rangle \quad \text{jeśli } \mathcal{B}[\![b]\!]\ s = \textbf{tt}$$
$$\langle \textbf{while } b \textbf{ do } S, s \rangle \Rightarrow s \quad \text{jeśli } \mathcal{B}[\![b]\!]\ s = \textbf{ff}$$

Niektóre własności

Fakt:

TINY *jest deterministyczny, tzn.: dla każdej konfiguracji* $\langle S, s \rangle$

$$\textit{jeśli } \langle S, s \rangle \Rightarrow \gamma_1 \textit{ i } \langle S, s \rangle \Rightarrow \gamma_2 \textit{ to } \gamma_1 = \gamma_2.$$

Dowód: Przez indukcję strukturalną po S.

Wniosek:

W języku TINY*, dla każdej konfiguracji* $\langle S, s \rangle$ *istnieje dokładnie jedno obliczenie rozpoczynające się w* $\langle S, s \rangle$.

Inna technika dowodowa:

Indukcja po długości obliczenia

Fakt:

Jeśli $\langle S_1; S_2, s\rangle \Rightarrow^k s'$ *to* $\langle S_1, s\rangle \Rightarrow^{k_1} \hat{s}$ *i* $\langle S_2, \hat{s}\rangle \Rightarrow^{k_2} s'$*, dla pewnych* $\hat{s} \in$ **State** *i* $k_1, k_2 > 0$ *takich że* $k = k_1 + k_2$*.*

Dowód: Przez indukcję po k:

$k = 0$: OK

$k > 0$: Wtedy $\langle S_1; S_2, s\rangle \Rightarrow \gamma \Rightarrow^{k-1} s'$. Z definicji relacji przejścia są tylko dwie możliwości:

- $\gamma = \langle S_2, \hat{s}\rangle$, przy czym $\langle S_1, s\rangle \Rightarrow \hat{s}$. OK
- $\gamma = \langle S_1'; S_2, s''\rangle$, przy czym $\langle S_1, s\rangle \Rightarrow \langle S_1', s''\rangle$. Wówczas z założenia indukcyjnego, $\langle S_1', s''\rangle \Rightarrow^{k_1} \hat{s}$ oraz $\langle S_2, \hat{s}\rangle \Rightarrow^{k_2} s'$, dla pewnych $\hat{s} \in$ **State** i $k_1, k_2 > 0$, takich że $k - 1 = k_1 + k_2$. OK

Fakt:

Pozostały kontekst nie wpływa na obliczenia:

Jeśli $\langle S_1, s\rangle \Rightarrow^k \langle S_1', s'\rangle$ *to* $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_1'; S_2, s'\rangle$;
jeśli $\langle S_1, s\rangle \Rightarrow^k s'$ *to* $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_2, s'\rangle$.

Warianty definicji

- zamiast przedstawionych reguł dla **if**:

$$\begin{array}{ll}
\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1', s'\rangle & \text{jeśli } \mathcal{B}[\![b]\!]\, s = \textbf{tt} \text{ i } \langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle \\
\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow s' & \text{jeśli } \mathcal{B}[\![b]\!]\, s = \textbf{tt} \text{ i } \langle S_1, s\rangle \Rightarrow s' \\
\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2', s'\rangle & \text{jeśli } \mathcal{B}[\![b]\!]\, s = \textbf{ff} \text{ i } \langle S_2, s\rangle \Rightarrow \langle S_2', s'\rangle \\
\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow s' & \text{jeśli } \mathcal{B}[\![b]\!]\, s = \textbf{ff} \text{ i } \langle S_2, s\rangle \Rightarrow s'
\end{array}$$

- podobnie dla **while**, pierwszy przypadek
- zamiast przedstawionych reguł dla **while**:

$$\langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow \langle \textbf{if } b \textbf{ then } (S; \textbf{while } b \textbf{ do } S) \textbf{ else skip}, s\rangle$$

- ...

semantyka wielkich kroków

Ogólna idea:

- definiujemy *konfiguracje*: $\gamma \in \Gamma$
- niektóre z nich oznaczamy jako *końcowe*: $\mathrm{T} \subseteq \Gamma$
- zamiast obliczeń rozważamy (definiujemy) *przejścia* bezpośrednio *do konfiguracji końcowych*, do których prowadzą obliczenia: $\leadsto \subseteq \Gamma \times \mathrm{T}$

Nieformalnie:

– jeśli $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \in \mathrm{T}$, to $\gamma_0 \leadsto \gamma_n$

– jeśli $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \notin \mathrm{T}$ oraz $\gamma_n \not\Rightarrow$, to $\gamma_0 \not\leadsto$

– jeśli $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots$ to $\gamma_0 \not\leadsto$

TINY: semantyka naturalna

$\langle x := e, s\rangle \rightsquigarrow s[x \mapsto (\mathcal{E}[\![e]\!]\ s)]$

$\langle \mathbf{skip}, s\rangle \rightsquigarrow s$

$$\frac{\langle S_1, s\rangle \rightsquigarrow s' \qquad \langle S_2, s'\rangle \rightsquigarrow s''}{\langle S_1; S_2, s\rangle \rightsquigarrow s''}$$

$$\frac{\langle S_1, s\rangle \rightsquigarrow s'}{\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \rightsquigarrow s'} \text{ jeśli } \mathcal{B}[\![b]\!]\ s = \mathbf{tt}$$

$$\frac{\langle S_2, s\rangle \rightsquigarrow s'}{\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \rightsquigarrow s'} \text{ jeśli } \mathcal{B}[\![b]\!]\ s = \mathbf{ff}$$

$$\frac{\langle S, s\rangle \rightsquigarrow s' \qquad \langle \mathbf{while}\ b\ \mathbf{do}\ S, s'\rangle \rightsquigarrow s''}{\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \rightsquigarrow s''} \text{ jeśli } \mathcal{B}[\![b]\!]\ s = \mathbf{tt}$$

$\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \rightsquigarrow s$ jeśli $\mathcal{B}[\![b]\!]\ s = \mathbf{ff}$

Konfiguracje:
$\Gamma = (\mathbf{Stmt} \times \mathbf{State}) \cup \mathbf{State}$

Konfiguracje końcowe: jak wyżej
$\mathrm{T} = \mathbf{State}$

Przejścia: jak podano tutaj

Jak to rozumieć?

„na gruncie teorii zbiorów"

Jak poprzednio:

$\rightsquigarrow \subseteq \Gamma \times \mathrm{T}$ jest najmniejszą relacją, taką że

- $\langle x := e, s\rangle \rightsquigarrow s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$, dla każdych $x \in \textbf{Var}$, $e \in \textbf{Exp}$, $s \in \textbf{State}$
- ...
- $\langle S_1; S_2, s\rangle \rightsquigarrow s''$ jeśli $\langle S_1, s\rangle \rightsquigarrow s'$ i $\langle S_2, s'\rangle \rightsquigarrow s''$, dla każdych $S_1, S_2 \in \textbf{Stmt}$, $s, s', s'' \in \textbf{State}$
- $\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \rightsquigarrow s'$ jeśli $\langle S_1, s\rangle \rightsquigarrow s'$ i $\mathcal{B}[\![b]\!]\, s = \textbf{tt}$, dla każdych $b \in \textbf{BExp}$, $S_1, S_2 \in \textbf{Stmt}$, $s, s' \in \textbf{State}$
- ...

Jak to rozumieć?

„*jako dowód*”

Podajemy

- aksjomaty, np. $\langle x := e, s\rangle \rightsquigarrow s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$, oraz
- reguły, np. $\dfrac{\langle S_1, s\rangle \rightsquigarrow s' \qquad \langle S_2, s'\rangle \rightsquigarrow s''}{\langle S_1; S_2, s\rangle \rightsquigarrow s''}$

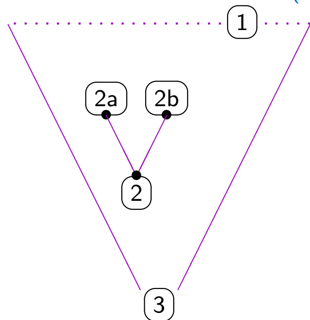
za pomocą których *wyprowadzamy* (lub też: *dowodzimy*) *stwierdzenia* postaci

$$\langle S, s\rangle \rightsquigarrow s'$$

Tak naprawdę: podajemy *schematy* aksjomatów i reguł, w których można *dowolnie* dobierać elementy podstawiane za występujące w nich metazmienne ($x \in$ **Var**, $e \in$ **Exp**, $s, s', s'' \in$ **State**, $S_1, S_2 \in$ **Stmt** itd.).

Dowody/wyprowadzenia

Skończone *drzewo dowodu* (lub *drzewo wyprowadzenia*):



- liście: etykietowane aksjomatami, np.
  (1) : $\langle x := e, s\rangle \rightsquigarrow s[x \mapsto (\mathcal{E}[\![e]\!]\ s)]$
- pozostałe węzły: etykietowane zgodnie z regułami, np.
  $$\frac{\boxed{2a} : \langle S_1, s\rangle \rightsquigarrow s' \quad \boxed{2b} : \langle S_2, s'\rangle \rightsquigarrow s''}{\boxed{2} : \langle S_1; S_2, s\rangle \rightsquigarrow s''}$$
- korzeń: udowodnione stwierdzenie,
  (3) : $\langle S, s\rangle \rightsquigarrow s'$

Często używamy notacji $\boxed{\vdash \langle S, s\rangle \rightsquigarrow s'}$ dla oznaczenia, że istnieje dowód stwierdzenia $\langle S, s\rangle \rightsquigarrow s'$.

Kolejna technika dowodowa

Indukcja po strukturze drzewa dowodu

Aby udowodnić jeśli $\vdash \langle S, s \rangle \rightsquigarrow s'$ to $P(S, s, s')$ pokazujemy, że:

- $P(x := e, s, s[x \mapsto (\mathcal{E}[\![e]\!]\ s)])$
- $P(\mathbf{skip}, s, s)$
- $P(S_1; S_2, s, s'')$ wynika z $P(S_1, s, s')$ oraz $P(S_2, s', s'')$
- $P(\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s, s')$ wynika z $P(S_1, s, s')$, gdy $\mathcal{B}[\![b]\!]\ s = \mathbf{tt}$
- $P(\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s, s')$ wynika $P(S_2, s, s')$, gdy $\mathcal{B}[\![b]\!]\ s = \mathbf{ff}$
- $P(\mathbf{while}\ b\ \mathbf{do}\ S, s, s'')$ wynika z $P(S, s, s')$ oraz $P(\mathbf{while}\ b\ \mathbf{do}\ S, s', s'')$, gdy $\mathcal{B}[\![b]\!]\ s = \mathbf{tt}$
- $P(\mathbf{while}\ b\ \mathbf{do}\ S, s, s)$, gdy $\mathcal{B}[\![b]\!]\ s = \mathbf{ff}$

uwaga na kwantyfikację

Niektóre własności

Fakt:

TINY *jest* *deterministyczny*, *tzn.:*

dla każdego $\vdash \langle S, s \rangle \rightsquigarrow s'$, *jeśli* $\vdash \langle S, s \rangle \rightsquigarrow s''$ *to* $s' = s''$.

Dowód: przez (łatwą) indukcję po dowodzie $\vdash \langle S, s \rangle \rightsquigarrow s'$.

Na marginesie: Czy daje się to także udowodnić przez indukcję po strukturze S — *co stanowi problem?*

semantyka **while** *nie* jest *kompozycjonalna*.

Więcej o kompozycjonalności później

Równoważność semantyczna

Instrukcje $S_1, S_2 \in$ **Stmt** są *równoważne naturalnie* (równoważne względem semantyki naturalnej)

$$\boxed{S_1 \equiv_{\mathcal{NS}} S_2}$$

jeśli dla każdych stanów $s, s' \in$ **State**,

$$\vdash \langle S_1, s\rangle \rightsquigarrow s' \quad \text{wttw} \quad \vdash \langle S_2, s\rangle \rightsquigarrow s'$$

Fakt:

Przez indukcję po wyprowadzeniu można udowodnić, że:

- $S; \textbf{skip} \equiv_{\mathcal{NS}} \textbf{skip}; S \equiv_{\mathcal{NS}} S$
- $(S_1; S_2); S_3 \equiv_{\mathcal{NS}} S_1; (S_2; S_3)$
- $\textbf{while}\ b\ \textbf{do}\ S \equiv_{\mathcal{NS}} \textbf{if}\ b\ \textbf{then}\ (S; \textbf{while}\ b\ \textbf{do}\ S)\ \textbf{else skip}$
- $\textbf{if}\ b\ \textbf{then}\ (\textbf{if}\ b'\ \textbf{then}\ S_1\ \textbf{else}\ S_1')\ \textbf{else}\ S_2$
  $\equiv_{\mathcal{NS}} \textbf{if}\ b \wedge b'\ \textbf{then}\ S_1\ \textbf{else}\ (\textbf{if}\ b \wedge \neg b'\ \textbf{then}\ S_1'\ \textbf{else}\ S_2)$

Własność kongruencji

Fakt:

Konstrukcje językowe zachowują równoważność semantyczną:

- *jeśli* $S_1 \equiv_{\mathcal{NS}} S_1'$ *oraz* $S_2 \equiv_{\mathcal{NS}} S_2'$ *to*

  $$\boxed{S_1; S_2 \equiv_{\mathcal{NS}} S_1'; S_2'}$$

- *jeśli* $S_1 \equiv_{\mathcal{NS}} S_1'$ *oraz* $S_2 \equiv_{\mathcal{NS}} S_2'$ *to*

  $$\boxed{\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \equiv_{\mathcal{NS}} \textbf{if } b \textbf{ then } S_1' \textbf{ else } S_2'}$$

- *jeśli* $S \equiv_{\mathcal{NS}} S'$ *to*

  $$\boxed{\textbf{while } b \textbf{ do } S \equiv_{\mathcal{NS}} \textbf{while } b \textbf{ do } S'}$$

Na marginesie: W podobny sposób można rozszerzyć definicję równoważności na wyrażenia i wyrażenia logiczne

„Są one w zasadzie takie same"

Fakt:

Obie semantyki są równoważne względem opisywanych przez nie wyników końcowych:

$$\vdash \langle S, s \rangle \rightsquigarrow s' \;\; \text{wttw} \;\; \langle S, s \rangle \Rightarrow^* s'$$

*dla wszystkich instrukcji $S \in$ **Stmt** i stanów $s, s' \in$ **State**.*

Dowód:

„$\Longrightarrow$": Przez indukcję po strukturze wyprowadzenia $\langle S, s \rangle \rightsquigarrow s'$.

„$\Longleftarrow$": Przez indukcję po długości obliczenia $\langle S, s \rangle \Rightarrow^* s'$.

„ $\Longrightarrow$": Przez indukcję po strukturze wyprowadzenia $\langle S, s\rangle \rightsquigarrow s'$.

- $\langle x := e, s\rangle \Rightarrow s[x \mapsto (\mathcal{E}[\![e]\!]\ s)]$. OK
- $\langle \textbf{skip}, s\rangle \Rightarrow s$. OK
- Przypuśćmy $\langle S_1, s\rangle \rightsquigarrow s'$ i $\langle S_2, s'\rangle \rightsquigarrow s''$, więc $\langle S_1, s\rangle \Rightarrow^* s'$ i $\langle S_2, s'\rangle \Rightarrow^* s''$. Wtedy $\langle S_1; S_2, s\rangle \Rightarrow^* \langle S_2, s'\rangle \Rightarrow^* s''$. OK
- Przypuśćmy $\mathcal{B}[\![b]\!]\ s = \textbf{tt}$ i $\langle S_1, s\rangle \rightsquigarrow s'$, więc $\langle S_1, s\rangle \Rightarrow^* s'$. Wtedy $\langle \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle \Rightarrow^* s'$. OK
- Przypuśćmy $\mathcal{B}[\![b]\!]\ s = \textbf{ff}$ i $\langle S_2, s\rangle \rightsquigarrow s'$, więc $\langle S_2, s\rangle \Rightarrow^* s'$. Wtedy $\langle \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2, s\rangle \Rightarrow \langle S_2, s\rangle \Rightarrow^* s'$. OK
- Przypuśćmy $\mathcal{B}[\![b]\!]\ s = \textbf{tt}$ i $\langle S, s\rangle \rightsquigarrow s'$ i $\langle \textbf{while}\ b\ \textbf{do}\ S, s'\rangle \rightsquigarrow s''$, więc $\langle S, s\rangle \Rightarrow^* s'$ i $\langle \textbf{while}\ b\ \textbf{do}\ S, s'\rangle \Rightarrow^* s''$. Wtedy $\langle \textbf{while}\ b\ \textbf{do}\ S, s\rangle \Rightarrow \langle S; \textbf{while}\ b\ \textbf{do}\ S, s\rangle \Rightarrow^* \langle \textbf{while}\ b\ \textbf{do}\ S, s'\rangle \Rightarrow^* s''$. OK
- Jeśli $\mathcal{B}[\![b]\!]\ s = \textbf{ff}$ to $\langle \textbf{while}\ b\ \textbf{do}\ S, s\rangle \Rightarrow s$. OK

„ $\Longleftarrow$": Przez indukcję po długości obliczenia $\langle S, s\rangle \Rightarrow^* s'$.

$\langle S, s\rangle \Rightarrow^k s'$: Weźmy $k > 0$ i $\boxed{\langle S, s\rangle \Rightarrow \gamma \Rightarrow^{k-1} s'}$. W zależności od pierwszego kroku (tylko kilka przykładowych przypadków):

- $\langle x := e, s\rangle \Rightarrow s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$. Wówczas $s' = s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$; $\langle x := e, s\rangle \rightsquigarrow s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$. OK
- $\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s''\rangle$, z $\langle S_1, s\rangle \Rightarrow \langle S_1', s''\rangle$.
  Wtedy $\langle S_1'; S_2, s''\rangle \Rightarrow^{k-1} s'$, więc $\langle S_1', s''\rangle \Rightarrow^{k_1} \widehat{s''}$ i $\langle S_2, \widehat{s''}\rangle \Rightarrow^{k_2} s'$, dla $k_1, k_2 > 0$ z $k_1 + k_2 = k - 1$.
  Wtedy $\langle S_1, s\rangle \rightsquigarrow \widehat{s''}$ i $\langle S_2, \widehat{s''}\rangle \rightsquigarrow s'$, więc $\langle S_1; S_2, s\rangle \rightsquigarrow s'$. OK
- $\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$, z $\mathcal{B}[\![b]\!]\, s = \textbf{tt}$. Wtedy $\langle S_1, s\rangle \Rightarrow^{k-1} s'$, więc $\langle S_1, s\rangle \rightsquigarrow s'$ i $\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \rightsquigarrow s'$.
  OK
- $\langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } b \textbf{ do } S, s\rangle$, z $\mathcal{B}[\![b]\!]\, s = \textbf{tt}$. Wtedy $\langle S; \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow^{k-1} s'$, zatem $\langle S, s\rangle \Rightarrow^{k_1} \hat{s}$ i $\langle \textbf{while } b \textbf{ do } S, \hat{s}\rangle \Rightarrow^{k_2} s'$, dla $k_1, k_2 > 0$ z $k_1 + k_2 = k - 1$. Zatem także $\langle S_1, s\rangle \Rightarrow^{k_1+1} \widehat{s''}$. Zatem $\langle S, s\rangle \rightsquigarrow \hat{s}$, $\langle \textbf{while } b \textbf{ do } S, \hat{s}\rangle \rightsquigarrow s'$, więc $\langle \textbf{while } b \textbf{ do } S, s\rangle \rightsquigarrow s'$. OK

„Denotacyjna” semantyka instrukcji

$$\mathcal{S}_{\mathcal{OS}} : \textbf{Stmt} \to (\textbf{State} \rightharpoonup \textbf{State})$$

uzyskana z semantyki naturalnej bądź operacyjnej następująco:

$$\mathcal{S}_{\mathcal{OS}}[\![S]\!]\, s = s' \;\text{ wttw }\; \langle S, s\rangle \leadsto s' \quad (\text{ wttw }\; \langle S, s\rangle \Rightarrow^* s')$$

Na marginesie: TINY jest deterministyczny, więc rzeczywiście otrzymujemy funkcję

$$\mathcal{S}_{\mathcal{OS}}[\![S]\!] : \textbf{State} \rightharpoonup \textbf{State}$$

Jest to jednak na ogół funkcja *częściowa*.
Tak naprawdę definiujemy więc:

$$\mathcal{S}_{\mathcal{OS}}[\![S]\!]\, s = \begin{cases} s' & \text{jeśli } \langle S, s\rangle \leadsto s', \text{ tzn. } \langle S, s\rangle \Rightarrow^* s' \\ \text{undefined} & \text{jeśli } \langle S, s\rangle \not\leadsto \end{cases}$$

Semantyka operacyjna a semantyka naturalna

„Są one zupełnie inne"

Semantyka naturalna jest bardziej abstrakcyjna niż semantyka operacyjna

Istnieją naturalnie równoważne instrukcje o zupełnie różnych zbiorach obliczeń indukowanych przez semantykę operacyjną.

- Semantyka naturalna nie uwzględnia obliczeń, które „zakleszczają się" lub „zapętlają się".
- Semantyka naturalna nie podaje szczegółów dotyczących obliczeń.

Równoważność operacyjna

Instrukcje $S_1, S_2 \in$ **Stmt** są *równoważne operacyjnie* (równoważne względem semantyki operacyjnej)

$$S_1 \equiv_{\mathcal{OS}} S_2$$

jeśli dla wszystkich stanów $s \in$ **State**, $\langle S_1, s\rangle \approx \langle S_2, s\rangle$, przy czym:
konfiguracje $\gamma_1, \gamma_2 \in \Gamma$ są *bipodobne*, $\gamma_1 \approx \gamma_2$, jeśli:

- $\gamma_1 \Rightarrow^* s'$ wttw $\gamma_2 \Rightarrow^* s'$
- jeśli $\gamma_1 \Rightarrow \gamma_1'$ to $\gamma_2 \Rightarrow^* \gamma_2'$ z $\gamma_1' \approx \gamma_2'$
- jeśli $\gamma_2 \Rightarrow \gamma_2'$ to $\gamma_1 \Rightarrow^* \gamma_1'$ z $\gamma_1' \approx \gamma_2'$

Ważne pojęcie:
BISYMULACJA

Fakt:

Jeśli $S_1 \equiv_{\mathcal{OS}} S_2$ *to* $S_1 \equiv_{\mathcal{NS}} S_2$

Przykłady przytoczone przy okazji równoważności naturalnej przenoszą się także i tutaj. Dla dotychczas rozważanego języka równoważność naturalna jest tym samym co operacyjna.

Niedeterminizm i zakleszczanie obliczeń

Rozszerzmy instrukcje $\boxed{S \in \textbf{Stmt}}$ o:

$$S ::= \cdots \mid \textbf{abort} \mid S_1 \textbf{ or } S_2$$

Semantyka operacyjna

$$\langle S_1 \textbf{ or } S_2, s\rangle \Rightarrow \langle S_1, s\rangle \qquad \langle S_1 \textbf{ or } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$$

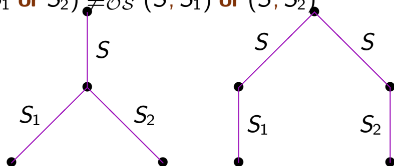
Semantyka naturalna

$$\frac{\langle S_1, s\rangle \rightsquigarrow s'}{\langle S_1 \textbf{ or } S_2, s\rangle \rightsquigarrow s'} \qquad \frac{\langle S_2, s\rangle \rightsquigarrow s'}{\langle S_1 \textbf{ or } S_2, s\rangle \rightsquigarrow s'}$$

Na marginesie: W obu przypadkach, **abort** powoduje zakleszczenie (przerwanie obliczenia)...

Co z równoważnościami?

- S_1 **or** $S_2 \equiv_{\mathcal{OS}} S_2$ **or** S_1
- **abort** $\equiv_{\mathcal{NS}}$ **while true do skip**
- **abort** $\not\equiv_{\mathcal{OS}}$ **while true do skip**
  (??? nie zawsze zachodzi przy innych wariantach bisymulacji)
- S **or abort** $\equiv_{\mathcal{NS}} S$ (*łaskawy niedeterminizm*)
- S **or abort** $\not\equiv_{\mathcal{OS}} S$ (chyba że $S \equiv_{\mathcal{OS}}$ **abort**)
- W ogólności, przy semantyce operacyjnej znaczenie ma moment niedeterministycznego wyboru: $S; (S_1$ **or** $S_2) \not\equiv_{\mathcal{OS}} (S; S_1)$ **or** $(S; S_2)$
- $S; (S_1$ **or** $S_2) \equiv_{\mathcal{NS}} (S; S_1)$ **or** $(S; S_2)$



Dodajmy „równoległość”

Rozszerzmy instrukcje $S \in \mathbf{Stmt}$ o złożenie „równoległe” (przeplot):

$S ::= \cdots \mid S_1 \mid\mid S_2$

Semantyka operacyjna

$$\langle S_1 \mid\mid S_2, s\rangle \Rightarrow \langle S_1' \mid\mid S_2, s'\rangle \quad \text{jeśli } \langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle$$
$$\langle S_1 \mid\mid S_2, s\rangle \Rightarrow \langle S_1 \mid\mid S_2', s'\rangle \quad \text{jeśli } \langle S_2, s\rangle \Rightarrow \langle S_2', s'\rangle$$

Sensownie

Semantyka naturalna

$$???\,\frac{\langle S_1, s\rangle \rightsquigarrow s' \quad \langle S_2, s'\rangle \rightsquigarrow s''}{\langle S_1 \textbf{ or } S_2, s\rangle \rightsquigarrow s''} \qquad \frac{\langle S_1, s'\rangle \rightsquigarrow s'' \quad \langle S_2, s\rangle \rightsquigarrow s'}{\langle S_1 \textbf{ or } S_2, s\rangle \rightsquigarrow s''}\,???$$

Bez sensu

Metoda

- zdefiniuj składnię (*dziedziny syntaktyczne*)
- zdefiniuj *dziedziny semantyczne*
- zdefiniuj *funkcje semantyczne*
- użyj *kompozycjonalnych* definicji

Dziedziny syntaktyczne

Każda kategora składniowa języka stanowi *dziedzinę syntaktyczną*, której elementami są wszystkich frazy składniowe tej kategorii.

Dziedziny semantyczne

Dziedziny semantyczne zawierają obiekty, które są zamierzonymi znaczeniami (*denotacjami)* fraz składniowych języka. Wszystkie denotacje pochodzą z dziedzin semantycznych, ale zazwyczaj nie wszystkie elementy dziedzin semantycznych są znaczeniami fraz języka.
Dziedziny semantyczne tworzy się z *dziedzin podstawowych* (**Int**, **Bool**) stosując *konstruktory dziedzin*: produkt, sumę (rozłączną), przestrzenie funkcyjne itd.
Każdej kluczowej kategorii składniowej odpowiada pewna dziedzina semantyczna.

Funkcje semantyczne

Dla każdej kategorii składniowej **Cat**, definiujemy *funkcję semantyczną*

$$\mathcal{C} \colon \mathbf{Cat} \to \mathbf{CAT}$$

która z każdą frazą składniową $ph \in \mathbf{Cat}$ związuje jej *denotację* w odpowiedniej dziedzinie semantycznej **CAT**:

$$\mathcal{C}[\![ph]\!] \in \mathbf{CAT}$$

Na marginesie: Wprowadza się w ten sposób także równoważność semantyczną: frazy $ph_1, ph_2 \in \mathbf{Cat}$ są *równoważne semantycznie* (równoważne względem semantyki denotacyjnej)

$$\boxed{ph_1 \equiv_{\mathcal{DS}} ph_2}$$

gdy $\mathcal{C}[\![ph_1]\!] = \mathcal{C}[\![ph_2]\!]$.

Kompozycjonalność

Funkcje semantyczne definiuje się *kompozycjonalnie* tak, aby denotacja frazy zależała jedynie od denotacji jej bezpośrednich składowych:

$$\mathcal{C}[\![\varphi(ph_1, \ldots, ph_n)]\!] = \Phi(\mathcal{C}[\![ph_1]\!], \ldots, \mathcal{C}[\![ph_n]\!])$$

Taką *klauzulę semantyczną* podaje się dla każdej konstrukcji składniowej.

Homomorfizm

Istotne konsekwencje:

INDUKCJA STRUKTURALNA

Własność kongruencji równoważności semantycznej

Dziedziny składniowe

Num	**(Var)**	**Exp**	**BExp**	**Stmt**

Trochę nieformalnie:

$$
\begin{array}{l}
N \in \mathbf{Num} ::= 0 \mid 1 \mid 2 \mid \cdots \\
(x \in \mathbf{Var} ::= \cdots) \\
e \in \mathbf{Exp} ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \\
b \in \mathbf{BExp} ::= \mathbf{true} \mid \mathbf{false} \mid e_1 \leqslant e_2 \mid \neg b' \mid b_1 \wedge b_2 \\
S \in \mathbf{Stmt} ::= x := e \mid \mathbf{skip} \mid S_1; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ b\ \mathbf{do}\ S'
\end{array}
$$

Semantyka denotacyjna języka Tiny

Dziedziny semantyczne

Int (**Bool**) (**State**) **EXP** **BEXP** **STMT**

$$\begin{aligned}
\mathbf{Int} &= \{0, 1, -1, 2, -2, \ldots\} \\
\mathbf{Bool} &= \{\mathbf{tt}, \mathbf{ff}\} \\
\mathbf{State} &= \mathbf{Var} \to \mathbf{Int} \\
\mathbf{EXP} &= \mathbf{State} \to \mathbf{Int} \\
\mathbf{BEXP} &= \mathbf{State} \to \mathbf{Bool} \\
\mathbf{STMT} &= \mathbf{State} \rightharpoonup \mathbf{State}
\end{aligned}$$

Funkcje semantyczne:

$$\begin{aligned}
\mathcal{N} &: \mathbf{Num} \to \mathbf{Int} \\
\mathcal{E} &: \mathbf{Exp} \to \mathbf{EXP} \\
\mathcal{B} &: \mathbf{BExp} \to \mathbf{BEXP} \\
\mathcal{S} &: \mathbf{Stmt} \to \mathbf{STMT}
\end{aligned}$$

Zanim przejdziemy dalej

(do klauzul semantycznych)

Pewne pojęcia pomocnicze:

- *λ-notacja*: $\lambda x{:}D.E$ oznacza funkcję, która odwzorowuje dowolny element $d \in D$ na $E[d/x]$
- *identyczność*: $id_D = \lambda x{:}D.x$
- *złożenie funkcji*: złożenie $f : D_1 \to D_2$ i $g : D_2 \to D_3$ zapisujemy jako $f;g : D_1 \to D_3$
- *warunek*: $ifte_D : \mathbf{Bool} \times D \times D \to D$ definiujemy jako

$$ifte_D(c, d_1, d_2) = \begin{cases} d_1 & \text{jeśli } c = \mathbf{tt} \\ d_2 & \text{jeśli } c = \mathbf{ff} \end{cases}$$

(będziemy często pomijać indeks D)

- *indeksowanie*: dla dowolnej funkcji $f: D_1 \times \cdots \times D_n \to D$ i dziedziny I,

$$lift^I(f): (I \to D_1) \times \cdots \times (I \to D_n) \to (I \to D)$$

definiujemy następująco:

$$lift^I(f)(fd_1, \ldots, fd_n) = \lambda i{:}I.f(fd_1(i), \ldots, fd_n(i))$$

Na przykład, warunek na funkcjach zależnych od stanu:

$$cond: \mathbf{BEXP} \times \mathbf{EXP} \times \mathbf{EXP} \to \mathbf{EXP}$$

który definiuje się *explicite* przez:

$$cond(B, E_1, E_2)(s) = ifte_{\mathbf{Int}}(B(s), E_1(s), E_2(s)) = \begin{cases} E_1(s) & \text{jeśli } B(s) = \mathbf{tt} \\ E_2(s) & \text{jeśli } B(s) = \mathbf{ff} \end{cases}$$

można zdefiniować jako $cond = lift^{\mathbf{State}}(ifte_{\mathbf{Int}})$.

Dotyczy to także funkcji częściowych

Klauzule semantyczne

$\mathcal{N}$: **Num** $\to$ **Int**

$\mathcal{N}[\![0]\!] = 0$
$\mathcal{N}[\![1]\!] = 1$
$\mathcal{N}[\![2]\!] = 2$
$\ldots$

$\mathcal{E}$: **Exp** $\to$ **EXP**

$\mathcal{E}[\![N]\!] = \lambda s{:}\mathbf{State}.\mathcal{N}[\![N]\!] \quad \mathcal{E}[\![x]\!] = \lambda s{:}\mathbf{State}.s\ x$
$\mathcal{E}[\![e_1 + e_2]\!] = \mathit{lift}^{\mathbf{State}}(+)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!])$
$\mathcal{E}[\![e_1 * e_2]\!] = \mathit{lift}^{\mathbf{State}}(*)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!])$
$\mathcal{E}[\![e_1 - e_2]\!] = \mathit{lift}^{\mathbf{State}}(-)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!])$

$\mathcal{B}$: **BExp** $\to$ **BEXP**

$\mathcal{B}[\![\mathbf{true}]\!] = \lambda s{:}\mathbf{State}.\mathbf{tt} \quad \mathcal{B}[\![\mathbf{false}]\!] = \lambda s{:}\mathbf{State}.\mathbf{ff}$
$\mathcal{B}[\![\neg b]\!] = \mathit{lift}^{\mathbf{State}}(\neg)(\mathcal{B}[\![b]\!]) \quad \mathcal{B}[\![e_1 \leqslant e_2]\!] = \mathit{lift}^{\mathbf{State}}(\leqslant)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!])$
$\mathcal{B}[\![b_1 \wedge b_2]\!] = \mathit{lift}^{\mathbf{State}}(\wedge)(\mathcal{B}[\![b_1]\!], \mathcal{B}[\![b_2]\!])$

Klauzule semantyczne

$\mathcal{S}: \textbf{Stmt} \to \textbf{STMT}$

$$
\begin{aligned}
\mathcal{S}[\![x := e]\!] &= \lambda s{:}\textbf{State}.s[x \mapsto \mathcal{E}[\![e]\!]\ s] \\
\mathcal{S}[\![\textbf{skip}]\!] &= id_{\textbf{State}} \\
\mathcal{S}[\![S_1; S_2]\!] &= \mathcal{S}[\![S_1]\!];\mathcal{S}[\![S_2]\!] \\
\mathcal{S}[\![\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2]\!] &= cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S_1]\!], \mathcal{S}[\![S_2]\!]) \\
\mathcal{S}[\![\textbf{while}\ b\ \textbf{do}\ S]\!] &= cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!];\mathcal{S}[\![\textbf{while}\ b\ \textbf{do}\ S]\!], id_{\textbf{State}})
\end{aligned}
$$

Coś nie tak?

Klauzula dla **while**:

$$\mathcal{S}[\![\textbf{while } b \textbf{ do } S]\!] = cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!];\mathcal{S}[\![\textbf{while } b \textbf{ do } S]\!], id_{\textbf{State}})$$

nie jest kompozycjonalna!
„Definiujemy":

$$??? \quad \mathcal{S}[\![\textbf{while } b \textbf{ do } S]\!] = \Phi(\ldots, \mathcal{S}[\![\textbf{while } b \textbf{ do } S]\!], \ldots) \quad ???$$

Niezbędne są *definicje stałopunktowe*

Potencjalne problemy z definicjami stałopunktowymi

Rozważmy definicje stałopunktowe w $\mathbf{STMT} = \mathbf{State} \rightharpoonup \mathbf{State}$, jak na przykład

$$\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!] = \Phi(\ldots, \mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!], \ldots)$$

- Czy zawsze istnieje punkt stały?

$$f = \lambda s{:}\mathbf{State}.\mathit{ifte}_{\mathbf{State}}(f(s)\ \text{nieokreślone}, s, f(s)[\mathit{var} \mapsto (f(s)\ \mathit{var}) + 1])$$

Można używać tylko niektórych Φ

- Czy jeśli istnieje punkt stały, to jest on jedyny?

$$f = \lambda s{:}\mathbf{State}.f(s)[\mathit{var} \mapsto 2 * (f(s)\ \mathit{var})]$$

(czy nawet: $f = \lambda s{:}\mathbf{State}.f(s)$)

Trzeba wybrać „najlepszy" punkt stały

Ku definicjom stałopunktowym

Przyjrzyjmy się klauzuli dla **while**:

$$\mathcal{S}[\![\textbf{while } b \textbf{ do } S]\!] = \Phi(\mathcal{S}[\![\textbf{while } b \textbf{ do } S]\!])$$

gdzie $\Phi\colon \mathbf{STMT} \to \mathbf{STMT}$ jest następujące:

$$\Phi(F) = cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!];F, id_{\mathbf{State}})$$

Musimy wybrać punkt stały, który będzie zgodny z naszymi intuicjami operacyjnymi. Potrzebujemy denotacji $fix(\Phi) \in \mathbf{STMT}$, która jest punktem stałym Φ (tak aby $\Phi(fix(\Phi)) = fix(\Phi)$) i która jest zgodna z semantyką operacyjną **while**, czyli takiej, aby

$$\langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow^* s' \quad \text{wttw} \quad fix(\Phi)\, s = s'$$

Właściwy wybór!

Przypuśmy, że taka zgodność semantyk zachodzi dla S, tzn.,
$\langle S, s\rangle \Rightarrow^* s'$ wttw $\mathcal{S}[\![S]\!]\, s = s'$.
Właściwy wybór:

$$\langle \textbf{while}\ b\ \textbf{do}\ S, s\rangle \Rightarrow^* s' \text{ wttw dla pewnego } n \geqslant 0, \Phi^n(\emptyset_{\textbf{State}\rightharpoonup\textbf{State}})\, s = s'$$

gdzie $\emptyset_{\textbf{State}\rightharpoonup\textbf{State}} : \textbf{State} \rightharpoonup \textbf{State}$ jest funkcją nigdzie niezdefiniowaną oraz
$\Phi^0(\emptyset_{\textbf{State}\rightharpoonup\textbf{State}}) = \emptyset_{\textbf{State}\rightharpoonup\textbf{State}}$ i $\Phi^{n+1}(\emptyset_{\textbf{State}\rightharpoonup\textbf{State}}) = \Phi(\Phi^n(\emptyset_{\textbf{State}\rightharpoonup\textbf{State}}))$.
Dowód: za chwilę.

Wniosek

$$\mathcal{S}[\![\textbf{while}\ b\ \textbf{do}\ S]\!] = \mathit{fix}(\Phi) = \bigcup_{n\geqslant 0} \Phi^n(\emptyset_{\textbf{State}\rightharpoonup\textbf{State}})$$

Jest to poprawna definicja dająca *najmniejszy* punkt stały Φ.

$$\textbf{while } \{sqr = (rt+1)^2 \wedge rt^2 \leqslant n\} \; sqr \leqslant n \textbf{ do } rt := rt + 1; sqr := sqr + 2 * rt + 1$$

$$\Phi(F) = cond(\mathcal{B}[\![sqr \leqslant n]\!], \mathcal{S}[\![rt := rt + 1; sqr := sqr + 2 * rt + 1]\!]; F, id_{\mathbf{State}})$$

$s(n, rt, sqr)$	$\Phi^0(\emptyset)(s)$	$\Phi^1(\emptyset)(s)$	$\Phi^2(\emptyset)(s)$	$\Phi^3(\emptyset)(s)$	$\Phi^4(\emptyset)(s)$	$\cdots$	$\bigcup \Phi^n(\emptyset)(s)$
$0, 0, 1$	?	$0, 0, 1$	$0, 0, 1$	$0, 0, 1$	$0, 0, 1$	$\cdots$	$0, 0, 1$
$1, 0, 1$	?	?	$1, 1, 4$	$1, 1, 4$	$1, 1, 4$	$\cdots$	$1, 1, 4$
$2, 0, 1$	?	?	$2, 1, 4$	$2, 1, 4$	$2, 1, 4$	$\cdots$	$2, 1, 4$
$3, 0, 1$	?	?	$3, 1, 4$	$3, 1, 4$	$3, 1, 4$	$\cdots$	$3, 1, 4$
$4, 0, 1$	?	?	?	$4, 2, 9$	$4, 2, 9$	$\cdots$	$4, 2, 9$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$8, 0, 1$	?	?	?	$8, 2, 9$	$8, 2, 9$	$\cdots$	$8, 2, 9$
$9, 0, 1$	?	?	?	?	$9, 3, 16$	$\cdots$	$9, 3, 16$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

$$\Phi(F) = cond(\mathcal{B}[\![sqr \leqslant n]\!], \mathcal{S}[\![rt := rt + 1; sqr := sqr + 2 * rt + 1]\!]; F, id_{\mathbf{State}})$$

$s(n, rt, sqr)$	$\Phi^0(\emptyset)(s)$	$\Phi^1(\emptyset)(s)$	$\Phi^2(\emptyset)(s)$	$\Phi^3(\emptyset)(s)$	$\Phi^4(\emptyset)(s)$	$\cdots$	$\bigcup \Phi^n(\emptyset)(s)$
0, 0, 1	?	0, 0, 1	0, 0, 1	0, 0, 1	0, 0, 1	$\cdots$	0, 0, 1
1, 0, 1	?	?	1, 1, 4	1, 1, 4	1, 1, 4	$\cdots$	1, 1, 4
1, 1, 4	?	1, 1, 4	1, 1, 4	1, 1, 4	1, 1, 4	$\cdots$	1, 1, 4
2, 0, 1	?	?	2, 1, 4	2, 1, 4	2, 1, 4	$\cdots$	2, 1, 4
2, 1, 4	?	2, 1, 4	2, 1, 4	2, 1, 4	2, 1, 4	$\cdots$	2, 1, 4
3, 0, 1	?	?	3, 1, 4	3, 1, 4	3, 1, 4	$\cdots$	3, 1, 4
3, 1, 4	?	3, 1, 4	3, 1, 4	3, 1, 4	3, 1, 4	$\cdots$	3, 1, 4
4, 0, 1	?	?	?	4, 2, 9	4, 2, 9	$\cdots$	4, 2, 9
4, 1, 4	?	?	4, 2, 9	4, 2, 9	4, 2, 9	$\cdots$	4, 2, 9
4, 2, 9	?	4, 2, 9	4, 2, 9	4, 2, 9	4, 2, 9	$\cdots$	4, 2, 9
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
9, 0, 1	?	?	?	?	9, 3, 16	$\cdots$	9, 3, 16
9, 1, 4	?	?	?	9, 3, 16	9, 3, 16	$\cdots$	9, 3, 16
9, 2, 9	?	?	9, 3, 16	9, 3, 16	9, 3, 16	$\cdots$	9, 3, 16
9, 3, 16	?	9, 3, 16	9, 3, 16	9, 3, 16	9, 3, 16	$\cdots$	9, 3, 16
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Dowód

„$\Longrightarrow$": Przez indukcję po długości obliczenia $\langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow^k s'$.

$k > 0$: Wówczas $\langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow \gamma \Rightarrow^{k-1} s'$. W zależności od pierwszego kroku:

- $\mathcal{B}[\![b]\!]\ s = \mathbf{ff}$ i $\gamma = s$. Wtedy $s' = s$ i $\Phi(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ s = s$. OK
- $\mathcal{B}[\![b]\!]\ s = \mathbf{tt}$ oraz $\gamma = \langle S; \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow^{k-1} s'$. Wtedy $\langle S, s\rangle \Rightarrow^{k_1} \hat{s}$ i $\langle \textbf{while } b \textbf{ do } S, \hat{s}\rangle \Rightarrow^{k_2} s'$, dla pewnego $\hat{s} \in \mathbf{State}$ i $k_1, k_2 > 0$ z $k_1 + k_2 = k - 1$. Zatem, $\mathcal{S}[\![S]\!]\ s = \hat{s}$ oraz $\Phi^n(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ \hat{s} = s'$ dla pewnego $n \geqslant 0$. Zatem, $\Phi^{n+1}(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ s = s'$. OK

Na marginesie: Dowód wymaga jedynie $\boxed{\langle S, s\rangle \Rightarrow^* s' \Longrightarrow \mathcal{S}[\![S]\!]\ s = s'}$

Dowód

„ $\Longleftarrow$": Przez indukcję po $n \geqslant 0$ przyjmując, że $\Phi^n(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ s = s'$.

$n > 0$: Wówczas

$$\Phi^n(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ s = cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!];\Phi^{n-1}(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}}), id_{\mathbf{State}})\ s.$$

- $\mathcal{B}[\![b]\!]\ s = \mathbf{ff}$: wtedy $\Phi^n(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ s = s$, więc $s' = s$, a także $\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow s$. OK
- $\mathcal{B}[\![b]\!]\ s = \mathbf{tt}$: wówczas $\Phi^n(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ s = \Phi^{n-1}(\emptyset_{\mathbf{State} \rightharpoonup \mathbf{State}})\ (\mathcal{S}[\![S]\!]\ s)$. Zatem, $\langle \mathbf{while}\ b\ \mathbf{do}\ S, \mathcal{S}[\![S]\!]\ s\rangle \Rightarrow^* s'$, a ponieważ $\langle S, s\rangle \Rightarrow^* (\mathcal{S}[\![S]\!]\ s)$, otrzymujemy $\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow \langle S; \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow^* \langle \mathbf{while}\ b\ \mathbf{do}\ S, \mathcal{S}[\![S]\!]\ s\rangle \Rightarrow^* s'$. OK

Na marginesie: Dowód wymaga jedynie $\boxed{\langle S, s\rangle \Rightarrow^* s' \Longleftarrow \mathcal{S}[\![S]\!]\ s = s'}$

Zgodność semantyki denotacyjnej

Twierdzenie:

Dla każdej instrukcji $S \in$ **Stmt** *i stanów* $s, s' \in$ **State***,*

$$\mathcal{S}[\![S]\!]\, s = s' \quad \text{wttw} \quad \langle S, s\rangle \Rightarrow^* s'$$

Dowód:

„$\Longrightarrow$": Przez indukcję strukturalną po S.

„$\Longleftarrow$": Przez indukcję po długości obliczenia $\langle S, s\rangle \Rightarrow^* s'$.

Przyjrzyjmy się teraz deklaracjom zmiennych:

TINY$^+$

$$S \in \textbf{Stmt} ::= \ldots \mid \textbf{begin}\ D_V\ S\ \textbf{end}$$
$$D_V \in \textbf{VDecl} ::= \textbf{var}\ x;\, D_V \mid \varepsilon$$

Lokacje

Zmienne odgrywają dwie role:

- identyfikatorów — tak używa się ich w programach
- nazw komórek pamięci, w których przechowuje się wartości

Aby te role oddzielić, należy zmienić strukturę dziedzin semantycznych.

Rozdzielamy stany na *środowiska* i *składy*

$$\begin{aligned}
\mathbf{Int} &= \{0, 1, -1, 2, -2, \ldots\}_\perp \\
\mathbf{Bool} &= \{\mathbf{tt}, \mathbf{ff}\}_\perp \\
\mathbf{VEnv} &= \mathbf{Var} \to (\mathbf{Loc} + \{??\}) \\
\mathbf{Store} &= \mathbf{Loc} \to (\mathbf{Int} + \{??\})
\end{aligned}$$

$$\begin{aligned}
&\mathit{combine} : \mathbf{VEnv} \times \mathbf{Store} \to \mathbf{State} \\
&\mathit{combine}(\rho_V, s) = \lambda x{:}\mathbf{Var}.s(\rho_V(x))
\end{aligned}$$

Niedokładnie, ale w zasadzie dobrze

Funkcje semantyczne

$$\mathcal{N} : \mathbf{Num} \to \mathbf{Int}$$
$$\mathcal{E} : \mathbf{Exp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Store} \to (\mathbf{Int} + \{??\})}_{\mathbf{EXP}}$$
$$\mathcal{B} : \mathbf{BExp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Store} \to (\mathbf{Bool} + \{??\})}_{\mathbf{BEXP}}$$

a następnie na przykład

$$\mathcal{E}[\![x]\!] = \lambda\rho_V{:}\mathbf{VEnv}.\lambda s{:}\mathbf{Store}.\mathit{ifte}(\rho_V\ x = ??, ??, \mathit{ifte}(s\ (\rho_V\ x) = ??, ??, s\ (\rho_V\ x)))$$
$$\mathcal{E}[\![e_1 + e_2]\!] = \lambda\rho_V{:}\mathbf{VEnv}.\lambda s{:}\mathbf{Store}.\mathit{ifte}(\mathcal{E}[\![e_1]\!]\ \rho_V\ s = ??, ??, \mathit{ifte}(\mathcal{E}[\![e_2]\!]\ \rho_V\ s = ??, ??, \mathcal{E}[\![e_1]\!]\ \rho_V\ s + \mathcal{E}[\![e_2]\!]\ \rho_V\ s))$$

Wygląda strasznie!
A czyta się to jeszcze gorzej!

Nieco notacji

- usuwamy (przenosimy) lambda abstrakcje
- stosujemy notację z **where**, **let**, jawnymi wyrażeniami **if-then** itp.
- zakładamy, że błędy ?? propagują się

Wówczas:

$$\mathcal{E}[\![x]\!]\ \rho_V\ s = s\ l \text{ where } l = \rho_V\ x$$
$$\mathcal{E}[\![e_1 + e_2]\!]\ \rho_V\ s = n_1 + n_2 \text{ where } n_1 = \mathcal{E}[\![e_1]\!]\ \rho_V\ s,\ n_2 = \mathcal{E}[\![e_2]\!]\ \rho_V\ s$$

Stosując combine, opisz związek z poprzednią semantyką

Zapisz wszystkie pozostałe klauzule dla $\mathcal{E}$ i $\mathcal{B}$.
Wyraź ich dokładne znaczenie
rozwijając wszystkie użyte pojęcia.

Instrukcje

Można pracować z „dużymi stanami"

$$\mathcal{S} : \textbf{Stmt} \to \underbrace{\textbf{VEnv} \times \textbf{Store} \to (\textbf{VEnv} \times \textbf{Store} + \{??\})}_{\textbf{STMT}}$$

ALE:

Instrukcje nie zmieniają środowiska!

Zatem:

$$\mathcal{S} : \textbf{Stmt} \to \underbrace{\textbf{VEnv} \to \textbf{Store} \to (\textbf{Store} + \{??\})}_{\textbf{STMT}}$$

Klauzule semantyczne

$$\mathcal{S}[\![x := e]\!]\ \rho_V\ s = s[l \mapsto n] \text{ where } l = \rho_V\ x, n = \mathcal{E}[\![e]\!]\ \rho_V\ s$$

$$\mathcal{S}[\![\mathbf{skip}]\!]\ \rho_V\ s = s$$

$$\mathcal{S}[\![S_1; S_2]\!]\ \rho_V\ s = \mathcal{S}[\![S_2]\!]\ \rho_V\ s_1 \text{ where } s_1 = \mathcal{S}[\![S_1]\!]\ \rho_V\ s$$

$$\begin{aligned}\mathcal{S}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\ \rho_V\ s = {} & \text{let } v = \mathcal{B}[\![b]\!]\ \rho_V\ s \text{ in} \\ & \text{if } v = \mathbf{tt} \text{ then } \mathcal{S}[\![S_1]\!]\ \rho_V\ s \\ & \text{if } v = \mathbf{ff} \text{ then } \mathcal{S}[\![S_2]\!]\ \rho_V\ s\end{aligned}$$

$$\begin{aligned}\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ s = {} & \text{let } v = \mathcal{B}[\![b]\!]\ \rho_V\ s \text{ in} \\ & \text{if } v = \mathbf{ff} \text{ then } s \\ & \text{if } v = \mathbf{tt} \text{ then } \mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ s' \\ & \qquad \text{where } s' = \mathcal{S}[\![S]\!]\ \rho_V\ s\end{aligned}$$

Stosując combine*, opisz związek z poprzednią semantyką*

Bardziej zwarta wersja

Przy założeniu propagacji błędów ?? przy składaniu funkcji ze **Store** w **Store** $+ \{??\}$:

$$
\begin{aligned}
&\mathcal{S}[\![x := e]\!]\ \rho_V\ s = s[l \mapsto n] \text{ where } l = \rho_V\ x, n = \mathcal{E}[\![e]\!]\ \rho_V\ s \\
&\mathcal{S}[\![\textbf{skip}]\!]\ \rho_V = id_{\textbf{Store}} \\
&\mathcal{S}[\![S_1; S_2]\!]\ \rho_V = \mathcal{S}[\![S_1]\!]\ \rho_V ; \mathcal{S}[\![S_2]\!]\ \rho_V \\
&\mathcal{S}[\![\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2]\!]\ \rho_V = \\
&\qquad cond(\mathcal{B}[\![b]\!]\ \rho_V, \mathcal{S}[\![S_1]\!]\ \rho_V, \mathcal{S}[\![S_2]\!]\ \rho_V) \\
&\mathcal{S}[\![\textbf{while}\ b\ \textbf{do}\ S]\!]\ \rho_V = \\
&\qquad cond(\mathcal{B}[\![b]\!]\ \rho_V, \mathcal{S}[\![S]\!]\ \rho_V ; \mathcal{S}[\![\textbf{while}\ b\ \textbf{do}\ S]\!]\ \rho_V, id_{\textbf{Store}})
\end{aligned}
$$

Brakująca klauzula dla bloków za chwilę

Deklaracje zmieniają środowiska

$$\mathcal{D}_V : \textbf{VDecl} \to \underbrace{\textbf{VEnv} \to \textbf{Store} \to (\textbf{VEnv} \times \textbf{Store} + \{??\})}_{\textbf{VDECL}}$$

$$\mathcal{D}_V[\![\varepsilon]\!]\ \rho_V\ s = \langle \rho_V, s \rangle$$
$$\mathcal{D}_V[\![\textbf{var}\ x; D_V]\!]\ \rho_V\ s = \mathcal{D}_V[\![D_V]\!]\ \rho'_V\ s'$$
$$\text{where } l = newloc(s),$$
$$\rho'_V = \rho_V[x \mapsto l], s' = s[l \mapsto ??]$$

Problem: Chcemy, aby $newloc : \textbf{Store} \to \textbf{Loc}$ dawała w wyniku nową, „wolną” lokację. Nie da się tego zdefiniować za pomocą dotychczasowych pojęć. Rozwiązanie: potrzebujemy więcej informacji w składach, za pomocą których określimy używane i wolne lokacje.

Proste rozwiązanie

Przyjmijmy:

$$\mathbf{Loc} = \{0, 1, 2, \ldots\}$$

Do każdego składu wprowadzamy wskaźnik na kolejną nieużywaną lokację:

$$\mathbf{Store} = (\mathbf{Loc} + \{next\}) \to (\mathbf{Int} + \{??\})$$

Klauzule semantyczne wyglądają teraz tak:

$$
\begin{aligned}
&\mathcal{D}_V[\![\varepsilon]\!]\ \rho_V\ s = \langle \rho_V, s \rangle \\
&\mathcal{D}_V[\![\mathbf{var}\ x;\ D_V]\!]\ \rho_V\ s = \mathcal{D}_V[\![D_V]\!]\ \rho'_V\ s' \\
&\qquad\qquad\qquad\qquad \text{where } l = s\ next, \rho'_V = \rho_V[x \mapsto l], \\
&\qquad\qquad\qquad\qquad\qquad s' = s[l \mapsto ??, next \mapsto l + 1]
\end{aligned}
$$

Semantyka bloków

$$\mathcal{S}[\![\textbf{begin } D_V \ S \textbf{ end}]\!] \ \rho_V \ s = \mathcal{S}[\![S]\!] \ \rho'_V \ s' \text{ where } \langle \rho'_V, s' \rangle = \mathcal{D}_V[\![D_V]\!] \ \rho_V \ s$$

Zakresem deklaracji jest blok, w którym ona występuje z wyjątkiem bloków, w których ta sama zmienna jest ponownie deklarowana

Na przykład zmienne w programie

$$\textbf{begin var } y; \textbf{var } x \ x := 1; \textbf{begin var } x \ y := 2; x := 5 \textbf{ end}; y := x \textbf{ end}$$

można w następujący sposób związać z ich deklaracjami:

$$\textbf{begin} \ \boxed{\textbf{var } y}; \boxed{\textbf{var } x} \ \boxed{x} := 1; \textbf{begin} \ \boxed{\textbf{var } x} \ \boxed{y} := 2; \boxed{x} := 5 \textbf{ end}; \boxed{y} := \boxed{x} \textbf{ end}$$

Procedury

TINY^{++}

$$
\begin{aligned}
S \in \textbf{Stmt} &::= \ldots \mid \textbf{begin}\ D_V\ D_P\ S\ \textbf{end} \mid \textbf{call}\ p \\
D_V \in \textbf{VDecl} &::= \textbf{var}\ x; D_V \mid \varepsilon \\
D_P \in \textbf{PDecl} &::= \textbf{proc}\ p\ \textbf{is}\ (S); D_P \mid \varepsilon
\end{aligned}
$$

- wiązanie zmiennych globalnych
- rekurencja

Wiązanie zmiennych globalnych

Wiązanie statyczne

```
begin var y;
      var x;
      proc p is (x := 1);
      begin var x;
            x := 3;
            call p;
            y := x
      end
end
```

Wiązanie dynamiczne

```
begin var y;
      var x;
      proc p is (x := 1);
      begin var x;
            x := 3;
            call p;   %%% z x
            y := x
      end
end
```

Wiązanie dynamiczne

$$\mathbf{PEnv} = \mathbf{IDE} \to (\mathbf{PROC_0} + \{??\})$$
$$\mathbf{PROC_0} = \mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Store} \to (\mathbf{Store} + \{??\})$$

$$\mathcal{S} : \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Store} \to (\mathbf{Store} + \{??\})}_{\mathbf{STMT}}$$

$$\mathcal{D}_P : \mathbf{PDecl} \to \underbrace{\mathbf{PEnv} \to (\mathbf{PEnv} + \{??\})}_{\mathbf{PDECL}}$$

Klauzule semantyczne

$\mathcal{S}[\![x := e]\!]\ \rho_V\ \rho_P\ s = s[l \mapsto n]$ where $l = \rho_V\ x, n = \mathcal{E}[\![e]\!]\ \rho_V\ s$

$\mathcal{S}[\![\mathbf{skip}]\!]\ \rho_V\ \rho_P = id_{\mathbf{Store}}$

$\mathcal{S}[\![S_1; S_2]\!]\ \rho_V\ \rho_P = \mathcal{S}[\![S_1]\!]\ \rho_V\ \rho_P ; \mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P$

$\mathcal{S}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\ \rho_V\ \rho_P =$
$\quad cond(\mathcal{B}[\![b]\!]\ \rho_V, \mathcal{S}[\![S_1]\!]\ \rho_V\ \rho_P, \mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P)$

$\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ \rho_P =$
$\quad cond(\mathcal{B}[\![b]\!]\ \rho_V, \mathcal{S}[\![S]\!]\ \rho_V\ \rho_P ; \mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ \rho_P, id_{\mathbf{Store}})$

$\mathcal{S}[\![\mathbf{call}\ p]\!]\ \rho_V\ \rho_P = P\ \rho_V\ \rho_P$ where $P = \rho_P\ p$

$\mathcal{S}[\![\mathbf{begin}\ D_V\ D_P\ S\ \mathbf{end}]\!]\ \rho_V\ \rho_P\ s =$
$\quad \mathcal{S}[\![S]\!]\ \rho'_V\ \rho'_P\ s'$ where $\langle \rho'_V, s' \rangle = \mathcal{D}_V[\![D_V]\!]\ \rho_V\ s,\ \ \rho'_P = \mathcal{D}_P[\![D_P]\!]\ \rho_P$

$\mathcal{D}_P[\![\varepsilon]\!] = id_{\mathbf{PEnv}}$

$\mathcal{D}_P[\![\mathbf{proc}\ p\ \mathbf{is}\ (S); D_P]\!]\ \rho_P = \mathcal{D}_P[\![D_P]\!]\ \rho_P[p \mapsto \mathcal{S}[\![S]\!]]$

begin var x;
    proc NO **is** (**if** $101 \leqslant x$ **then** $x := x - 10$
        else ($x := x + 11$; **call** NO; **call** NO));
    $x := 1543345$;
    call NO
end

Wiązanie statyczne

$$\mathbf{PEnv} = \mathbf{IDE} \to (\mathbf{PROC_0} + \{??\})$$
$$\mathbf{PROC_0} = \mathbf{Store} \to (\mathbf{Store} + \{??\})$$

$$\mathcal{S} : \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Store} \to (\mathbf{Store} + \{??\})}_{\mathbf{STMT}}$$
$$\mathcal{D}_P : \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to (\mathbf{PEnv} + \{??\})}_{\mathbf{PDECL}}$$

Klauzule semantyczne

$$\mathcal{S}[\![x := e]\!]\ \rho_V\ \rho_P\ s = s[l \mapsto n] \text{ where } l = \rho_V\ x, n = \mathcal{E}[\![e]\!]\ \rho_V\ s$$

$$\mathcal{S}[\![\textbf{skip}]\!]\ \rho_V\ \rho_P = id_{\textbf{Store}}$$

$$\mathcal{S}[\![S_1; S_2]\!]\ \rho_V\ \rho_P = \mathcal{S}[\![S_1]\!]\ \rho_V\ \rho_P;\mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P$$

$$\mathcal{S}[\![\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2]\!]\ \rho_V\ \rho_P = cond(\mathcal{B}[\![b]\!]\ \rho_V, \mathcal{S}[\![S_1]\!]\ \rho_V\ \rho_P, \mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P)$$

$$\mathcal{S}[\![\textbf{while}\ b\ \textbf{do}\ S]\!]\ \rho_V\ \rho_P = cond(\mathcal{B}[\![b]\!]\ \rho_V, \mathcal{S}[\![S]\!]\ \rho_V\ \rho_P;\mathcal{S}[\![\textbf{while}\ b\ \textbf{do}\ S]\!]\ \rho_V\ \rho_P, id_{\textbf{Store}})$$

$$\mathcal{S}[\![\textbf{call}\ p]\!]\ \rho_V\ \rho_P = P \text{ where } P = \rho_P\ p$$

$$\mathcal{S}[\![\textbf{begin}\ D_V\ D_P\ S\ \textbf{end}]\!]\ \rho_V\ \rho_P\ s = \mathcal{S}[\![S]\!]\ \rho'_V\ \rho'_P\ s' \text{ where } \langle \rho'_V, s' \rangle = \mathcal{D}_V[\![D_V]\!]\ \rho_V\ s,\ \rho'_P = \mathcal{D}_P[\![D_P]\!]\ \rho_V\ \rho_P$$

$$\mathcal{D}_P[\![\varepsilon]\!]\ \rho_V = id_{\textbf{PEnv}}$$

$$\mathcal{D}_P[\![\textbf{proc}\ p\ \textbf{is}\ (S); D_P]\!]\ \rho_V\ \rho_P = \mathcal{D}_P[\![D_P]\!]\ \rho_V\ \rho_P[p \mapsto P] \text{ where } P = \mathcal{S}[\![S]\!]\ \rho_V\ \rho_P[p \mapsto P]$$

Parametry

Przekazywanie parametrów:

- przez wartość
- przez zmienną
- przez nazwę

Zakładamy **wiązanie statyczne**

$$
\begin{aligned}
S \in \textbf{Stmt} &::= \ldots \mid \textbf{begin}\ D_V\ D_P\ S\ \textbf{end} \\
&\quad \mid \textbf{call}\ p \mid \textbf{call}\ p(\textbf{vr}\ x) \mid \textbf{call}\ p(\textbf{vl}\ e) \mid \textbf{call}\ p(\textbf{nm}\ e) \\
D_V \in \textbf{VDecl} &::= \textbf{var}\ x; D_V \mid \varepsilon \\
D_P \in \textbf{PDecl} &::= \textbf{proc}\ p\ \textbf{is}\ (S); D_P \mid \textbf{proc}\ p(\textbf{vr}\ x)\ \textbf{is}\ (S); D_P \\
&\quad \mid \textbf{proc}\ p(\textbf{vl}\ x)\ \textbf{is}\ (S); D_P \mid \textbf{proc}\ p(\textbf{nm}\ x)\ \textbf{is}\ (S); D_P \\
&\quad \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{PEnv} &= \mathbf{IDE} \to (\mathbf{PROC}_0 + \mathbf{PROC}_1^{vr} + \mathbf{PROC}_1^{vl} + \mathbf{PROC}_1^{nm} + \{??\}) \\
\mathbf{PROC}_0 &= \mathbf{Store} \to (\mathbf{Store} + \{??\}) \\
\mathbf{PROC}_1^{vr} &= \mathbf{Loc} \to \mathbf{PROC}_0 \\
\mathbf{PROC}_1^{vl} &= \mathbf{Int} \to \mathbf{PROC}_0 \\
\mathbf{PROC}_1^{nm} &= (\mathbf{Store} \to (\mathbf{Int} + \{??\})) \to \mathbf{PROC}_0
\end{aligned}
$$

Funkcje semantyczne

Jak poprzednio:

$$\mathcal{S} : \textbf{Stmt} \to \underbrace{\textbf{VEnv} \to \textbf{PEnv} \to \textbf{Store} \to (\textbf{Store} + \{??\})}_{\textbf{STMT}}$$

$$\mathcal{D}_P : \textbf{PDecl} \to \underbrace{\textbf{VEnv} \to \textbf{PEnv} \to (\textbf{PEnv} + \{??\})}_{\textbf{PDECL}}$$

Klazule semantyczne

bez parametrów

$\mathcal{S}[\![\textbf{call } p]\!]\ \rho_V\ \rho_P = P \text{ where } P = \rho_P\ p$

$\mathcal{D}_P[\![\textbf{proc } p \textbf{ is } (S); D_P]\!]\ \rho_V\ \rho_P =$

$\quad \mathcal{D}_P[\![D_P]\!]\ \rho_V\ \rho_P[p \mapsto P] \text{ where } P = \mathcal{S}[\![S]\!]\ \rho_V\ \rho_P[p \mapsto P]$

Parametr przekazywany przez zmienną

$$\mathcal{S}[\![\textbf{call } p(\textbf{vr } x)]\!]\ \rho_V\ \rho_P = P\ l \text{ where } P = \rho_P\ p \in \textbf{PROC}_1^{\text{vr}},\ \ l = \rho_V\ x$$

$$\mathcal{D}_P[\![\textbf{proc } p(\textbf{vr } x) \textbf{ is } (S); D_P]\!]\ \rho_V\ \rho_P =$$
$$\qquad \mathcal{D}_P[\![D_P]\!]\ \rho_V\ \rho_P[p \mapsto P] \text{ where } P\ l = \mathcal{S}[\![S]\!]\ \rho_V[x \mapsto l]\ \rho_P[p \mapsto P]$$

Parametr przekazywany przez wartość

$$
\begin{array}{l}
\mathcal{S}[\![\textbf{call}\ p(\textbf{vl}\ e)]\!]\ \rho_V\ \rho_P\ s = \\
\quad P\ n\ s\ \text{where}\ P = \rho_P\ p \in \mathbf{PROC}_1^{\text{vl}},\ \ n = \mathcal{E}[\![e]\!]\ \rho_V\ s \\
\mathcal{D}_P[\![\textbf{proc}\ p(\textbf{vl}\ x)\ \textbf{is}\ (S);\ D_P]\!]\ \rho_V\ \rho_P = \\
\quad \mathcal{D}_P[\![D_P]\!]\ \rho_V\ \rho_P[p \mapsto P]\ \text{where} \\
\qquad P\ n\ s = \mathcal{S}[\![S]\!]\ \rho_V'\ \rho_P[p \mapsto P]\ s'\ \text{where} \\
\qquad\quad l = s\ next,\ \ \rho_V' = \rho_V[x \mapsto l],\ \ s' = s[l \mapsto n, next \mapsto l+1]
\end{array}
$$

Parametr przekazywany przez nazwę

$\mathcal{S}[\![\textbf{call } p(\textbf{nm } e)]\!]\ \rho_V\ \rho_P = P\ (\mathcal{E}[\![e]\!]\ \rho_V)$ where $P = \rho_P\ p \in \textbf{PROC}_1^{nm}$

$\mathcal{D}_P[\![\textbf{proc } p(\textbf{nm } x) \textbf{ is } (S); D_P]\!]\ \rho_V\ \rho_P =$

$\qquad \mathcal{D}_P[\![D_P]\!]\ \rho_V\ \rho_P[p \mapsto P]$ where $P\ E = \mathcal{S}[\![S]\!]\ \rho_V[x \mapsto E]\ \rho_P[p \mapsto P]$

OOOPS! $\rho_V[x \mapsto E] \notin \textbf{VEnv}$ *Trzeba poprawić!*

$\textbf{VEnv} = \textbf{Var} \to (\textbf{Loc} + (\textbf{Store} \to (\textbf{Int} + \{??\})) + \{??\})$

$\mathcal{E}[\![x]\!]\ \rho_V\ s =$ let $v = \rho_V\ x$ in if $v \in \textbf{Loc}$ then $s\ v$
if $v \in (\textbf{Store} \to (\textbf{Int} + \{??\}))$ then $v\ s$

Opisuje to wyliczanie parametrów przekazywanych przez nazwę,
ale nie przypisania na zmienne przekazywane w taki sposób

Wejście/wyjście

TINY^{+++}

$S \in \textbf{Stmt} ::= \ldots \mid \textbf{read}\ x \mid \textbf{write}\ e$

Dziedziny semantyczne

$$\begin{aligned}
\mathbf{Stream} &= \mathbf{Int} \times \mathbf{Stream} + \{\mathbf{eof}\} \\
\mathbf{Input} &= \mathbf{Stream} \\
\mathbf{Output} &= \mathbf{Stream} \\
\mathbf{State} &= \mathbf{Store} \times \mathbf{Input} \times \mathbf{Output}
\end{aligned}$$

Właściwie:

$$\mathbf{Stream} = (\mathbf{Int} \otimes_L \mathbf{Stream}) \oplus \{\mathbf{eof}\}_\perp$$

gdzie:

$$\mathbf{D} \otimes_L \mathbf{D}' = \mathbf{D} \otimes \mathbf{D}'_\perp$$

Funkcje semantyczne

$$\mathcal{E} : \textbf{Exp} \to \underbrace{\textbf{VEnv} \to \textbf{State} \to (\textbf{Int} + \{??\})}_{\textbf{EXP}}$$

$$\mathcal{B} : \textbf{BExp} \to \underbrace{\textbf{VEnv} \to \textbf{State} \to (\textbf{Bool} + \{??\})}_{\textbf{BEXP}}$$

Trzeba zmienić tylko jedną klauzulę:

$$\mathcal{E}[\![x]\!]\ \rho_V\ \langle s, i, o\rangle = s\ l \text{ where } l = \rho_V\ x$$

Semantyka instrukcji

$$\mathcal{S} : \textbf{Stmt} \to \underbrace{\textbf{VEnv} \to \textbf{PEnv} \to \textbf{State} \to (\textbf{State} + \{??\})}_{\textbf{STMT}}$$

I znów tylko jedna klauzula wymaga zmian:

$$\mathcal{S}[\![x := e]\!]\ \rho_V\ \rho_P\ \langle s, i, o\rangle = \langle s[l \mapsto n], i, o\rangle$$
$$\text{where } l = \rho_V\ x, n = \mathcal{E}[\![e]\!]\ \rho_V\ \langle s, i, o\rangle$$

(plus podobna zmiana w $\mathcal{D}_V[\![\textbf{var}\ x; D_V]\!]\ \ldots = \ldots$) i dwie klauzule do dodania:

$$\mathcal{S}[\![\textbf{read}\ x]\!]\ \rho_V\ \rho_P\ \langle s, i, o\rangle = \langle s[l \mapsto n], i', o\rangle \text{ where } l = \rho_V\ x, \langle n, i'\rangle = i$$
$$\mathcal{S}[\![\textbf{write}\ e]\!]\ \rho_V\ \rho_P\ \langle s, i, o\rangle = \langle s, i, \langle n, o\rangle\rangle \text{ where } n = \mathcal{E}[\![e]\!]\ \rho_V\ \langle s, i, o\rangle$$

Przyjmujemy, że $\boxed{\langle n, i'\rangle = i}$ generuje ?? gdy $i = \textbf{eof}$

Programy

Nowa dziedzina składniowa:

$$\textbf{Prog} ::= \textbf{prog}\ S$$

z oczywistą funkcją semantyczną:

$$\mathcal{P} : \textbf{Prog} \to \underbrace{\textbf{Input} \to (\textbf{Output} + \{??\})}_{\textbf{PROG}}$$

określoną następująco:

$$\mathcal{P}[\![\textbf{prog}\ S]\!]\ i = o' \text{ where } \mathcal{S}[\![S]\!]\ \rho_V^{\emptyset}\ \rho_P^{\emptyset}\ \langle s^{\emptyset}, i, \textbf{eof}\rangle = \langle s', i', o'\rangle,$$
$$\rho_V^{\emptyset}\ x = ??, \rho_P^{\emptyset}\ p = ??, s^{\emptyset}\ next = 0, s^{\emptyset}\ l = ??$$

Zmiana filozofii

Czy chcemy dopuścić, by instrukcje zerowały wyjście?

Z: Co się dzieje teraz?

Na: Jaki będzie wynik końcowy?

Semantyka bezpośrednia a semantyka kontynuacyjna

begin ... ; ... ; ... **end**

$s^{\emptyset} \xrightarrow{\mathcal{S}[\![\ldots]\!]} s_i \xrightarrow{\mathcal{S}[\![\ldots]\!]} s_j \xrightarrow{\mathcal{S}[\![\ldots]\!]} s' \rightsquigarrow$ „wynik końcowy"

begin ... ; ... ; ... **end**

$\kappa' : \rightsquigarrow$ „wynik końcowy"

$\xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa_i : \rightsquigarrow$ „wynik końcowy"

$\xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa_j : \rightsquigarrow$ „wynik końcowy"

$\xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa^{\emptyset} : \rightsquigarrow$ „wynik końcowy"

Kontynuacje

$$\boxed{\textbf{Cont} = \textbf{State} \to \textbf{Ans}}$$

Teraz:

- stany nie zawierają wyjścia
- odpowiedzi są wyjściem (lub błędami)
- tak wyglądają kontynuacje instrukcji; semantykę instrukcji definiuje się tak:

$$\mathcal{S} : \textbf{Stmt} \to \underbrace{\textbf{VEnv} \to \textbf{PEnv} \to \textbf{Cont} \to \textbf{Cont}}_{\textbf{STMT}}$$

Czyli: $\boxed{\textbf{STMT} = \textbf{VEnv} \to \textbf{PEnv} \to \textbf{Cont} \to \textbf{State} \to \textbf{Ans}}$

Kontynuacje wyrażeń i deklaracji

- kontynuacje dla innych kategorii składniowych mogą być dodatkowo parametryzowane:
  - wyrażenia przekazują wartości, więc

$$\begin{aligned}\mathbf{Cont_E} &= \mathbf{Int} \to \mathbf{State} \to \mathbf{Ans}\\ \mathbf{Cont_B} &= \mathbf{Bool} \to \mathbf{State} \to \mathbf{Ans}\end{aligned}$$

  - deklaracje przekazują środowiska, więc

$$\begin{aligned}\mathbf{Cont_{D_V}} &= \mathbf{VEnv} \to \mathbf{State} \to \mathbf{Ans}\\ \mathbf{Cont_{D_P}} &= \mathbf{PEnv} \to \mathbf{State} \to \mathbf{Ans}\end{aligned}$$

Tiny^{+++}

$$
\begin{aligned}
N \in \textbf{Num} &::= 0 \mid 1 \mid 2 \mid \cdots \\
x \in \textbf{Var} &::= \cdots \\
p \in \textbf{IDE} &::= \cdots \\
e \in \textbf{Exp} &::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \\
b \in \textbf{BExp} &::= \textbf{true} \mid \textbf{false} \mid e_1 \leqslant e_2 \mid \neg b' \mid b_1 \wedge b_2 \\
S \in \textbf{Stmt} &::= x := e \mid \textbf{skip} \mid S_1 ; S_2 \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \mid \textbf{while}\ b\ \textbf{do}\ S' \\
&\quad \mid \textbf{begin}\ D_V\ D_P\ S\ \textbf{end} \mid \textbf{call}\ p \mid \textbf{call}\ p(\textbf{vr}\ x) \\
&\quad \mid \textbf{read}\ x \mid \textbf{write}\ e \\
D_V \in \textbf{VDecl} &::= \textbf{var}\ x ; D_V \mid \varepsilon \\
D_P \in \textbf{PDecl} &::= \textbf{proc}\ p\ \textbf{is}\ (S) ; D_P \mid \textbf{proc}\ p(\textbf{vr}\ x)\ \textbf{is}\ (S) ; D_P \mid \varepsilon \\
\textbf{Prog} &::= \textbf{prog}\ S
\end{aligned}
$$

Dziedziny semantyczne

$$
\begin{aligned}
\mathbf{Int} &= \ldots \\
\mathbf{Bool} &= \ldots \\
\mathbf{Loc} &= \ldots \\
\mathbf{Store} &= \ldots \\
\mathbf{VEnv} &= \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{Input} &= \mathbf{Int} \times \mathbf{Input} + \{\mathbf{eof}\} \\
\mathbf{State} &= \mathbf{Store} \times \mathbf{Input} \\
\mathbf{Output} &= \mathbf{Int} \times \mathbf{Output} + \{\mathbf{eof}, ??\}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{Cont} &= \mathbf{State} \to \mathbf{Output} \\
\mathbf{Cont_E} &= \mathbf{Int} \to \mathbf{Cont} \\
\mathbf{Cont_B} &= \mathbf{Bool} \to \mathbf{Cont} \\
\mathbf{Cont_{D_V}} &= \mathbf{VEnv} \to \mathbf{Cont} \\
\mathbf{Cont_{D_P}} &= \mathbf{PEnv} \to \mathbf{Cont}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{PROC_0} &= \mathbf{Cont} \to \mathbf{Cont} \\
\mathbf{PROC_1^{vr}} &= \mathbf{Loc} \to \mathbf{PROC_0} \\
\mathbf{PEnv} &= \\
&\quad \mathbf{IDE} \to (\mathbf{PROC_0} + \mathbf{PROC_1^{vr}} + \{??\})
\end{aligned}
$$

Funkcje semantyczne

$$\mathcal{E}: \mathbf{Exp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont_E} \to \mathbf{Cont}}_{\mathbf{EXP}}$$

$$\mathcal{B}: \mathbf{BExp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont_B} \to \mathbf{Cont}}_{\mathbf{BEXP}}$$

$$\mathcal{S}: \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont} \to \mathbf{Cont}}_{\mathbf{STMT}}$$

$$\mathcal{D}_V: \mathbf{VDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont_{D_V}} \to \mathbf{Cont}}_{\mathbf{VDECL}}$$

$$\mathcal{D}_P: \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont_{D_P}} \to \mathbf{Cont}}_{\mathbf{PDECL}}$$

$$\mathcal{P}: \mathbf{Prog} \to \underbrace{\mathbf{Input} \to \mathbf{Output}}_{\mathbf{PROG}}$$

Przykładowe klauzule semantyczne

Programy:

$$\mathcal{P}[\![\textbf{prog } S]\!]\, i = \mathcal{S}[\![S]\!]\, \rho_V^\emptyset\, \rho_P^\emptyset\, \kappa^\emptyset\, \langle s^\emptyset, i\rangle$$
$$\text{where } \rho_V^\emptyset\, x = ??, \rho_P^\emptyset\, p = ??, \kappa^\emptyset\, s = \textbf{eof},$$
$$s^\emptyset\, next = 0, s^\emptyset\, l = ??$$

Deklaracje:

$$\mathcal{D}_P[\![\varepsilon]\!]\, \rho_V\, \rho_P\, \kappa_P = \kappa_P\, \rho_P$$
$$\mathcal{D}_P[\![\textbf{proc } p \textbf{ is } (S); D_P]\!]\, \rho_V\, \rho_P = $$
$$\mathcal{D}_P[\![D_P]\!]\, \rho_V\, \rho_P[p \mapsto P] \text{ where } P = \mathcal{S}[\![S]\!]\, \rho_V\, \rho_P[p \mapsto P]$$
$$\mathcal{D}_V[\![\textbf{var } x; D_V]\!]\, \rho_V\, \kappa_V\, \langle s, i\rangle = $$
$$\mathcal{D}_V[\![D_V]\!]\, \rho_V'\, \kappa_V\, \langle s', i\rangle \text{ where } l = s\, next, \rho_V' = \rho_V[x \mapsto l],$$
$$s' = s[l \mapsto ??, next \mapsto l + 1]$$

Tak naprawdę nie użyto tu kontynuacji, ale:

klauzule można napisać w bardziej standardowym stylu kontynuacyjnym

Przykładowe klauzule semantyczne

Wyrażenia:

$\mathcal{E}[\![x]\!]\ \rho_V\ \kappa_E = \lambda\langle s,i\rangle{:}\mathbf{State}.\kappa_E\ n\ \langle s,i\rangle$ where $l = \rho_V\ x, n = s\ l$

to oznacza: ?? jeśli $\rho_V\ x = ??$ lub $s\ l = ??$

$\mathcal{E}[\![e_1 + e_2]\!]\ \rho_V\ \kappa_E =$
$\quad \mathcal{E}[\![e_1]\!]\ \rho_V\ \lambda n_1{:}\mathbf{Int}.\mathcal{E}[\![e_2]\!]\ \rho_V\ \lambda n_2{:}\mathbf{Int}.\kappa_E\ (n_1 + n_2)$

sprawdź typy!

Wyrażenia logiczne:

$\mathcal{B}[\![\mathbf{true}]\!]\ \rho_V\ \kappa_B = \kappa_B\ \mathbf{tt}$

$\mathcal{B}[\![e_1 \leqslant e_2]\!]\ \rho_V\ \kappa_B =$
$\quad \mathcal{E}[\![e_1]\!]\ \rho_V\ \lambda n_1{:}\mathbf{Int}.\mathcal{E}[\![e_2]\!]\ \rho_V\ \lambda n_2{:}\mathbf{Int}.$
$\quad\quad \kappa_B\ \mathit{ifte}(n_1 \leqslant n_2, \mathbf{tt}, \mathbf{ff})$

bez komentarza?

Instrukcje

$$\mathcal{S}[\![x := e]\!]\ \rho_V\ \rho_P\ \kappa = \mathcal{E}[\![e]\!]\ \rho_V\ (\lambda n{:}\mathbf{Int}.\lambda\langle s, i\rangle{:}\mathbf{State}.\kappa\ \langle s[l \mapsto n], i\rangle)$$
$$\text{where } l = \rho_V\ x$$
$$\mathcal{S}[\![\mathbf{skip}]\!]\ \rho_V\ \rho_P = id_{\mathbf{Cont}}$$
$$\mathcal{S}[\![S_1; S_2]\!]\ \rho_V\ \rho_P\ \kappa = \mathcal{S}[\![S_1]\!]\ \rho_V\ \rho_P\ (\mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P\ \kappa)$$
$$\mathcal{S}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\ \rho_V\ \rho_P\ \kappa =$$
$$\mathcal{B}[\![b]\!]\ \rho_V\ \lambda v{:}\mathbf{Bool}.ifte(v, \mathcal{S}[\![S_1]\!]\ \rho_V\ \rho_P\ \kappa, \mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P\ \kappa)$$
$$\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ \rho_P\ \kappa =$$
$$\mathcal{B}[\![b]\!]\ \rho_V\ \lambda v{:}\mathbf{Bool}.ifte(v, \mathcal{S}[\![S]\!]\ \rho_V\ \rho_P\ (\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ \rho_P\ \kappa), \kappa)$$
$$\mathcal{S}[\![\mathbf{call}\ p]\!]\ \rho_V\ \rho_P = P \text{ where } P = \rho_P\ p$$
$$\mathcal{S}[\![\mathbf{call}\ p(\mathbf{vr}\ x)]\!]\ \rho_V\ \rho_P = P\ l \text{ where } P = \rho_P\ p \in \mathbf{PROC}_1^{\mathrm{vr}},\ \ l = \rho_V\ x$$
$$\mathcal{S}[\![\mathbf{read}\ x]\!]\ \rho_V\ \rho_P\ \kappa\ \langle s, i\rangle = \kappa\ \langle s[l \mapsto n], i'\rangle \text{ where } l = \rho_V\ x, \langle n, i'\rangle = i$$
$$\mathcal{S}[\![\mathbf{write}\ e]\!]\ \rho_V\ \rho_P\ \kappa = \mathcal{E}[\![e]\!]\ \rho_V\ \lambda n{:}\mathbf{Int}.\lambda\langle s, i\rangle{:}\mathbf{State}.\langle n, \kappa\ \langle s, i\rangle\rangle$$

Bloki

$$\mathcal{S}[\![\textbf{begin}\ D_V\ D_P\ S\ \textbf{end}]\!]\ \rho_V\ \rho_P\ \kappa = \\ \quad \mathcal{D}_V[\![D_V]\!]\ \rho_V\ \lambda\rho'_V{:}\textbf{VEnv}.\mathcal{D}_P[\![D_P]\!]\ \rho'_V\ \rho_P\ \lambda\rho'_P{:}\textbf{PEnv}.\mathcal{S}[\![S]\!]\ \rho'_V\ \rho'_P\ \kappa$$

Wydzieliliśmy tę klauzulę, bo zaraz dodamy skoki. . .

Skoki

$$S \in \textbf{Stmt} ::= \ldots \mid L{:}S \mid \textbf{goto}\ L$$
$$L \in \textbf{LAB} ::= \ldots$$

- Etykiety są widoczne wewnątrz bloku, w którym zostały zadeklarowane
- Nie zezwalamy na skoki do bloku, choć skoki do innych instrukcji są dozwolone

Semantyka — szkic

- Jeszcze jedno środowisko:

$$\mathbf{LEnv} = \mathbf{LAB} \to (\mathbf{Cont} + \{??\})$$

- Dodajemy je jako dodatkowy parametr dla odpowiednich funkcji semantycznych:

$$\mathcal{S}: \ \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{LEnv} \to \mathbf{Cont} \to \mathbf{Cont}}_{\mathbf{STMT}}$$

$$\mathcal{D}_P: \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{LEnv} \to \mathbf{Cont}_{\mathbf{D_P}} \to \mathbf{Cont}}_{\mathbf{PDECL}}$$

- Klauzule semantyczne dla deklaracji i instrukcji w „starej" postaci mają dodatkowy parametr, który jedynie przekazują w „dół"; semantyka dla programów wprowadza środowisko etykiet nie zawierające żadnej etykiety.

. . . cdn.

Skoki — szkic semantyki cd.

- Dodajemy semantykę dla instrukcji przypominającą tę dla deklaracji:

$$\mathcal{D}_S : \mathbf{Stmt} \to \mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{LEnv} \to \mathbf{Cont} \to \mathbf{LEnv}$$

- z kilkowa łatwymi klauzulami:

$$\mathcal{D}_S[\![x := e]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \rho_L$$

i podobnie dla **skip**, **call** p, **call** $p(\mathbf{vr}\ x)$, **read** x, **write** e oraz **goto** L, gdzie nie wprowadza się żadnych widocznych etykiet. O dziwo także:

$$\mathcal{D}_S[\![\mathbf{begin}\ D_V\ D_P\ S\ \mathbf{end}]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \rho_L$$

...cdn.

Skoki — szkic semantyki cd.

- I teraz kilka nie tak całkiem oczywistych klauzul:

$$
\begin{aligned}
&\mathcal{D}_S[\![S_1; S_2]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \\
&\qquad \mathcal{D}_S[\![S_1]\!]\ \rho_V\ \rho_P\ \rho_L\ (\mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa) + \mathcal{D}_S[\![S_2]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa \\
&\mathcal{D}_S[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \\
&\qquad \mathcal{D}_S[\![S_1]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa + \mathcal{D}_S[\![S_2]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa \\
&\mathcal{D}_S[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \\
&\qquad \mathcal{D}_S[\![S]\!]\ \rho_V\ \rho_P\ \rho_L\ (\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa) \\
&\mathcal{D}_S[\![L\!:\!S]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \\
&\qquad (\mathcal{D}_S[\![S]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa)[L \mapsto \mathcal{S}[\![S]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa]
\end{aligned}
$$

Trzeba jeszcze wyjaśnić „uaktualnianie”:

$$
(\rho_L + \rho'_L)\ L = \begin{cases} \rho_L\ L & \text{jeśli } \rho'_L\ L = ?? \\ \rho'_L\ L & \text{jeśli } \rho'_L\ L \neq ?? \end{cases}
$$

. . . cdn.

Skoki — szkic semantyki cd.

- Na koniec dodajemy kilka klauzul do (zwykłej) semantyki instrukcji etykietowanych, skoków (teraz oczywiste) i bloków — dość skomplikowane:

$$
\begin{array}{l}
\mathcal{S}[\![L{:}S]\!] = \mathcal{S}[\![S]\!] \\
\mathcal{S}[\![\mathbf{goto}\ L]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \kappa_L \text{ where } \kappa_L = \rho_L\ L \\
\mathcal{S}[\![\mathbf{begin}\ D_V\ D_P\ S\ \mathbf{end}]\!]\ \rho_V\ \rho_P\ \rho_L\ \kappa = \\
\quad \mathcal{D}_V[\![D_V]\!]\ \rho_V\ \lambda\rho'_V{:}\mathbf{VEnv}.\mathcal{D}_P[\![D_P]\!]\ \rho'_V\ \rho_P\ \rho_L\ \lambda\rho'_P{:}\mathbf{PEnv}. \\
\qquad\qquad \mathcal{S}[\![S]\!]\ \rho'_V\ \rho'_P\ \rho'_L\ \kappa \text{ where } \rho'_L = \mathcal{D}_S[\![S]\!]\ \rho'_V\ \rho'_P\ (\rho_L + \rho'_L)\ \kappa
\end{array}
$$

. . . nie do końca dobrze?

- trzeba sprawdzić, że etykiety danego bloku nie powtarzają się — łatwe!
- etykiety danego bloku powinny być widoczne w deklaracjach procedur w tym bloku
- wszystkie zewnętrzne etykiety muszą zostać związane z $\perp$

wymaga więcej uwagi przy uaktualnianiu!

„Semantyka standardowa”

- kontynuacje (w celu obsługi różnego rodzajów skoków i uproszczenia notacji)
- uważna klasyfikacja różnych dziedzin wartości (wartości przypisywalne, składowalne, wyjściowe, domknięcia itd.) z odpowiadającą im semantyką wyrażeń (różnych rodzajów)
- dziedziny Scotta i równania dziedzinowe
- jedynie funkcje ciągłe
- . . .

. . . już wkrótce . . .

Częściowe porządki zupełne

Częściowy porządek zupełny, *cpo*:

$$\boxed{\mathbf{D} = \langle D, \sqsubseteq, \bot \rangle}$$

- $\sqsubseteq \subseteq D \times D$ jest porządkiem częściowym na D takim, że każdy przeliczalny łańcuch $d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_i \sqsubseteq \ldots$ ma kres górny $\bigsqcup_{i>0} d_i$ w D
- $\bot \in D$ jest najmniejszym elementem w sensie porządku $\sqsubseteq$.

Na marginesie: Równoważnie: wszystkie przeliczalne *skierowane* podzbiory D mają kresy górne w D. ($\Delta \subseteq D$ is *skierowany* jeśli dla każdych $x, y \in \Delta$ istnieje $d \in \Delta$ spełniający $x \sqsubseteq d$ i $y \sqsubseteq d$.)
Na marginesie: Wymaganie, aby *wszystkie* łańcuchy miały kresy górne w D *nie* jest równoważne ($C \subseteq D$ jest *łańcuchem* jeśli dla wszystkich $x, y \in C$, $x \sqsubseteq y$ lub $y \sqsubseteq x$.)
Jednak żądanie aby *wszystkie przeliczalne* łańcuchy miały kresy górne w D jest równoważne.

Przykłady

Przykłady	Kontrprzykłady	Komentarze
$\langle \mathcal{P}(X), \subseteq, \emptyset \rangle$	$\langle \mathcal{P}_{fin}(X), \subseteq, \emptyset \rangle$	$\mathcal{P}(X)$ to zbiór wszystkich podzbiorów X, a $\mathcal{P}_{fin}(X)$ to zbiór wszystkich skończonych podzbiorów X
$\langle X \rightharpoonup Y, \subseteq, \emptyset_{X \rightharpoonup Y} \rangle$	$\langle X \to Y, \subseteq, ??? \rangle$	zbiory funkcji częściowych i całkowitych
$\langle \mathbf{Nat}^{\infty}, \leqslant, 0 \rangle$	$\langle \mathbf{Nat}, \leqslant, 0 \rangle$	$\mathbf{Nat}^{\infty} = \mathbf{Nat} \cup \{\omega\}$; $n \leqslant \omega$, dla każdego $n \in \mathbf{Nat}$
$\langle (\mathbf{R}^{+})^{\infty}, \leqslant, 0 \rangle$	$\langle (\mathbf{Q}^{+})^{\infty}, \leqslant, 0 \rangle$	nieujemne liczby rzeczywiste $\mathbf{R}^{+}$ i wymierne $\mathbf{Q}^{+}$ z „nieskończonością”
$\langle (\mathbf{R}^{+})^{\leqslant a}, \leqslant, 0 \rangle$	$\langle (\mathbf{Q}^{+})^{\leqslant a}, \leqslant, 0 \rangle$	ich ograniczone wersje
$\langle A^{\leqslant \omega}, \sqsubseteq, \varepsilon \rangle$	$\langle A^{*}, \sqsubseteq, \varepsilon \rangle$	$A^{\leqslant \omega} = A^{*} \cup A^{\omega}$ (skończone i nieskończone napisy o elementach z A, w tym napis pusty ε); $\sqsubseteq$ jest porządkiem prefiksowym

Funkcje ciągłe

Niech $\mathbf{D} = \langle D, \sqsubseteq, \bot \rangle$ i $\mathbf{D}' = \langle D', \sqsubseteq', \bot' \rangle$ będą cpo. Funkcja $f : D \to D'$ jest

- *monotoniczna*, jeśli zachowuje porządek, tzn., dla każdych $d_1, d_2 \in D$,

$$\boxed{d_1 \sqsubseteq d_2 \text{ implikuje } f(d_1) \sqsubseteq' f(d_2)}$$

- *ciągła*, jeśli zachowuje kresy górne wszystkich przeliczalnych łańcuchów, tzn., dla każdego łańcucha $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ w D,

$$\boxed{f(\bigsqcup_{i \geqslant 0} d_i) = \bigsqcup_{i \geqslant 0} f(d_i)}$$

- *rzetelna*, jeśli zachowuje najmniejszy element, tzn.,

$$\boxed{f(\bot) = \bot'}$$

Na marginesie: Funkcje ciągłe są monotoniczne; nie muszą być rzetelne.
Na marginesie: Funkcje monotoniczne nie muszą być ciągłe.

Intuicje?

Topologia

Niech $\mathbf{D} = \langle D, \sqsubseteq, \perp \rangle$ będzie cpo. Mówimy, że zbiór $X \subseteq D$ jest *otwarty* jeśli

- jeśli $d_1 \in X$ i $d_1 \sqsubseteq d_2$ to $d_2 \in X$
- jeśli $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ jest taki, że $\bigsqcup_{i \geqslant 0} d_i \in X$, to dla pewnego $i \geqslant 0$, $d_i \in X$.

Powyższa definicja wprowadza topologię na D:

- $\emptyset$ i D są otwarte
- przecięcie dwóch zbiorów otwartych jest otwarte
- suma dowolnej rodziny zbiorów otwartych jest otwarta

Dla danych cpo $\mathbf{D} = \langle D, \sqsubseteq, \perp \rangle$ i $\mathbf{D}' = \langle D', \sqsubseteq', \perp' \rangle$, funkcja $f \colon D \to D'$ jest ciągła wtedy i tylko wtedy gdy jest ciągła w sensie topologicznym, czyli, dla dowolnego $X' \subseteq D'$ otwartego w $\mathbf{D}'$, jego przeciwobraz względem f, $f^{-1}(X') \subseteq D$, jest otwarty w $\mathbf{D}$.

Więcej intuicji?

Teoria informacji

Traktujemy cpo $\mathbf{D} = \langle D, \sqsubseteq, \perp \rangle$ jako „przestrzeń informacji".

- jeśli $d_1 \sqsubseteq d_2$ to d_2 reprezentuje „więcej informacji" niż d_1; $\perp$ oznacza „brak informacji"
- zbiory skierowane reprezentują spójne zbiory „fragmentów informacji", a kres górny takiego zbioru reprezentuje „informację", którą można otrzymać z „informacji" w tym zbiorze
- funkcja jest monotoniczna, jeśli zawsze dostając więcej informacji, dostarcza więcej informacji
- funkcja jest ciągła jeśli traktuje informacje „fragment po fragmencie"

bardzo nieformalnie

Dla zbioru elementów X, rozważmy cpo $\langle \mathcal{P}(X), \supseteq, X \rangle$ złożone z „informacji" o elementach z X (zbiór $I \subseteq X$ reprezentuje własność — informację — która zachodzi dla wszystkich elementów z I i tylko dla nich)

Najlepsza intuicja?

Funkcje częściowe

$\langle X \rightharpoonup Y, \subseteq, \emptyset_{X \rightharpoonup Y} \rangle$

- $\emptyset_{X \rightharpoonup Y}$ nie jest nigdzie określona
- dla danych dwóch funkcji częściowych $f, g \colon X \rightharpoonup Y$, $f \subseteq g$, jeśli g nie jest określona mniej niż f, a gdy f jest określona, to g daje ten sam wynik
- dla danego zbioru skierowanego funkcji częściowych $\mathcal{F} \subseteq X \rightharpoonup Y$, żadne dwie funkcje z $\mathcal{F}$ nie dają różnych wyników dla pewnego argumentu; zatem $\bigsqcup \mathcal{F} = \bigcup \mathcal{F}$, która jest funkcją częściową w $X \rightharpoonup Y$
- funkcja $F \colon (X \rightharpoonup Y) \to (X' \rightharpoonup Y')$ jest ciągła, jeśli $F(f)(x')$ (dla $f \colon X \rightharpoonup Y$ i $x' \in X'$) zależy jedynie od skończonej liczby aplikacji f do argumentów w X. Typowe funkcje nieciągłe to: sprawdzanie określoności, badanie nieskończenie wielu wartości, ...

nieformalnie !

Twierdzenie o punkcie stałym

Twierdzenie:

Niech $\mathbf{D} = \langle D, \sqsubseteq, \perp \rangle$ *będzie cpo, a* $f : D \to D$ *funkcją ciągłą. Istnieje najmniejszy punkt stały* $\mathit{fix}(f) \in D$ *funkcji* f*, czyli istnieje* $\mathit{fix}(f) \in D$ *taki że:*

- $f(\mathit{fix}(f)) = \mathit{fix}(f)$
- *jeśli* $f(d) = d$ *dla pewnego* $d \in D$ *to* $\mathit{fix}(f) \sqsubseteq d$

Dowód:

Zdefiniujmy $f^0(\perp) = \perp$ oraz $f^{i+1}(\perp) = f(f^i(\perp))$ dla $i \geqslant 0$. Otrzymujemy łańcuch:

$$f^0(\perp) \sqsubseteq f^1(\perp) \sqsubseteq \cdots \sqsubseteq f^i(\perp) \sqsubseteq f^{i+1}(\perp) \sqsubseteq \cdots$$

Weźmy:

$$\boxed{\mathit{fix}(f) = \bigsqcup_{i \geqslant 0} f^i(\perp)}$$

- $f(\mathit{fix}(f)) = f(\bigsqcup_{i \geqslant 0} f^i(\perp)) = \perp \sqcup \bigsqcup_{i \geqslant 0} f(f^i(\perp)) = \bigsqcup_{i \geqslant 0} f^i(\perp) = \mathit{fix}(f)$
- Przypuśćmy, że $f(d) = d$ dla pewnego $d \in D$; wówczas $f^i(\perp) \sqsubseteq d$ for $i \geqslant 0$. Zatem $\mathit{fix}(f) = \bigsqcup_{i \geqslant 0} f^i(\perp) \sqsubseteq d$.

Techniki dowodowe

Niech $\mathbf{D} = \langle D, \sqsubseteq, \bot \rangle$ będzie cpo, a $f : D \to D$ funkcją ciągłą.

Fakt:

Dla każdego $d \in D$, jeśli $f(d) \sqsubseteq d$ to $fix(f) \sqsubseteq d$.

Indukcja stałopunktowa

Własność $P \subseteq D$ jest *dopuszczalna* jeśli jest zachowywana przez kresy górne wszystkich przeliczalnych łańcuchów: dla każdego łańcucha $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$, jeśli $d_i \in P$ dla każdego $i \geqslant 0$ to także $\bigsqcup_{i \geqslant 0} d_i \in P$ oraz $\bot \in P$.

Fakt:

Dla każdej własności dopuszczalnej $P \subseteq D$, która jest domknięta na f (czyli: jeśli $d \in P$ to $f(d) \in P$)

$$fix(f) \in P$$

Przypomnijmy (bezpośrednią) semantykę **while**:

$$\mathcal{S}[\![\textbf{while } b \textbf{ do } S]\!] = \mathit{fix}(\Phi)$$

gdzie $\Phi\colon \mathbf{STMT} \to \mathbf{STMT}$ jest zadane przez
$\Phi(F) = \mathit{cond}(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!];F, \mathit{id}_{\mathbf{State}})$.

Czy **STMT** jest cpo?
Czy Φ jest ciągła?

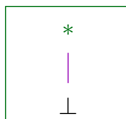
W tym przypadku można łatwo sprawdzić, że rzeczywiście
$\langle \mathbf{STMT}, \subseteq, \emptyset_{\mathbf{State} \to \mathbf{State}} \rangle$ jest cpo i $\Phi\colon \mathbf{STMT} \to \mathbf{STMT}$ jest ciągła.

Ale: nie chcemy sprawdzać takich własności zawsze,
gdy chcemy podać definicję stałopunktową!

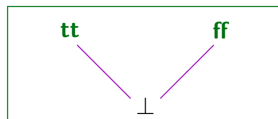
Dziedziny podstawowe

Dla każdego zbioru X, $\mathbf{X}_\perp = \langle X_\perp, \sqsubseteq, \perp \rangle$ jest *płaskim cpo*, gdzie $X_\perp = X \cup \{\perp\}$, $\perp$ jest nowym elementem, $\perp \sqsubseteq x$ dla każdego $x \in X$, a poza tym $\sqsubseteq$ jest trywialna.

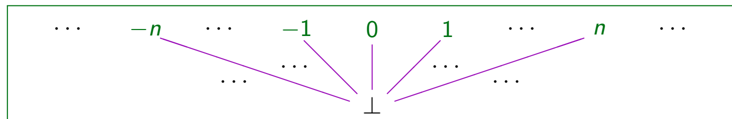
$\{*\}_\perp$:



Bool$_\perp$:



Int$_\perp$:



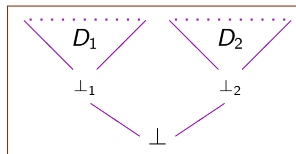
Dla dowolnych cpo $\mathbf{D}_1 = \langle D_1, \sqsubseteq_1, \bot_1 \rangle$ i $\mathbf{D}_2 = \langle D_2, \sqsubseteq_2, \bot_2 \rangle$:

Produkt

Produktem $\mathbf{D}_1$ i $\mathbf{D}_2$ jest następujący cpo:

$$\mathbf{D}_1 \times \mathbf{D}_2 = \langle D_1 \times D_2, \sqsubseteq, \langle \bot_1, \bot_2 \rangle \rangle$$

przy czym dla każdych $d_1, d_1' \in D_1$ i $d_2, d_2' \in D_2$, $\langle d_1, d_2 \rangle \sqsubseteq \langle d_1', d_2' \rangle$ jeśli $d_1 \sqsubseteq_1 d_1'$ i $d_2 \sqsubseteq_2 d_2'$.



Suma

Sumą rozłączną $\mathbf{D}_1$ i $\mathbf{D}_2$ jest następujący cpo:

$$\mathbf{D}_1 + \mathbf{D}_2 = \langle (D_1 \times \{1\}) \cup (D_2 \times \{2\}) \cup \{\bot\}, \sqsubseteq, \bot \rangle$$

przy czym dla $d_1, d_1' \in D_1$, $\langle d_1, 1 \rangle \sqsubseteq \langle d_1', 1 \rangle$ jeśli $d_1 \sqsubseteq_1 d_1'$, dla $d_2, d_2' \in D_2$, $\langle d_2, 2 \rangle \sqsubseteq \langle d_2', 2 \rangle$ jeśli $d_2 \sqsubseteq_2 d_2'$, oraz dla $d_1 \in D_1$, $d_2 \in D_2$, $\bot \sqsubseteq \langle d_1, 1 \rangle$ i $\bot \sqsubseteq \langle d_2, 2 \rangle$.

Dla uniknięcia proliferacji „pinezek”:

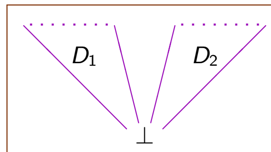
Produkt spłaszczony

Produktem spłaszczonym $\mathbf{D}_1$ i $\mathbf{D}_2$ jest następujący cpo:

$$\mathbf{D}_1 \otimes \mathbf{D}_2 = \langle (D_1 \setminus \{\bot_1\}) \times (D_2 \setminus \{\bot_2\}) \cup \{\bot\}, \sqsubseteq, \bot \rangle$$

przy czym dla wszytkich $d_1, d_1' \in D_1 \setminus \{\bot_1\}$ oraz $d_2, d_2' \in D_2 \setminus \{\bot_2\}$, $\langle d_1, d_2 \rangle \sqsubseteq \langle d_1', d_2' \rangle$ jeśli $d_1 \sqsubseteq_1 d_1'$ i $d_2 \sqsubseteq_2 d_2'$, oraz $\bot \sqsubseteq \langle d_1, d_2 \rangle$.

Suma spłaszczona



Sumą spłaszczoną $\mathbf{D}_1$ i $\mathbf{D}_2$ jest następujący cpo:

$$\mathbf{D}_1 \oplus \mathbf{D}_2 = \langle ((D_1 \setminus \{\bot_1\}) \times \{1\}) \cup ((D_2 \setminus \{\bot_2\}) \times \{2\}) \cup \{\bot\}, \sqsubseteq, \bot \rangle$$

przy czym dla wszystkich $d_1, d_1' \in D_1 \setminus \{\bot_1\}$, $\langle d_1, 1 \rangle \sqsubseteq \langle d_1', 1 \rangle$ jeśli $d_1 \sqsubseteq_1 d_1'$, dla $d_2, d_2' \in D_2 \setminus \{\bot_2\}$, $\langle d_2, 2 \rangle \sqsubseteq \langle d_2', 2 \rangle$ jeśli $d_2 \sqsubseteq_2 d_2'$, oraz $\bot \sqsubseteq \langle d_1, 1 \rangle$ i $\bot \sqsubseteq \langle d_2, 2 \rangle$.

Przestrzenie funkcji

Przestrzeń funkcji ciągłych z $\mathbf{D}_1$ w $\mathbf{D}_2$ to następujący cpo:

$$[\mathbf{D}_1 \to \mathbf{D}_2] = \langle [D_1 \to D_2], \sqsubseteq, \bot \rangle$$

przy czym

- $[D_1 \to D_2]$ jest zbiorem wszystkich *ciągłych* funkcji z D_1 w D_2
- dla funkcji $f, g \colon D_1 \to D_2$, $f \sqsubseteq g$ jeśli dla każdego $d_1 \in D_1$, $f(d_1) \sqsubseteq_2 g(d_1)$
- $\bot(d_1) = \bot_2$ dla każdego $d_1 \in D_1$.

$\sqsubseteq$ nie zależy od porządku w D_1

Dla dowolnego zbioru X, *przestrzenią funkcji* z X w $\mathbf{D}_2$ jest następujący cpo:

$$(X \to \mathbf{D}_2) = \langle X \to D_2, \sqsubseteq, \bot \rangle$$

gdzie $X \to D_2$ jest zbiorem funkcji (całkowitych) z X w D_2 uporządkowanych przez relację $\sqsubseteq$ zdefiniowaną powyżej.

Izomorfizm dziedzin

Dwa cpo $\mathbf{D}_1$ i $\mathbf{D}_2$ są *izomorficzne*

$$\mathbf{D}_1 \cong \mathbf{D}_2$$

jeśli istnieje bijekcja między D_1 a D_2, która zachowuje i odbija porządki.

Przykłady:

$$\mathbf{Bool}_\perp \cong \{*\}_\perp \oplus \{*\}_\perp$$

$$\langle X \rightharpoonup Y, \subseteq, \emptyset_{X \rightharpoonup Y} \rangle \cong \langle X \to Y_\perp, \sqsubseteq, \perp \rangle$$

Dziedziny semantyczne rozważamy z dokładnością do izomorfizmu

Można więc zapomnieć o (wartościach logicznych i) funkcjach częściowych !

Trudniej zapomnieć o liczbach naturalnych

Konstruowanie funkcji ciągłych

- Każda funkcja stała jest ciągła
- Używane do tej pory funkcje częsciowe na zbiorach można zastąpić przez (rzetelne) funkcje ciągłe między dziedzinami płaskimi, np., naciągając nieco notację:
  - $ifte_{\mathbf{D}} \in [\mathbf{Bool}_\perp \times \mathbf{D} \times \mathbf{D} \to \mathbf{D}]$ można określić tak:

$$ifte_{\mathbf{D}}(c, d, d') = \begin{cases} ifte_D(c, d, d') & \text{jeśli } c \neq \perp \\ \perp_{\mathbf{D}} & \text{jeśli } c = \perp \end{cases}$$

  - $_ + _ \in [\mathbf{Int}_\perp \times \mathbf{Int}_\perp \to \mathbf{Int}_\perp]$ można zdefiniować jako:

$$n + n' = \begin{cases} n + n' & \text{jeśli } n \neq \perp \text{ i } n' \neq \perp \\ \perp & \text{jeśli } n = \perp \text{ lub } n' = \perp \end{cases}$$

Dalsze konstrukcje

- złożenie funkcji: $_;_ \in [[\mathbf{D}_1 \to \mathbf{D}_2] \times [\mathbf{D}_2 \to \mathbf{D}_3] \to [\mathbf{D}_1 \to \mathbf{D}_3]]$, tzn.:
  - złożenie funkcji ciągłych jest ciągłe
  - funkcja złożenia jest ciągła
- indeksowanie:
  $\mathit{lift}^{\mathbf{I}} \in [[\mathbf{D}_1 \times \ldots \times \mathbf{D}_n \to \mathbf{D}] \to [[\mathbf{I} \to \mathbf{D}_1] \times \ldots \times [\mathbf{I} \to \mathbf{D}_n] \to [\mathbf{I} \to \mathbf{D}]]]$, tzn.:
  - indeksowanie funkcji ciągłej daje funkcję ciągłą
  - funkcja indeksowania jest ciągła
- Weźmy dowolną funkcję $f : D_1 \times \ldots \times D_n \to D$. f jest funkcją ciągłą z dziedziny produktowej $\mathbf{D}_1 \times \ldots \times \mathbf{D}_n$ w $\mathbf{D}$ wtedy i tylko wtedy, gdy jest ciągła względem każdego argumentu z osobna
  - uzasadnia to stosowania lambda notacji do tworzenia funkcji ciągłych: $\Lambda \in [[\mathbf{D}_0 \times \mathbf{D}_1 \times \ldots \times \mathbf{D}_n \to \mathbf{D}] \to [\mathbf{D}_1 \times \ldots \times \mathbf{D}_n \to [\mathbf{D}_0 \to \mathbf{D}]]]$

. . . i dalej

- zastosowanie funkcji ciągłej jest ciągłe; $_(_) \in [[\mathbf{D}_1 \to \mathbf{D}_2] \times \mathbf{D}_1 \to \mathbf{D}_2]$
- rzuty: $\pi_1 \in [\mathbf{D}_1 \times \mathbf{D}_2 \to \mathbf{D}_1]$ i $\pi_2 \in [\mathbf{D}_1 \times \mathbf{D}_2 \to \mathbf{D}_2]$
- (dwuargumentowa funkcja tworzenia par jest ciągła, tylko jak to zapisać?)
- włożenia: $\iota_1 \in [\mathbf{D}_1 \to \mathbf{D}_1 + \mathbf{D}_2]$ oraz $\iota_2 \in [\mathbf{D}_2 \to \mathbf{D}_1 + \mathbf{D}_2]$,
- sprawdzanie przynależności do dziedziny: $\mathit{is_in_1} \in [\mathbf{D}_1 + \mathbf{D}_2 \to \mathbf{Bool}_\perp]$ oraz $\mathit{is_in_2} \in [\mathbf{D}_1 + \mathbf{D}_2 \to \mathbf{Bool}_\perp]$
- parowanie funkcji: $\langle _, _ \rangle \colon [[\mathbf{D} \to \mathbf{D}_1] \times [\mathbf{D} \to \mathbf{D}_2] \to [\mathbf{D} \to \mathbf{D}_1 \times \mathbf{D}_2]]$, gdzie dla $f \in [\mathbf{D} \to \mathbf{D}_1]$ i $g \in [\mathbf{D} \to \mathbf{D}_2]$, $\langle f, g \rangle = \lambda d{:}D.\langle f(d), g(d) \rangle$.
- sumowanie funkcji: $[_, _] \colon [[\mathbf{D}_1 \to \mathbf{D}] \times [\mathbf{D}_2 \to \mathbf{D}] \to [\mathbf{D}_1 + \mathbf{D}_2 \to \mathbf{D}]]$, gdzie dla $f \in [\mathbf{D}_1 \to \mathbf{D}]$ i $g \in [\mathbf{D}_2 \to \mathbf{D}]$, $[f, g](d) = \mathit{ifte}_{\mathbf{D}}(\mathit{is_in_1}(d), f(d), g(d))$

Wystarczy...

- operator najmniejszego punktu stałego $\mathit{fix}(_) \in [[\mathbf{D} \to \mathbf{D}] \to \mathbf{D}]$
  - najmniejszy punkt stały jest funcją ciągłą na dziedzinie funkcji ciągłych
  - dla $\mathbf{D} = [\mathbf{D}_1 \to \mathbf{D}_2]$, najmniejszy punkt stały funkcji ciągłej na dziedzinie funkcji ciągłych jest funcją ciągłą...

Nie wszystkie funkcje są ciągłe...

Ale jest ich wystarczająco dużo...

Elementy cpo $d_1 \in D_1$, ..., $d_n \in D_n$ można definiować pisząc *układy równań stałopunktowych*:

$$\begin{aligned} d_1 &= \Phi_1(d_1, \ldots, d_n) \\ &\ldots \\ d_n &= \Phi_n(d_1, \ldots, d_n) \end{aligned}$$

gdzie $\Phi_1 \in [\mathbf{D}_1 \times \ldots \times \mathbf{D}_n \to \mathbf{D}_1]$, ..., $\Phi_n \in [\mathbf{D}_1 \times \ldots \times \mathbf{D}_n \to \mathbf{D}_n]$.
Definiuje to $\langle d_1, \ldots, d_n \rangle$ jako najmniejszy punkt stały funkcji

$$\langle \Phi_1, \ldots, \Phi_n \rangle \in [\mathbf{D}_1 \times \ldots \times \mathbf{D}_n \to \mathbf{D}_1 \times \ldots \times \mathbf{D}_n]$$

Funkcje ciągłe, używane w powyższych definicjach, można budować wykorzystując funkcje podstawowe i różne omówione już sposoby ich łączenia.

Równania dziedzinowe

$$
\begin{aligned}
\mathbf{Int} &= \{0, 1, -1, 2, -2, \ldots\}_\perp \\
\mathbf{Bool} &= \{\mathbf{tt}, \mathbf{ff}\}_\perp \\
\mathbf{State} &= \mathbf{Var} \to \mathbf{Int} \\
\mathbf{EXP} &= [\mathbf{State} \to \mathbf{Int}] \\
\mathbf{BEXP} &= [\mathbf{State} \to \mathbf{Bool}] \\
\mathbf{STMT} &= [\mathbf{State} \to \mathbf{State}]
\end{aligned}
$$

Po prostu korzystamy z omówionych poprzednio operacji do tworzenia cpo

Jeśli definicje dziedzin okazują się rekurencyjne to stosujemy technikę aproksymacji, jak w przypadku elementów dziedzin

Rekurencyjne równania dziedzinowe

$$\mathbf{Stream} = A_\perp \times \mathbf{Stream}$$

Dokładniej: $\mathbf{Stream} = A_\perp \otimes_L \mathbf{Stream}$

$$\begin{aligned}
\mathbf{Stream}^0 &= \{\perp\} \\
\mathbf{Stream}^1 &= \{\perp \sqsubseteq \langle a_1, \perp\rangle\} \\
\mathbf{Stream}^2 &= \{\perp \sqsubseteq \langle a_1, \perp\rangle \sqsubseteq \langle a_1, \langle a_2, \perp\rangle\rangle\} \\
&\dots \\
\mathbf{Stream}^n &= \{\perp \sqsubseteq \langle a_1, \perp\rangle \sqsubseteq \langle a_1, \langle a_2, \perp\rangle\rangle \sqsubseteq \cdots \sqsubseteq \langle a_1, \langle a_2, \langle \dots, \langle a_n, \perp\rangle \dots\rangle\rangle\rangle\} \\
&\dots \\
\mathbf{Stream} &= \bigsqcup_{n \geqslant 0} \mathbf{Stream}^n \\
&= \{\perp \sqsubseteq \langle a_1, \perp\rangle \sqsubseteq \langle a_1, \langle a_2, \perp\rangle\rangle \sqsubseteq \cdots \sqsubseteq \langle a_1, \langle a_2, \langle \dots, \langle a_n, \perp\rangle \dots\rangle\rangle\rangle \\
&\qquad \sqsubseteq \cdots \langle a_1, \langle a_2, \langle \dots, \langle a_n, \langle \dots\rangle\rangle \dots\rangle\rangle\rangle\}
\end{aligned}$$

gdzie $a_1, a_2, \dots, a_n, \dots \in A$.

Rekurencyjne równania dziedzinowe

$$\textbf{Stream} = A_\perp \times \textbf{Stream}$$

$\textbf{Stream}^0 = \{\perp\}$
$\textbf{Stream}^1 = \{\perp \sqsubseteq \langle a_1, \perp\rangle\}$
$\textbf{Stream}^2 = \{\perp \sqsubseteq \langle a_1, \perp\rangle \sqsubseteq \langle a_1, a_2, \perp\rangle\}$
$\dots$
$\textbf{Stream}^n = \{\perp \sqsubseteq \langle a_1, \perp\rangle \sqsubseteq \langle a_1, a_2, \perp\rangle \sqsubseteq \cdots \sqsubseteq \langle a_1, a_2, \dots, a_n, \perp\rangle\}$
$\dots$
$\textbf{Stream} = \bigsqcup_{n \geqslant 0} \textbf{Stream}^n =$
$\{\perp \sqsubseteq \langle a_1, \perp\rangle \sqsubseteq \langle a_1, a_2, \perp\rangle \sqsubseteq \cdots \sqsubseteq \langle a_1, a_2, \dots, a_n, \perp\rangle \sqsubseteq$
$\sqsubseteq \cdots \sqsubseteq \langle a_1, a_2, \dots, a_n, \dots\rangle\}$

gdzie $a_1, a_2, \dots, a_n, \dots \in A$.

Dokładniej:
$\textbf{Stream} = A_\perp \otimes_L \textbf{Stream}$

Problemy?

Jeśli definicje dziedzin okazują się rekurencyjne,
to stosujemy technikę aproksymacji,
jak w przypadku elementów dziedzin

Naprawdę?
Nie ma problemów?

Przypuśćmy, że chcemy wprowadzić (bezparametrowe) procedury, będące po prostu nazwanymi instrukcjami, które chcemy przechowywać w stanach i używać w instrukcjach wywołania procedury:

$$\mathbf{State} = \mathbf{Var} \to \mathbf{VAL} \qquad \mathbf{VAL} = \mathbf{Int} + \mathbf{PROC} \qquad \mathbf{PROC} = [\mathbf{State} \to \mathbf{State}]$$

Dziedziny refleksywne

Nie istnieje (nietrywialny) *zbiór* spełniający

$$\mathbf{D} = \mathbf{D} \to \mathbf{D}$$

Jednak każda postać *samo-aplikacji* (beztypowe parametry procedur, wiązanie dynamiczne itp.) wymaga dziedziny semantycznej tej lub podobnej postaci.

Modele dla λ-rachunku

W szczególności są one niezbędne jako modele dla λ-rachunku, formalnego rachunku bez typów, gdzie każdy term może być aplikowany do każdego argumentu.

Historia: semantyka języka ALGOL 60
Christopher Strachey, Dana Scott & wielu innych

Dobre naiwne rozwiązanie

„Naiwna" semantyka denotacyjna

- Stosujemy standardowe teoriomnogościowe konstrukcje dziedzin.
- Nigdy nie używamy „ciężkiej" rekurencji, występującej w definicji dziedziny refleksywnej.
- Do rozwiązywania równań dziedzinowych stosujemy „naiwne" teoriomnogościowe przybliżenia i sumy teoriomnogościowe.
- Działa to dla języków z typami, z hierarchią pojęć i dziedzin.

Rozwiązanie

Scott-eria

- Ograniczamy rozmiar dziedzin: przeliczalna baza oraz dodatkowe techniczne ograniczenia
- Stosujemy jedynie funkcje ciągłe
- Definiujemy „dziedzinę wszystkich dziedzin", w której można zinterpretować wszystkie takie dziedziny
- Definiujemy funkcje ciągłe na tej dziedzinie, które interpretują odpowiednie konstruktory dziedzin
- Zapisujemy i rozwiązujemy równania dziedzinowe jako równania stałopunktowe w tej dziedzinie

Modele: $\mathbf{P}\omega$, $\mathbf{T}\omega$, *systemy informacyjne, ...*

Poprawność

Programy powinny być:

- *czytelne; efektywne; niezawodne; przyjazne dla użytkownika; dobrze udokumentowane; . . .*
- *ale nade wszystko, POPRAWNE*
- *nie zapominajmy, także wykonywalne. . .*

Poprawność programu ma sens jedynie względem ścisłej *specyfikacji* wymagań.

Zdefiniowanie poprawności

Są potrzebne:

- Formalna definicja rozważanych programów

  składnia i semantyka języka programowania

- Formalna definicja stosowanych specyfikacji

  składnia i semantyka języka specyfikowania

- Formalna definicja pojęcia poprawności

  co to znaczy, że program spełnia specyfikację

Dowodzenie poprawności

Są potrzebne:

- System formalny do dowodzenia poprawności programów względem specyfikacji

logika z rachunkiem do dowodzenia faktów o poprawności programów

- (Meta-)dowód, że w logice da się udowodnić jedynie prawdziwe fakty

poprawność logiki

- (Meta-)dowód, że w logice da się udowodnić wszystkie prawdziwe fakty

pełność logiki

przy akceptowalnych założeniach technicznych

Wyspecyfikowany program

$$\{n \geqslant 0\}$$
$$rt := 0;\ sqr := 1;$$
$$\textbf{while}\ sqr \leqslant n\ \textbf{do}$$
$$(rt := rt + 1;\ sqr := sqr + 2 * rt + 1)$$
$$\{rt^2 \leqslant n < (rt + 1)^2\}$$

Jeśli rozpoczniemy wykonanie programu z nieujemną wartością n
i program zakończy się, to wartością zmiennej rt
będzie część całkowita pierwiastka kwadratowego z n

Formuły wyrażające poprawność:

$$\{\varphi\}\ S\ \{\psi\}$$

- S jest instrukcją języka TINY
- *warunek wstępny* φ oraz *warunek końcowy* ψ są formułami pierwszego rzędu o zmiennych z **Var**

Zamierzone znaczenie:

Częściowa poprawność:
nie gwarantujemy zakończenia obliczeń!

Jeśli obliczenie programu S rozpocznie się w stanie spełniającym φ i zakończy się, to stan końcowy tego obliczenia spełnia ψ

Definicja formalna

Przypomnijmy najprostszą semantykę języka TINY:

$$\mathcal{S} : \mathbf{Stmt} \to \mathbf{State} \rightharpoonup \mathbf{State}$$

Wprowadzamy nową kategorię składniową:

$$\varphi \in \mathbf{Form} ::= b \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \neg\varphi' \mid \exists x.\varphi' \mid \forall x.\varphi'$$

z odpowiednią funkcją semantyczną:

$$\mathcal{F} : \mathbf{Form} \to \mathbf{State} \to \mathbf{Bool}$$

i standardowymi klauzulami semantycznymi.

Definiujemy także w zwykły sposób *zmienne wolne* formuły oraz *podstawienie* wyrażenia za zmienną

Dalszy ciąg notacji

Dla $\varphi \in \textbf{Form}$:

$$\{\varphi\} = \{s \in \textbf{State} \mid \mathcal{F}[\![\varphi]\!]\, s = \textbf{tt}\}$$

Dla $S \in \textbf{Stmt}$, $A \subseteq \textbf{State}$:

$$A\,[\![S]\!] = \{s \in \textbf{State} \mid \mathcal{S}[\![S]\!]\, a = s, \text{dla pewnego } a \in A\}$$

$\models \{\varphi\}\, S\, \{\psi\}$

wttw

$\{\varphi\}\, [\![S]\!] \subseteq \{\psi\}$

Przeczytajmy to:

Formuła częściowej poprawności $\{\varphi\}\, S\, \{\psi\}$ jest prawdziwa, co zapisujemy jako $\models \{\varphi\}\, S\, \{\psi\}$, jeśli dla wszystkich stanów $s \in$ **State**

jeśli $\mathcal{F}[\![\varphi]\!]\, s = \mathbf{tt}$ oraz $\mathcal{S}[\![S]\!]\, s \in$ **State**

to $\mathcal{F}[\![\psi]\!]\, (\mathcal{S}[\![S]\!]\, s) = \mathbf{tt}$

Logika Hoare'a: reguły wnioskowania

$$\frac{}{\{\varphi[x \mapsto e]\}\ x := e\ \{\varphi\}}$$

$$\frac{}{\{\varphi\}\ \textbf{skip}\ \{\varphi\}}$$

$$\frac{\{\varphi\}\ S_1\ \{\theta\} \quad \{\theta\}\ S_2\ \{\psi\}}{\{\varphi\}\ S_1; S_2\ \{\psi\}}$$

$$\frac{\{\varphi \wedge b\}\ S_1\ \{\psi\} \quad \{\varphi \wedge \neg b\}\ S_2\ \{\psi\}}{\{\varphi\}\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \{\psi\}}$$

$$\frac{\{\varphi \wedge b\}\ S\ \{\varphi\}}{\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ S\ \{\varphi \wedge \neg b\}}$$

$$\frac{\varphi' \Rightarrow \varphi \quad \{\varphi\}\ S\ \{\psi\} \quad \psi \Rightarrow \psi'}{\{\varphi'\}\ S\ \{\psi'\}}$$

Przykład dowodu

Udowodnimy następującą formułę Hoare'a:

$$
\begin{array}{l}
\{n \geqslant 0\} \\
\quad rt := 0; \\
\quad sqr := 1; \\
\quad \textbf{while } sqr \leqslant n \textbf{ do} \\
\qquad\quad rt := rt + 1; \\
\qquad\quad sqr := sqr + 2 * rt + 1 \\
\{rt^2 \leqslant n \wedge n < (rt+1)^2\}
\end{array}
$$

Będziemy niejawnie stosować regułę wynikania, zastępując asercje równoważnymi o prostszej postaci

Krok po kroku

- $\{n \geqslant 0\}\ rt := 0\ \{n \geqslant 0 \land rt = 0\}$
- $\{n \geqslant 0 \land rt = 0\}\ sqr := 1\ \{n \geqslant 0 \land rt = 0 \land sqr = 1\}$
- $\{n \geqslant 0\}\ rt := 0;\ sqr := 1\ \{n \geqslant 0 \land rt = 0 \land sqr = 1\}$
- $\boxed{\{n \geqslant 0\}\ rt := 0;\ sqr := 1\ \{sqr = (rt+1)^2 \land rt^2 \leqslant n\}}$

EUREKA!!!
Właśnie odkryliśmy *niezmiennik pętli*

Niezmiennik pętli

- $\{(sqr = (rt+1)^2 \wedge rt^2 \leqslant n) \wedge sqr \leqslant n\}\ rt := rt + 1\ \{sqr = rt^2 \wedge sqr \leqslant n\}$
- $\{sqr = rt^2 \wedge sqr \leqslant n\}\ sqr := sqr + 2 * rt + 1\ \{sqr = (rt+1)^2 \wedge rt^2 \leqslant n\}$
- $\{(sqr = (rt+1)^2 \wedge rt^2 \leqslant n) \wedge sqr \leqslant n\}$
  $rt := rt + 1; sqr := sqr + 2 * rt + 1$
  $\{sqr = (rt+1)^2 \wedge rt^2 \leqslant n\}$
- $\{sqr = (rt+1)^2 \wedge rt^2 \leqslant n\}$
  while $sqr \leqslant n$ **do**
  $rt := rt + 1; sqr := sqr + 2 * rt + 1$
  $\{(sqr = (rt+1)^2 \wedge rt^2 \leqslant n) \wedge \neg(sqr \leqslant n)\}$

Wreszcie

- $\{sqr = (rt+1)^2 \wedge rt^2 \leqslant n\}$
  while $sqr \leqslant n$ **do**
  $rt := rt + 1; sqr := sqr + 2 * rt + 1$
  $\{rt^2 \leqslant n \wedge n < (rt+1)^2\}$
-

$\{n \geqslant 0\}$
$rt := 0; sqr := 1$
while $sqr \leqslant n$ **do**
$rt := rt + 1; sqr := sqr + 2 * rt + 1$
$\{rt^2 \leqslant n \wedge n < (rt+1)^2\}$

QED

W pełni wyspecyfikowany program

$\{n \geqslant 0\}$
$rt := 0;$
$\{n \geqslant 0 \land rt = 0\}$
$sqr := 1;$
$\{n \geqslant 0 \land rt = 0 \land sqr = 1\}$
while $\{sqr = (rt+1)^2 \land rt^2 \leqslant n\}$ $sqr \leqslant n$ **do**
$\quad rt := rt + 1;$
$\quad \{sqr = rt^2 \land sqr \leqslant n\}$
$\quad sqr := sqr + 2 * rt + 1$
$\{rt^2 \leqslant n < (rt+1)^2\}$

W przedstawionym dowodzie użyliśmy pewnych faktów dotyczących stosowanych typów danych (**Int** z operacjami i relacjami wbudowanymi w składnię języka TINY). Każde użycie reguły wynikania wymaga takich faktów. Zdefiniujmy *teorię* **Int**

$$\mathcal{TH}(\mathbf{Int})$$

jako zbiór wszystkich formuł, które zachodzą we wszystkich stanach. Przedstawiony dowód pokazuje, że:

$$\mathcal{TH}(\mathbf{Int}) \vdash \begin{array}{l} \{n \geqslant 0\} \\ \quad rt := 0; sqr := 1 \\ \quad \textbf{while } sqr \leqslant n \textbf{ do } rt := rt + 1; sqr := sqr + 2 * rt + 1 \\ \{rt^2 \leqslant n \wedge n < (rt+1)^2\} \end{array}$$

Poprawność

Twierdzenie:

System dowodowy Hoare'a (zawierający przedstawione reguły) jest poprawny, tzn.:

$$\textit{jeśli} \quad \boxed{\mathcal{TH}(\mathbf{Int}) \vdash \{\varphi\}\, S\, \{\psi\}} \quad \textit{to} \quad \boxed{\models \{\varphi\}\, S\, \{\psi\}}$$

Zatem przedstawiony dowód formuły poprawności częściowej uzasadnia następujący fakt semantyczny:

$$\models \begin{array}{l} \{n \geqslant 0\} \\ \quad rt := 0;\, sqr := 1 \\ \quad \textbf{while}\ sqr \leqslant n\ \textbf{do}\ rt := rt + 1;\, sqr := sqr + 2 * rt + 1 \\ \{rt^2 \leqslant n \wedge n < (rt+1)^2\} \end{array}$$

Dowód

(**poprawności systemu dowodzenia Hoare'a**)

Przez indukcję po strukturze dowodu w logice Hoare'a:

reguła dla przypisania: Łatwe, ale jest potrzebny lemat (dowodzony przez indukcję po strukturze formuły):

$$\mathcal{F}[\![\varphi[x \mapsto e]]\!]\, s = \mathcal{F}[\![\varphi]\!]\, s[x \mapsto \mathcal{E}[\![e]\!]\, s]$$

Zatem, dla $s \in \mathbf{State}$, jeśli $s \in \{\varphi[x \mapsto e]\}$, to $\mathcal{S}[\![x := e]\!]\, s = s[x \mapsto \mathcal{E}[\![e]\!]\, s] \in \{\varphi\}$.

reguła dla skip: Oczywiste.

reguła dla złożenia: Załóżmy, że $\{\varphi\}\, [\![S_1]\!] \subseteq \{\theta\}$ i $\{\theta\}\, [\![S_2]\!] \subseteq \{\psi\}$. Wtedy $\{\varphi\}\, [\![S_1; S_2]\!] = \{\varphi\}\, [\![S_1]\!]\, [\![S_2]\!] \subseteq \{\theta\}\, [\![S_2]\!] \subseteq \{\psi\}$.

reguła dla if-then-else: Łatwe.

reguła wynikania: Znów to samo, ze względu na oczywistą obserwację, że $\{\varphi_1\} \subseteq \{\varphi_2\}$ wtw, gdy $\varphi_1 \Rightarrow \varphi_2 \in \mathcal{TH}(\mathbf{Int})$.

Poprawność reguły dla pętli

reguła dla pętli: Trzeba pokazać, że najmniejszy punkt stały operatora

$$\Phi(F) = cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!];F, id_{\mathbf{State}})$$

spełnia

$$fix(\Phi)(\{\varphi\}) \subseteq \{\varphi \wedge \neg b\}$$

Postępujemy zgodnie z indukcją stałopunktową (*to jest własność dopuszczalna!*). Przypuśćmy, że $F(\{\varphi\}) \subseteq \{\varphi \wedge \neg b\}$ dla pewnego F: **State** $\rightharpoonup$ **State**, i rozważmy $s \in \{\varphi\}$ z $s' = \Phi(F)(s) \in$ **State**. Mamy dwa przypadki:

- Jeśli $\mathcal{B}[\![b]\!]\ s =$ **ff** to $s' = s \in \{\varphi \wedge \neg b\}$.
- Jeśli $\mathcal{B}[\![b]\!]\ s =$ **tt** to $s' = F(\mathcal{S}[\![S]\!]\ s)$. Otrzymujemy $s' \in \{\varphi \wedge \neg b\}$ na mocy założenia o F, gdyż $\{\varphi \wedge b\}\ [\![S]\!] \subseteq \{\varphi\}$ z założenia indukcyjnego, które implikuje $\mathcal{S}[\![S]\!]\ s \in \{\varphi\}$.

Zatem, $\Phi(F)(\{\varphi\}) \subseteq \{\varphi \wedge \neg b\}$, co kończy dowód.

Problemy z pełnością

- Jeśli $\mathcal{T} \subseteq$ **Form** jest rekrencyjnie przeliczalny, to zbiór wszystkich trójek Hoare'a wyprowadzalnych z $\mathcal{T}$ jest również rekurencyjnie przeliczalny.
- $\models \{\textbf{true}\}\, S\, \{\textbf{false}\}$ wtw, gdy S nie zatrzymuje się dla żadnego stanu początkowego.
- Ponieważ problem stopu dla języka TINY nie jest rozstrzygalny, więc zbiór wszystkich trójek postaci $\{\textbf{true}\}\, S\, \{\textbf{false}\}$, dla których $\models \{\textbf{true}\}\, S\, \{\textbf{false}\}$, nie jest rekurencyjnie przeliczalny.

A jednak:

$$\boxed{\mathcal{TH}(\textbf{Int}) \vdash \{\varphi\}\, S\, \{\psi\}} \quad \text{wtw, gdy} \quad \boxed{\models \{\varphi\}\, S\, \{\psi\}}$$

Formuła logiki Hoare'a jako zadanie programistyczne

Dla zadanych warunków wstępnego φ i końcowego ψ
zbuduj program S, taki że

$$\{\varphi\}\ S\ \{\psi\}$$

Przykład

Zbuduj instrukcję S, taką że

$$\{n \geqslant 0\}\ S\ \{rt^2 \leqslant n \wedge n < (rt+1)^2\}$$

Jedno z poprawnych rozwiązań:

$\{n \geqslant 0\}$

$\quad rt := 0;\ sqr := 1$

$\quad$ **while** $sqr \leqslant n$ **do** $rt := rt + 1;\ sqr := sqr + 2 * rt + 1$

$\{rt^2 \leqslant n \wedge n < (rt+1)^2\}$

A oto inne poprawne rozwiązanie:

$$\{n \geqslant 0\}$$
while true do skip
$$\{rt^2 \leqslant n \wedge n < (rt+1)^2\}$$

gdyż: $\vdash$

$$\{n \geqslant 0\}$$
while $\{\text{true}\}$ **true do skip**
$$\{rt^2 \leqslant n \wedge n < (rt+1)^2\}$$

Poprawność częściowa:
nie wymaga i nie daje gwarancji
zakończenia działania programu

Poprawność całkowita

Poprawność całkowita $=$ poprawność częściowa $+$ warunek stopu

Formuły całkowitej poprawności:

$$[\varphi]\ S\ [\psi]$$

Zamierzone znaczenie:

Gdy program S *rozpoczyna się w stanie spełniającym warunek wstępny* φ
to zakończy działanie w stanie spełniającym warunek końcowy ψ

$$\models [\varphi]\, S\, [\psi]$$
wttw
$$\{\varphi\} \subseteq [\![S]\!]\, \{\psi\}$$

gdzie dla $S \in \mathbf{Stmt}$, $A \subseteq \mathbf{State}$ zachodzi:

$$[\![S]\!]\, A = \{s \in \mathbf{State} \mid \mathcal{S}[\![S]\!]\, s = a, \text{dla pewnego } a \in A\}$$

Rozwińmy tę własność:

Warunek całkowitej poprawności $[\varphi]\, S\, [\psi]$ zachodzi (co zapisujemy jako $\models [\varphi]\, S\, [\psi]$), gdy dla wszystkich stanów $s \in \mathbf{State}$

$$\text{jeśli } \mathcal{F}[\![\varphi]\!]\, s = \mathbf{tt} \text{ to } \mathcal{S}[\![S]\!]\, s \in \mathbf{State} \text{ i } \mathcal{F}[\![\psi]\!]\, (\mathcal{S}[\![S]\!]\, s) = \mathbf{tt}$$

Poprawność całkowita: reguły dowodzenia

$$\frac{}{[\varphi[x \mapsto e]]\ x := e\ [\varphi]}$$

$$\frac{}{[\varphi]\ \textbf{skip}\ [\varphi]}$$

$$\frac{[\varphi]\ S_1\ [\theta] \quad [\theta]\ S_2\ [\psi]}{[\varphi]\ S_1; S_2\ [\psi]}$$

$$\frac{[\varphi \wedge b]\ S_1\ [\psi] \quad [\varphi \wedge \neg b]\ S_2\ [\psi]}{[\varphi]\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ [\psi]}$$

$$\frac{???}{[???]\ \textbf{while}\ b\ \textbf{do}\ S\ [???]}$$

$$\frac{\varphi' \Rightarrow \varphi \quad [\varphi]\ S\ [\psi] \quad \psi \Rightarrow \psi'}{[\varphi']\ S\ [\psi']}$$

Jeśli wyrażenia mogą powodować błędy, to są tu konieczne pewne poprawki!

Reguła całkowitej poprawności dla pętli

$$\frac{(nat(l) \wedge \varphi(l+1)) \Rightarrow b \qquad [nat(l) \wedge \varphi(l+1)]\ S\ [\varphi(l)] \qquad \varphi(0) \Rightarrow \neg b}{[\exists l. nat(l) \wedge \varphi(l)]\ \textbf{while}\ b\ \textbf{do}\ S\ [\varphi(0)]}$$

gdzie

- $\varphi(l)$ jest formułą ze zmienną wolną l nie występującą w **while** b **do** S,
- $nat(l)$ oznacza $0 \leqslant l$ oraz
- $\varphi(l+1)$ i $\varphi(0)$ powstają przez podstawienie za l w $\varphi(l)$. odpowiednio $l+1$ i 0

Nieformalnie: l jest *licznikiem* określającym liczbę wykonań ciała pętli

Poprawność

(**reguł dowodzenia poprawności całkowitej dla instrukcji języka** TINY)

jeśli $\mathcal{TH}(\mathbf{Int}) \vdash [\varphi]\, S\, [\psi]$ to $\models [\varphi]\, S\, [\psi]$

Dowód: Przez indukcję po strukturze drzewa dowodu: wszystkie przypadki dowodzi się jak przy poprawności częściowej, z wyjątkiem reguły dla pętli.

reguła dla pętli: Rozważmy $s \in \{nat(l) \wedge \varphi(l)\}$. Przez indukcję po $s(l)$ (to jest liczba naturalna) pokazujemy, że $\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\ s = s'$ dla pewnego $s' \in \{\varphi(0)\}$ (łatwe!).
Aby zakończyć dowód, wystarczy teraz zauważyć, że jeśli zmienna x nie występuje w instrukcji $S' \in \mathbf{Stmt}$ to dla dowolnych stanów, różniących się od siebie co najwyżej wartością dla x, instrukcja S' wykonana dla jednego z nich jako stanu początkowego zakończy działanie wtedy i tylko wtedy, gdy zakończy działanie także dla każdego innego z tych stanów, a przy tym, otrzymane stany końcowe różnią się co najwyżej wartością dla x.

Pełność

(**systemu dowodzenia poprawności całkowitej dla instrukcji języka** TINY)

Zachodzi:

$$\boxed{\mathcal{TH}(\mathbf{Int}) \vdash [\varphi]\, S\, [\psi]} \quad \text{wtw} \quad \boxed{\models [\varphi]\, S\, [\psi]}$$

Dowód (szkic): Tylko przypadek pętli jest istotnie nowy: tutaj jako $\varphi(I)$ należy przyjąć koniunkcję niezmiennika pętli (jak w poprawności częściowej) oraz formuły:

„pętla kończy się po dokładnie I *obrotach"*

Tak się składa, że ten warunek daje się tu wyrazić (gdyż skończone krotki liczb całkowitych i złożone z nich ciągi skończonej długości można zakodować jako liczby naturalne)!

Przykład

Aby udowonić:

$$[n \geqslant 0 \wedge rt = 0 \wedge sqr = 1]$$
$$\textbf{while } sqr \leqslant n \textbf{ do}$$
$$rt := rt + 1;\ sqr := sqr + 2 * rt + 1$$
$$[rt^2 \leqslant n \wedge n < (rt+1)^2]$$

stosujemy następujący niezmiennik z licznikiem obrotów l:

$$sqr = (rt+1)^2 \wedge rt^2 \leqslant n \wedge l = \lfloor\sqrt{n}\rfloor - rt$$

Małe oszustwo: „$l = \lfloor\sqrt{n}\rfloor - rt$"
trzeba wyrazić formułą pierwszego rzędu
w języku operacji i predykatów w TINY

Na szczęście: da się to zrobić!

A nawet jest to dość łatwe:
$(rt + l)^2 \leqslant n < (rt + l + 1)^2$

Relacja $\succ \subseteq W \times W$ jest *dobrze ufundowana* jeśli nie istnieje nieskończony łańcuch

$$a_0 \succ a_1 \succ \ldots \succ a_i \succ a_{i+1} \succ \ldots$$

Standardowy przykład:

$\langle \mathbf{Nat}, > \rangle$

Na marginesie: Przechodnio-zwrotne domknięcie $\succ^* \subseteq W \times W$ relacji dobrze ufundowanej $\succ \subseteq W \times W$ jest porządkiem częściowym w W.
Ale: odjęcie identyczności z porządku częściowego na W nie musi być relacją dobrze ufundowaną.

Kilka przykładów:

- **Nat**^{n} ze (ścisłym) porządkiem po współrzędnych;
- A^* uporządkowana relacją bycia właściwym prefiksem;
- **Nat*** ze (ścisłym) porządkiem leksykograficznym generowanym przez standardowy porządek na **Nat**;
- dowolna liczba porządkowa ze standardowym (ścisłym) porządkiem;itd.

Poprawność całkowita = poprawność częściowa + własność stopu

Metoda dowodzenia

Aby pokazać

$$[\varphi]\ \textbf{while}\ b\ \textbf{do}\ S\ [\varphi \land \neg b]$$

- udowodnij „poprawność częściową" tej pętli: $[\varphi \land b]\ S\ [\varphi]$
- pokaż „własność stopu" tej pętli: znajdź zbiór W z relacją dobrze ufundowaną $\succ \subseteq W \times W$ oraz funkcję $w\colon \textbf{State} \to W$ taką, że dla wszystkich stanów $s \in \{\varphi \land b\}$,

$$w(s) \succ w(\mathcal{S}[\![S]\!]\ s)$$

UWAGA: dopuszczając częściowość funkcji w, wymagamy jej określoności na $\{\varphi\}$.

Przykład

Udowodnij:

$$
\begin{array}{l}
[x \geqslant 0 \wedge y \geqslant 0] \\
\quad \textbf{while } x > 0 \textbf{ do} \\
\qquad \textbf{if } y > 0 \textbf{ then } y := y - 1 \textbf{ else } (x := x - 1; y := f(x)) \\
[\textbf{true}]
\end{array}
$$

gdzie f daje w wyniku liczbę naturalną dla każdej naturalnej wartości argumentu.

- Jeśli nic nie wiadomo o f, to przedstawiona poprzednio reguła dowodzenia poprawności całkowitej pętli (z licznikami) jest bezużyteczna.
- Ale: warunek stopu można łatwo udowodnić stosując funkcję $w : \textbf{State} \to \textbf{Nat} \times \textbf{Nat}$, gdzie $w(s) = \langle s\, x, s\, y \rangle$:
  po każdym wykonaniu ciała pętli wartość w zmniejsza się w sensie (dobrze ufundowanego) porządku leksykograficznego na parach liczb naturalnych.

W pełni wyspecyfikowany program

$[x \geqslant 0 \wedge y \geqslant 0]$
while $[x \geqslant 0 \wedge y \geqslant 0]$ $x > 0$ **do** **decr** $\langle x, y \rangle$ **in** **Nat** $\times$ **Nat** **wrt** $\succ$
  if $y > 0$ **then** $y := y - 1$ **else** $(x := x - 1; y := f(x))$
[**true**]

...z różnymi wariantami notacyjnymi;
zakładamy zewnętrzne definicje zbioru
dobrze ufundowanego i funkcji w ten zbiór

Logika Hoare'a: problem #2

Zbuduj instrukcję S, taką że

$$\{n \geqslant 0\}\ S\ \{rt^2 \leqslant n \wedge n < (rt+1)^2\}$$

Kolejne poprawne rozwiązanie:

$$\{n \geqslant 0\}$$
$$rt := 0;\ n := 0$$
$$\{rt^2 \leqslant n \wedge n < (rt+1)^2\}$$

OOOOPS?!

Jest kilka technik radzenia sobie z tym problemem:

- zmienne, które nie mogą wystąpić w programie;
- binarne warunki końcowe;
- różne postaci logiki algorytmicznej/dynamicznej, z modalnościami.

Szkic

- Nowa kategoria składniowa **BForm** *formuł binarnych*, które są zwykłymi formułami z tą różnicą, że można w nich używać zarówno zwykłych zmiennych $x \in$ **Var** jak i ich kopii „z przeszłości" $\widehat{x} \in \widehat{\textbf{Var}}$. Dla każdej frazy składniowej ω, oznaczmy przez $\widehat{\omega}$ frazę powstałą przez zastąpienie każdej zmiennej x w ω przez $\widehat{x}$.
- Funkcja semantyczna: $\mathcal{BF} \colon \textbf{BForm} \to \textbf{State} \times \textbf{State} \to \textbf{Bool}$
  $\mathcal{BF}[\![\psi]\!] \langle s_0, s \rangle$ definiuje się w standardowy sposób, ale do określenia wartości zmiennych „z przeszłości" $\widehat{x} \in \widehat{\textbf{Var}}$ stosuje się stan s_0, a do określenia wartości zwykłych zmiennych $x \in$ **Var** korzysta się ze stanu s.

Warunki poprawności

$$pre\ \varphi;\ S\ \ post\ \psi$$

gdzie $\varphi \in$ **Form** jest (unarnym) warunkiem wstępnym, $S \in$ **Stmt** jest instrukcją (jak zwykle), a $\psi \in$ **BForm** jest binarnym warunkiem końcowym.

Semantyka:

Warunek $pre\ \varphi;\ S\ \ post\ \psi$ zachodzi, co zapisujemy jako $\models pre\ \varphi;\ S\ \ post\ \psi$, gdy dla wszystkich stanów $s \in$ **State**

$$\text{jeśli } \mathcal{F}[\![\varphi]\!]\ s = \mathbf{tt} \text{ to } \mathcal{S}[\![S]\!]\ s \in \mathbf{State} \text{ oraz } \mathcal{BF}[\![\psi]\!]\ \langle s, \mathcal{S}[\![S]\!]\ s\rangle = \mathbf{tt}$$

$$\frac{}{pre\ \varphi;\ x := e\ post\ (\widehat{\varphi} \land x = \widehat{e} \land \vec{y} = \widehat{\vec{y}})}$$

gdzie $\vec{y}$ są zmiennymi różnymi od x.

$$\frac{}{pre\ \varphi;\ \textbf{skip}\ post\ (\varphi \land \vec{y} = \widehat{\vec{y}})}$$

$$\frac{pre\ \varphi_1;\ S_1\ post\ (\psi_1 \land \varphi_2) \qquad pre\ \varphi_2;\ S_2\ post\ \psi_2}{pre\ \varphi_1;\ S_1; S_2\ post\ \psi_1 * \psi_2}$$

gdzie $\psi_1 * \psi_2$ to $\exists \vec{z}.(\psi_1[\vec{x} \mapsto \vec{z}] \land \psi_2[\widehat{\vec{x}} \mapsto \vec{z}])$, przy czym wszystkie zmienne wolne w ψ_1 lub ψ_2 są wśród $\vec{x}$ lub $\widehat{\vec{x}}$, a $\vec{z}$ są nowymi zmiennymi.

Dalsze reguły

$$\frac{\textit{pre}\ \varphi \wedge b;\ S_1\ \textit{post}\ \psi \qquad \textit{pre}\ \varphi \wedge \neg b;\ S_2\ \textit{post}\ \psi}{\textit{pre}\ \varphi;\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textit{post}\ \psi}$$

$$\frac{\textit{pre}\ \varphi \wedge b;\ S\ \textit{post}\ (\psi \wedge \widehat{e} \succ e) \qquad \psi \Rightarrow \varphi \qquad (\psi * \psi) \Rightarrow \psi}{\textit{pre}\ \varphi;\ \textbf{while}\ b\ \textbf{do}\ S\ \textit{post}\ ((\psi \vee (\varphi \wedge \vec{y} = \widehat{\vec{y}})) \wedge \neg b)}$$

gdzie $\succ$ jest relacją dobrze ufundowaną i wszystkie zmienne wolne są wśród $\vec{y}$ lub $\widehat{\vec{y}}$.

$$\frac{\varphi' \Rightarrow \varphi \qquad \textit{pre}\ \varphi;\ S\ \textit{post}\ \psi \qquad \psi \Rightarrow \psi'}{\textit{pre}\ \varphi';\ S\ \textit{post}\ \psi'}$$

$$\frac{\textit{pre}\ \varphi;\ S\ \textit{post}\ \psi}{\textit{pre}\ \varphi;\ S\ \textit{post}\ (\widehat{\varphi} \wedge \psi)}$$

Reguły można (trzeba?) dopracować...

Przykład

Mamy teraz:

$$\models \begin{array}{l} \mathit{pre}\ n \geqslant 0; \\ \quad rt := 0; sqr := 1 \\ \quad \textbf{while}\ sqr \leqslant n\ \textbf{do}\ rt := rt + 1; sqr := sqr + 2 * rt + 1 \\ \mathit{post}\ rt^2 \leqslant \widehat{n} \wedge \widehat{n} < (rt+1)^2 \end{array}$$

$$\mathit{Ale}: \quad \not\models \begin{array}{l} \{n \geqslant 0\} \\ \quad rt := 0; n := 0 \\ \{rt^2 \leqslant \widehat{n} \wedge \widehat{n} < (rt+1)^2\} \end{array}$$

Szkic

Ogólna idea:

Rozszerzamy zbiór formuł logicznych tak, aby był domknięty na zwykłe spójniki logiczne i kwantyfikatory oraz na operatory modalne

Składnia: Dla dowolnej formuły φ i instrukcji $S \in$ **Stmt**, tworzymy nową formułę:

$$\langle S \rangle \varphi$$

Semantyka: $\mathcal{F}[\![\langle S \rangle \varphi]\!]\, s = \begin{cases} \mathcal{F}[\![\varphi]\!]\, s' & \text{jeśli } \mathcal{S}[\![S]\!]\, s = s' \in \textbf{State} \\ \textbf{ff} & \text{jeśli } \mathcal{S}[\![S]\!]\, s \notin \textbf{State} \end{cases}$

System dowodzenia

. . . aksjomaty i reguły dotyczące
standardowych spójników i kwantyfikatorów . . .

Plus aksjomaty i reguły dla operatorów modalnych — związki między operatorami modalnymi i spójnikami zdaniowymi; (de)kompozycja operatorów modalnych — na przykład:

$$\langle S\rangle(\varphi \wedge \psi) \iff (\langle S\rangle\varphi \wedge \langle S\rangle\psi)$$

$$\langle S\rangle\neg\varphi \implies \neg\langle S\rangle\varphi$$

$$\langle S\rangle\mathbf{true} \implies (\neg\langle S\rangle\varphi \implies \langle S\rangle\neg\varphi)$$

$$\langle S_1; S_2\rangle\varphi \iff \langle S_1\rangle(\langle S_2\rangle\varphi)$$

itd.

Kluczowe do uzyskania pełności są tu *nieskończone reguły dla pętli*

Podstawowe pojęcia algebry uniwersalnej:

- sygnatury i algebry
- homomorfizmy, podalgebry, kongruencje
- równości i rozmaitości
- rachunek równościowy
- specyfikacje równościowe i algebry początkowe
- warianty: algebry częściowe, struktury pierwszego rzędu

Oraz wzmianka o zastosowaniach w

podstawach semantyki oprogramowania, weryfikacji, specyfikacji, konstruowaniu programów, ...

Typ danych TINY

Jego *sygnatura* Σ (składnia):

$$
\begin{array}{ll}
\textbf{sorts} & Int, Bool; \\
\textbf{opns} & 0, 1 \colon Int; \\
 & plus, times, minus \colon Int \times Int \to Int; \\
 & false, true \colon Bool; \\
 & lteq \colon Int \times Int \to Bool; \\
 & not \colon Bool \to Bool; \\
 & and \colon Bool \times Bool \to Bool;
\end{array}
$$

oraz Σ-*algebra* $\mathcal{A}$ (semantyka):

$$
\begin{array}{ll}
\textbf{carriers} & \mathcal{A}_{Int} = \textbf{Int}, \mathcal{A}_{Bool} = \textbf{Bool} \\
\textbf{operations} & 0_{\mathcal{A}} = 0, 1_{\mathcal{A}} = 1 \\
 & plus_{\mathcal{A}}(n, m) = n + m, times_{\mathcal{A}}(n, m) = n * m \\
 & minus_{\mathcal{A}}(n, m) = n - m \\
 & false_{\mathcal{A}} = \textbf{ff}, true_{\mathcal{A}} = \textbf{tt} \\
 & lteq_{\mathcal{A}}(n, m) = \textbf{tt if } n \leqslant m \textbf{ else ff} \\
 & not_{\mathcal{A}}(b) = \textbf{tt if } b = \textbf{ff else ff} \\
 & and_{\mathcal{A}}(b, b') = \textbf{tt if } b = b' = \textbf{tt else ff}
\end{array}
$$

Sygnatury

Sygnatury algebraiczne:

$$\Sigma = (S, \Omega)$$

- *nazwy sortów*: S
- *nazwy operacji rozdzielone względem arności i sortu wyniku*:
  $\Omega = \langle \Omega_{w,s} \rangle_{w \in S^*, s \in S}$

Alternatywnie:

$$\Sigma = (S, \Omega, arity, sort)$$

z *nazwami sortów* S, *nazwami operacji* Ω, oraz *nazwami operacji z funkcjami przypisującymi im arność i sort wyniku*

$$arity : \Omega \to S^* \text{ oraz } sort : \Omega \to S.$$

- $f : s_1 \times \ldots \times s_n \to s$ oznacza $s_1, \ldots, s_n, s \in S$ i $f \in \Omega_{s_1 \ldots s_n, s}$

Porównaj te dwie definicje

Algebras

Ustalmy na chwilę sygnaturę $\Sigma = (S, \Omega)$.

- Σ-*algebra*:

$$A = (|A|, \langle f_A \rangle_{f \in \Omega})$$

  - *zbiory nośników:* $|A| = \langle |A|_s \rangle_{s \in S}$
  - *operacje:*: $f_A \colon |A|_{s_1} \times \ldots \times |A|_{s_n} \to |A|_s$, dla $f \colon s_1 \times \ldots \times s_n \to s$
- klasa wszystkich Σ-algebr:

$$\mathbf{Alg}(\Sigma)$$

Czy $\mathbf{Alg}(\Sigma)$ może być pusta? Skończona?
Czy $A \in \mathbf{Alg}(\Sigma)$ może mieć puste nośniki?

Podalgebry

- Σ-*podalgebra* algebry $A \in \mathbf{Alg}(\Sigma)$, $A_{sub} \subseteq A$ jest określona przez podzbiór $|A_{sub}| \subseteq |A|$ domknięty na operacje:
  - dla $f : s_1 \times \ldots \times s_n \to s$ i $a_1 \in |A_{sub}|_{s_1}, \ldots, a_n \in |A_{sub}|_{s_n}$,
  $$f_{A_{sub}}(a_1, \ldots, a_n) = f_A(a_1, \ldots, a_n)$$
- dla $A \in \mathbf{Alg}(\Sigma)$ i $X \subseteq |A|$, *podalgebrą* A *generowaną przez* X, $\langle A \rangle_X$, jest najmniejsza podalgebra A zawierająca X.
- $A \in \mathbf{Alg}(\Sigma)$ jest *osiągalna* jeśli $\langle A \rangle_\emptyset$ pokrywa się z A.

Twierdzenie:

Dla każdej $A \in \mathbf{Alg}(\Sigma)$ *i każdego* $X \subseteq |A|$, *istnieje* $\langle A \rangle_X$.

Dowód (szkic):

- utwórz podalgebrę generowaną przez zbiór X domykając go na operacje z A; lub
- przecięcie rodziny podalgebr algebry A jest podalgebrą algebry A.

- Σ-*homomorfizmem* algebr $A, B \in \mathbf{Alg}(\Sigma)$, $h\colon A \to B$ jest funkcja $h\colon |A| \to |B|$, która zachowuje operacje:
  - dla $f : s_1 \times \ldots \times s_n \to s$ i $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$,
    $h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$

Twierdzenie:

Niech $h\colon A \to B$ będzie homomorfizmem, A_{sub} podalgebrą A, a B_{sub} podalgebrą B. Wówczas obraz A_{sub} poprzez h, $h(A_{sub})$, jest podalgebrą algebry B, a przeciwobraz B_{sub} poprzez h, $h^{-1}(B_{sub})$, jest podalgebrą algebry A.

Twierdzenie:

Niech $h\colon A \to B$ będzie homomorfizmem i $X \subseteq |A|$. Wówczas $h(\langle A\rangle_X) = \langle B\rangle_{h(X)}$.

Twierdzenie:

Funkcja identyczności określona na nośniku $A \in \mathbf{Alg}(\Sigma)$ jest homomorfizmem $id_A\colon A \to A$. Złożenie homomorfizmów $h\colon A \to B$ i $g\colon B \to C$ jest homomorfizmem $h;g\colon A \to C$.

Izomorfizmy

- Σ-*izomorfizmem* algebr $A, B \in \mathbf{Alg}(\Sigma)$, jest dowolny Σ-homomorfizm $i\colon A \to B$, który ma *funkcję odwrotną*, czyli Σ-homomorphism $i^{-1}\colon B \to A$ taki, że $i;i^{-1} = id_A$ oraz $i^{-1};i = id_B$.
- Σ-algebry są *izomorficzne* jeśli istnieje między nimi izomorfizm.

Twierdzenie:

Σ-homomorfizm jest Σ-izomorfizmem wttw, gdy jest bijekcją („1-1" i „na").

Twierdzenie:

Identyczności są izomorfizmami i złożenie izomorfizmów jest izomorfizmem.

Kongruencje

- niech $A \in \mathbf{Alg}(\Sigma)$. Σ-*kongruencją na* A jest relacja równoważności $\equiv \subseteq |A| \times |A|$ domknięta na operacje:
  - dla $f : s_1 \times \ldots \times s_n \to s$ i $a_1, a_1' \in |A|_{s_1}, \ldots, a_n, a_n' \in |A|_{s_n}$,
    jeśli $a_1 \equiv_{s_1} a_1', \ldots, a_n \equiv_{s_n} a_n'$ to $f_A(a_1, \ldots, a_n) \equiv_s f_A(a_1', \ldots, a_n')$.

Twierdzenie:

Dla dowolnej relacji $R \subseteq |A| \times |A|$ określonej na nośniku Σ-algebry A, istnieje najmniejsza kongruencja w A, zawierająca R.

Twierdzenie:

Dla dowolnego Σ-homomorfizmu $h\colon A \to B$, jądro h, $K(h) \subseteq |A| \times |A|$, gdzie $a\ K(h)\ a'$ wttw, gdy $h(a) = h(a')$, jest Σ-kongruencją na A.

Algebry ilorazowe

- dla dowolnej $A \in \mathbf{Alg}(\Sigma)$ i Σ-kongruencji $\equiv \subseteq |A| \times |A|$ na A, określamy w naturalny sposób *algebrę ilorazową* $A/{\equiv}$, której nośnikiem są klasy abstrakcji relacji $\equiv$:
  - dla $s \in S$, $|A/{\equiv}|_s = \{[a]_{\equiv} \mid a \in |A|_s\}$, z $[a]_{\equiv} = \{a' \in |A|_s \mid a \equiv a'\}$
  - dla $f \colon s_1 \times \ldots \times s_n \to s$ i $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$,
  $$f_{A/\equiv}([a_1]_{\equiv}, \ldots, [a_n]_{\equiv}) = [f_A(a_1, \ldots, a_n)]_{\equiv}$$

Twierdzenie:

Powyższa definicja jest poprawna; ponadto naturalne odwzorowanie, które każdemu elementowi przypisuje jego klasę abstrakcji jest Σ-homomorfizmem $[_]_{\equiv} \colon A \to A/{\equiv}$.

Twierdzenie:

Niech $\equiv$ i $\equiv'$ będą Σ-kongruencjami na A. Wówczas $\equiv \subseteq \equiv'$ wttw, gdy istnieje Σ-homomorfizm $h \colon A/{\equiv} \to A/{\equiv'}$ taki, że $[_]_{\equiv};h = [_]_{\equiv'}$.

Twierdzenie:

Dla dowolnego Σ-homomorphism $h \colon A \to B$, $A/K(h)$ jest izomorficzna z $h(A)$.

Produkty

- dla dowolnej rodziny $A_i \in \mathbf{Alg}(\Sigma)$, $i \in \mathcal{I}$ określamy w naturalny sposób *algebrę produktową*, $\prod_{i \in \mathcal{I}} A_i$, określoną na iloczynie kartezjańskim nośników A_i, $i \in \mathcal{I}$:
  - dla $s \in S$, $|\prod_{i \in \mathcal{I}} A_i|_s = \prod_{i \in \mathcal{I}} |A_i|_s$
  - dla $f : s_1 \times \ldots \times s_n \to s$ i $a_1 \in |\prod_{i \in \mathcal{I}} A_i|_{s_1}, \ldots, a_n \in |\prod_{i \in \mathcal{I}} A_i|_{s_n}$, dla $i \in \mathcal{I}$, $f_{\prod_{i \in \mathcal{I}} A_i}(a_1, \ldots, a_n)(i) = f_{A_i}(a_1(i), \ldots, a_n(i))$

Twierdzenie:

Dla dowolnej rodziny $\langle A_i \rangle_{i \in \mathcal{I}}$ Σ-algebr, rzuty $\pi_i(a) = a(i)$, gdzie $i \in \mathcal{I}$ oraz $a \in \prod_{i \in \mathcal{I}} |A_i|$, są Σ-homomorfizmami $\pi_i : \prod_{i \in \mathcal{I}} A_i \to A_i$.

Zdefiniuj produkt pustej rodziny Σ-algebr.
Kiedy rzut π_i jest izomorfizmem?

Rozważmy S-sortowy zbiór X zmiennych.

- *termy* $t \in |T_\Sigma(X)|$ tworzy się w zwykły sposób używając zmiennych X, stałych i operacji z Ω: $|T_\Sigma(X)|$ jest najmniejszym zbiorem takim, że:
  - $X \subseteq |T_\Sigma(X)|$
  - dla $f : s_1 \times \ldots \times s_n \to s$ oraz $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$, $f(t_1, \ldots, t_n) \in |T_\Sigma(X)|_s$
- dla dowolnej Σ-algebra A i dowolnego wartościowania $v : X \to |A|$, *wartość* $t_A[v]$ *termu* $t \in |T_\Sigma(X)|$ *w* A *przy* v określa się indukcyjnie:
  - $x_A[v] = v_s(x)$, dla $x \in X_s$, $s \in S$
  - $(f(t_1, \ldots, t_n))_A[v] = f_A((t_1)_A[v], \ldots, (t_n)_A[v])$, dla $f : s_1 \times \ldots \times s_n \to s$ i $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$

Tu i w dalszym ciągu wykładu zakładamy, że „składnia” termów jest jednoznaczna!

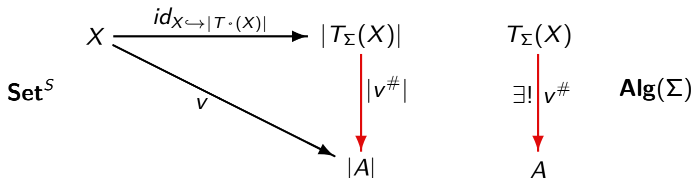
Algebry termów

Rozważmy S-sortowy zbiór X zmiennych.

- Nośnikiem *algebry termów* $T_\Sigma(X)$ jest zbiór termów, a operacje określa się w niej „składniowo”:
  - dla $f\colon s_1 \times \ldots \times s_n \to s$ i $t_1 \in |T_\Sigma(X)|_{s_1}, \ldots, t_n \in |T_\Sigma(X)|_{s_n}$, $f_{T_\Sigma(X)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$.

Twierdzenie:

Dla dowolnego S-sortowego zbioru X zmiennych, Σ-algebry A oraz wartościowania $v\colon X \to |A|$, istnieje dokładnie jeden Σ-homomorfizm $v^\#\colon T_\Sigma(X) \to A$, który rozszerza v. Dla dowolnego $t \in |T_\Sigma(X)|$ zachodzi ponadto $v^\#(t) = t_A[v]$.



Równości

- *Równość*:

$$\boxed{\forall X . t = t'}$$

gdzie:

  - X jest zbiorem zmiennych
  - $t, t' \in |T_\Sigma(X)|_s$ są termami tego samego sortu.

- *Relacja spełnialności*: Σ-algebra A *spełnia* $\forall X . t = t'$

$$\boxed{A \models \forall X . t = t'}$$

jeśli dla wszystkich $v \colon X \to |A|$, $t_A[v] = t'_A[v]$.

Wynikanie semantyczne

$$\Phi \models_\Sigma \varphi$$

Σ*-równość* φ *jest semantyczną konsekwencją zbioru* Σ*-równości* Φ*,*
jeśli φ *zachodzi w każdej* Σ*-algebrze, która spełnia* Φ*.*

Na marginesie:

- *Modele* zbioru równości: $Mod(\Phi) = \{A \in \mathbf{Alg}(\Sigma) \mid A \models \Phi\}$
- *Teoria* klasy algebr: $Th(\mathcal{C}) = \{\varphi \mid \mathcal{C} \models \varphi\}$
- $\Phi \models \varphi \iff \varphi \in Th(Mod(\Phi))$
- *Mod* and *Th* tworzą *powiązanie Galois*

Rachunek równościowy

$$\frac{}{\forall X.t = t} \qquad \frac{\forall X.t = t'}{\forall X.t' = t} \qquad \frac{\forall X.t = t' \quad \forall X.t' = t''}{\forall X.t = t''}$$

$$\frac{\forall X.t_1 = t_1' \quad \dots \quad \forall X.t_n = t_n'}{\forall X.f(t_1 \dots t_n) = f(t_1' \dots t_n')} \qquad \frac{\forall X.t = t'}{\forall Y.t[\theta] = t'[\theta]} \text{dla } \theta \colon X \to |T_\Sigma(Y)|$$

Uwage na zmienne!

Z $a = f(x)$ i $f(x) = b$ *nie* wynika $a = b$, chyba że ...

$$\Phi \vdash_\Sigma \varphi$$

Σ-*równość* φ *jest kosekwencją dowodową zbioru* Σ-*równości* Φ*,*
jeśli φ *można wyprowadzić za pomocą reguł z* Φ*.*

Jak to uzasadnić?

Semantyka!

Twierdzenie:

Rachunek równościowy jest poprawny i pełny:

$$\Phi \models \varphi \iff \Phi \vdash \varphi$$

- poprawność: „wszystko, co daje się udowodnić, jest prawdziwe” ($\Phi \models \varphi \Longleftarrow \Phi \vdash \varphi$)
- pełność: „wszystko, co prawdziwe, daje się udowodnić” ($\Phi \models \varphi \Longrightarrow \Phi \vdash \varphi$)

Dowód (szkic):

- poprawność: łatwe!
- pełność: nie takie łatwe!

Motywacje

> *Oprogramowanie (typy danych, moduły, programy, bazy danych...):*
> *zbiory danych z operacjami na nich*

- Pomijamy: kod, efektywność, odporność, niezawodność, ...
- Skupiamy się na: POPRAWNOŚCI

Algebra uniwersalna
analogia:

interfejs modułu	$\rightsquigarrow$ sygnatura
moduł	$\rightsquigarrow$ algebra
specyfikacja modułu	$\rightsquigarrow$ klasa algebr

$$\langle \Sigma, \Phi \rangle$$

- sygnatura Σ określa statyczny interfejs modułu
- aksjomaty (Σ-równości) określają pożądane własności modułu

ALE:

Twierdzenie:

Klasa Σ-algebr jest definiowalna równościowo wttw, gdy jest domknięta na podalgebry, produkty i obrazy homomorficzne.

Specyfikacje równościowe zazwyczaj dopuszczają wiele niepożądanych „modułów”

Przykład

$$
\begin{array}{l}
\textbf{spec}\ \textsc{NaiveNat} = \textbf{sort}\ Nat \\
\quad \textbf{opns}\ 0 \colon Nat; \\
\qquad succ \colon Nat \to Nat; \\
\qquad _ + _ \colon Nat \times Nat \to Nat \\
\quad \textbf{axioms}\ \forall n{:}Nat . n + 0 = n; \\
\qquad \forall n, m{:}Nat . n + succ(m) = succ(n + m)
\end{array}
$$

Teraz:

$$\textsc{NaiveNat} \not\models \forall n, m{:}Nat . n + m = m + n$$

Co gorsza:

Istnieją modele $M \in Mod(\textsc{NaiveNat})$ takie, że $M \models 0 = succ(0)$, a nawet:

$$M \models \forall n, m{:}Nat . n = m$$

Jak to naprawić

- *Ograniczenie klasy modeli*:

  początkowość: „no junk" & „no confusion"

  Także: *osiągalność* („no junk") i jej bardziej ogólne odmiany (algebry wolne, algebry generowane).

  Na marginesie: Takie ograniczenia można traktować jak specjalne formuły (wyższego rzędu).

- Inne (silniejsze) *systemy logiczne*: równości warunkowe, logika pierwszego rzędu, logiki wyższych rzędów, inne wodotryski
  - więcej na ten temat – gdzieindziej. . .

Instytucje!

Nastąpił wysyp różnych systemów logicznych. . .

Modele początkowe

Twierdzenie:

Każda specyfikacja równościowa $\langle \Sigma, \Phi \rangle$ *ma model początkowy: istnieje* Σ*-algebra* $I \in Mod(\Phi)$ *taka, że dla każdej* Σ*-algebry* $M \in Mod(\Phi)$ *istnieje dokładnie jeden* Σ*-homomorfizm z* I *w* M*.*

Dowód (szkic):

- I jest algebrą ilorazową Σ-termów bez zmiennych poprzez kongruencję, która skleja ze sobą wszystkie termy t, t' takie, że $\Phi \models \forall\emptyset. t = t'$.
- I jest osiągalną podalgebrą produktu „wszystkich" (z dokładnością do izomorfizmu) osiągalnych algebr z $Mod(\Phi)$.

Na marginesie: Można to uogólnić do twierdzenia o istnieniu modelu wolnego $\langle \Sigma, \Phi \rangle$ nad dowolnym (wielosortowym) zbiorem danych.

Na marginesie: Istnienie modelu początkowego (i wolnego) przenosi się na specyfikacje z równościami warunkowymi, **ale niewiele dalej!**

Example

$$\textbf{spec}\ \textsc{Nat} = \textbf{initial}\ \{\ \begin{array}{l} \textbf{sort}\ \mathit{Nat} \\ \textbf{opns}\ 0\colon \mathit{Nat}; \\ \qquad \mathit{succ}\colon \mathit{Nat} \to \mathit{Nat}; \\ \qquad _+_\colon \mathit{Nat} \times \mathit{Nat} \to \mathit{Nat} \\ \textbf{axioms}\ \forall n{:}\mathit{Nat}.n + 0 = n; \\ \qquad \forall n, m{:}\mathit{Nat}.n + \mathit{succ}(m) = \mathit{succ}(n + m) \end{array} \}$$

Mamy:

$$\textsc{Nat} \models \forall n, m{:}\mathit{Nat}.n + m = m + n$$

Inny przykład

$$
\begin{array}{ll}
\textbf{spec}\ \textsc{NatPred} = & \textbf{sort}\ Nat \\
& \textbf{opns}\ 0\colon Nat; error\colon Nat; \\
& \quad succ\colon Nat \to Nat; \\
& \quad _+_\colon Nat \times Nat \to Nat; \\
& \quad pred\colon Nat \to Nat \\
& \textbf{axioms}\ \forall n{:}Nat.n + 0 = n; \\
& \quad \forall n, m{:}Nat.n + succ(m) = succ(n+m); \\
& \quad \forall n{:}Nat.pred(succ(n)) = n; \\
& \quad pred(0) = error; \\
& \quad pred(error) = error; succ(error) = error; \\
& \quad \forall n{:}Nat.error + n = error; \\
& \quad \forall n{:}Nat.n + error = error
\end{array}
$$

Wygląda dobrze. Ale spróbujmy dodać mnożenie:

$$
\begin{array}{l}
0 * n = 0; succ(m) * n = n + (m * n); \\
error * n = error; n * error = error
\end{array}
$$

i wszystko się wali!

Algebry częściowe

- *Sygnatura algebraiczna* Σ: jak poprzednio
- Σ-*algebra częściowa*:

$$\boxed{A = (|A|, \langle f_A \rangle_{f \in \Omega})}$$

  jak poprzednio, ale operacje $f_A \colon |A|_{s_1} \times \ldots \times |A|_{s_n} \rightharpoonup |A|_s$, dla $f \colon s_1 \times \ldots \times s_n \to s$, mogą być teraz *funkcjami częściowymi*.

  Na marginesie: Również stałe mogą mieć niezdefiniowane wartości.

- **PAlg**(Σ) oznacza klasę wszystkich Σ-algebr częściowych.

Kolejne pojęcia

Ustalmy na chwilę sygnaturę $\Sigma = (S, \Omega)$.

- *podalgebra* $A_{sub} \subseteq A$: dana przez podzbiór $|A_{sub}| \subseteq |A|$ domknięty na operacje;
  (Na marginesie: można podać przynamniej dwa inne naturalne określenia)
- *homomorfizm* $h\colon A \to B$: odwzorowanie $h\colon |A| \to |B|$ zachowujące określoność i wyniki operacji; homomorfizm jest *silny* jeśli dodatkowo odzwierciedla określoność operacji; (silne) homomorfizmy są domknięte na składanie;
  (Na marginesie: bardzo ciekawa alternatywa: odwzorowanie *częściowe* $h\colon |A| \rightharpoonup |B|$ zachowujące wyniki operacji)
- *kongruencja* $\equiv$ na A: relacja równoważności $\equiv \subseteq |A| \times |A|$ domknięta na operacje (gdy są one określone); kongruencja jest *silna* jeśli dodatkowo odzwierciedla określoność operacji; (silne) kongruencje są jądrami (silnych) homomorfizmów;
- *algebra ilorazowa* $A/\equiv$: określana w naturalny sposób na klasach abstrakcji kongruencji $\equiv$; naturalny homomorfizm z A w $A/\equiv$ jest silny, gdy kongruencja jest silna.

Formuły

(Silna) równość:	*Określoność*:
$\forall X . t \stackrel{s}{=} t'$ jak poprzednio	$\forall X . def\ t$ gdzie X jest zbiorem zmiennych, a $t \in \lvert T_\Sigma(X)\rvert_s$ jest termem

Relacja spełnialności

Σ-algebra częściowa A *spełnia* $\forall X . t \stackrel{s}{=} t'$	Σ-algebra częściowa A *spełnia* $\forall X . def\ t$
$A \models \forall X . t \stackrel{s}{=} t'$	$A \models \forall X . def\ t$
jeśli dla każdego $v \colon X \to \lvert A\rvert$, $t_A[v]$ jest określony wttw, gdy $t'_A[v]$ jest określona, i wówczas $t_A[v] = t'_A[v]$	jeśli dla każdego $v \colon X \to \lvert A\rvert$, $t_A[v]$ jest określona

Inne podejście

- *Równość (z określonością)*:

$$\forall X. t \stackrel{e}{=} t'$$

gdzie:

- X jest zbiorem zmiennych
- $t, t' \in |T_\Sigma(X)|_s$ są termami tego samego sortu.

- *Relacja spełnialności*: Σ-algebra A *spełnia* $\forall X. t \stackrel{e}{=} t'$

$$A \models \forall X. t \stackrel{e}{=} t'$$

jeśli dla każdego $v \colon X \to |A|$, $t_A[v] = t'_A[v]$ — obie strony są określone i równe. equal.

Na marginesie:

- $\forall X. t \stackrel{e}{=} t'$ wttw, gdy $\forall X. (t \stackrel{s}{=} t' \wedge \mathit{def}\ t)$
- $\forall X. t \stackrel{s}{=} t'$ iff $\forall X. (\mathit{def}\ t \iff \mathit{def}\ t') \wedge (\mathit{def}\ t \implies t \stackrel{e}{=} t')$

Przykład

$$
\begin{array}{l}
\textbf{spec } \textsc{NatPred} = \textbf{initial } \{ \; \textbf{sort } \mathit{Nat} \\
\quad \textbf{opns } 0 : \mathit{Nat}; \\
\quad\quad \mathit{succ} : \mathit{Nat} \to \mathit{Nat}; \\
\quad\quad _+_ : \mathit{Nat} \times \mathit{Nat} \to \mathit{Nat}; \\
\quad\quad \mathit{pred} : \mathit{Nat} \to ? \; \mathit{Nat} \\
\quad \textbf{axioms } \forall n{:}\mathit{Nat}.\, n + 0 = n; \\
\quad\quad \forall n, m{:}\mathit{Nat}.\, n + \mathit{succ}(m) = \mathit{succ}(n + m); \\
\quad\quad \forall n{:}\mathit{Nat}.\, \mathit{pred}(\mathit{succ}(n)) \stackrel{s}{=} n \\
\}
\end{array}
$$

Struktury pierwszego rzędu

- *Sygnatura pierwszego rzędu* $\Sigma = (S, \Omega, \Pi)$: sygnatura (S, Ω) plus *nazwy predykatów*, rozdzielone ze względu na arności: $\Pi = \langle \Pi_w \rangle_{w \in S^*}$
- Σ-*struktura pierwszego rzędu*:

$$\boxed{A = (|A|, \langle f_A \rangle_{f \in \Omega}, \langle p_A \rangle_{p \in \Pi})}$$

  składa się z:
  - (S, Ω)-algebry $(|A|, \langle f_A \rangle_{f \in \Omega})$
  - *predykatów* (relacji): $p_A \subseteq |A|_{s_1} \times \ldots \times |A|_{s_n}$,
    dla $p\colon s_1 \times \ldots \times s_n$ (i.e., $p \in \Pi_{s_1 \ldots s_n}$)
- **Str**(Σ) oznacza klasę wszystkich Σ-struktur pierwszego rzędu.

Kolejne pojęcia

Ustalmy na chwilę sygnaturę $\Sigma = (S, \Omega, \Pi)$.

- *podstruktura* $A_{sub} \subseteq A$: dana przez podzbiór $|A_{sub}| \subseteq |A|$ domknięty na operacje i taki, że zawieranie zachowuje prawdziwość predykatów; podstruktura jest *zamknięta* jeśli zwierania zachowuje także fałszywość predykatów;
- *homomorfizm* $h \colon A \to B$: odwzorowanie $h \colon |A| \to |B|$ zachowujące wyniki operacji i prawdziwość predykatów; jest *zamknięte*, jeśli dodatkowo zachowuje fałszywość predykatów; (zamknięte) homomofizmy są domknięte na składanie;
- *kongruencja* $\equiv$on A: relacja równoważności $\equiv \subseteq |A| \times |A|$ domknięta na operacje; jest *zamknięta*, jeśli dodatkowo zachowuje prawdziwość (i fałszywość) predykatów; (zamknięte) kongruencje są jądrami (zamkniętych) homomorfizmów;
- *struktura ilorazowa* $A/\equiv$: określana w naturalny sposób na klasach abstrakcji kongruencji $\equiv$ tak, że naturalne odwzorowanie z A w $A/\equiv$ jest homomorfizmem; jest zamknięta, jeśli kongruencja jest zamknięta.

Formuły

- Σ-*formuły atomowe* nad zbiorem X zmiennych:
  - $t = t'$, gdzie $t, t' \in |T_{(S,\Omega)}(X)|_s$, $s \in S$
  - $p(t_1, \ldots t_n)$, gdzie $p\colon s_1 \times \ldots \times s_n$, $t_1 \in |T_{(S,\Omega)}(X)|_{s_1}, \ldots t_n \in |T_{(S,\Omega)}(X)|_{s_n}$
- Σ-*formulae* zawierają formuły atomowe i są domknięte na spójniki logiczne i kwantyfikatory; Σ-*zdania* są Σ-formułami bez zmiennych wolnych
- *Relacja spełnialności* definiowana jak zwykle między Σ-strukturami A a Σ-zdaniami φ

$$\boxed{A \models \varphi}$$

Jak poprzednio prowadzi to do pojęcia *klasy modeli* dla zbioru zdań, *wynikania semantycznego* zbioru zdań, *teorii* klasy modeli itd.

Model *początkowy* (i wolny) istnieje dla specyfikacji pierwszego rzędu z warunkowymi formułami atomowymi kwantyfikowanymi uniwersalnie, *ale w ogólności może on nie istnieć!*

Zamiast dziedziny **Int**, rozważmy dowolny typ danych określony przez

- Σ: sygnaturę algebraiczną zawierającą rodzaj *Bool*, stałe i spójniki logiczne
- $\mathcal{A}$: Σ-strukturę (nośnik wraz z operacjami o nazwach z Σ zdefiniowanymi na $|A|$) ze standardową interpretacją stałych i spójników logicznych

Składnia: Jak w TINY, z tą różnicą że:

- używamy Σ-termów zamiast wyrażeń całkowitych
- z każdą zmienną jest związany jej rodzaj z Σ
- przypisanie na zmienną jest dozwolone jedynie wówczas, gdy jej rodzaj jest taki, jak rodzaj termu
- używamy Σ-termów rodzaju *Bool* zamiast wyrażeń logicznych

Dziedziny semantyczne: Jak w TINY ale modyfikujemy stan:

$$\mathbf{State}_{\mathcal{A}} = \mathbf{Var} \to |\mathcal{A}|$$

(tak aby zmienne i ich wartości były podzielone na rodzaje z Σ)

Funkcje semantyczne: Jak w TINY, ale interpretując operacje w $|\mathcal{A}|$

$$\{\varphi\}\ S\ \{\psi\}$$

— — — jak poprzednio — — —

Przykładowa struktura i algebra

- do struktury Σ języka Tiny dodajemy:

$$
\begin{array}{ll}
\textbf{sorts} & \mathit{Array}; \\
\textbf{opns} & \mathit{newarr} : \mathit{Array}; \\
 & \mathit{put} : \mathit{Array} \times \mathit{Int} \times \mathit{Int} \to \mathit{Array}; \\
 & \mathit{get} : \mathit{Array} \times \mathit{Int} \to \mathit{Int};
\end{array}
$$

- i wzbogacamy algebrę $\mathcal{A}$ dla Tiny o nośnik i operacje:

$$
\begin{array}{ll}
\textbf{carriers} & \mathcal{A}_{\mathit{Array}} = \mathbf{Int} \to \mathbf{Int} \\
\textbf{operations} & \mathit{newarr}_{\mathcal{A}}(j) = 0 \\
 & \mathit{put}_{\mathcal{A}}(a, i, n) = a[i \mapsto n] \\
 & \mathit{get}_{\mathcal{A}}(a, i) = a(i)
\end{array}
$$

Przykład

$\{a{:}Array \land 0 \leqslant n\}$

$\quad m := 0;$

$\quad \textbf{while}\ \{0 \leqslant m \leqslant n \land \textit{is-sorted}(a, 0, m)\}\ m + 1 \leqslant n\ \textbf{do}$

$\qquad m := m + 1;\ k := m;$

$\qquad \textbf{while}\ \{0 \leqslant k \leqslant m \leqslant n \land \textit{is-nearly-sorted}(a, 0, k, m)\}\ 1 \leqslant k\ \textbf{do}$

$\qquad\quad k := k - 1;$

$\qquad\quad \textbf{if}\ get(a, k) \leqslant get(a, k + 1)\ \textbf{then}\ k := 0$

$\qquad\quad \textbf{else}\ x := get(a, k + 1);\ a := put(a, k + 1, get(a, k));$

$\qquad\qquad a := put(a, k, x)$

$\{\textit{is-sorted}(a, 0, n)\}$

gdzie:

$\textit{is-sorted}(a, i, j) \equiv a{:}Array \land \forall i', j'{:}Int.\, i \leqslant i' \leqslant j' \leqslant j \Rightarrow get(a, i') \leqslant get(a, j')$

$\textit{is-nearly-sorted}(a, i, k, j) \equiv \textit{is-sorted}(a, i, k - 1) \land \textit{is-sorted}(a, k, j) \land$
$\quad \forall i', j'{:}Int.(i \leqslant i' \leqslant k - 1 \land k + 1 \leqslant j' \leqslant j) \Rightarrow get(a, i') \leqslant get(a, j')$

Logika Hoare'a: reguły dowodzenia

— — — jak poprzednio — — —

$$\frac{}{\{\varphi[x \mapsto e]\}\ x := e\ \{\varphi\}}$$

$$\frac{}{\{\varphi\}\ \textbf{skip}\ \{\varphi\}}$$

$$\frac{\{\varphi\}\ S_1\ \{\theta\} \quad \{\theta\}\ S_2\ \{\psi\}}{\{\varphi\}\ S_1;\ S_2\ \{\psi\}}$$

$$\frac{\{\varphi \wedge b\}\ S_1\ \{\psi\} \quad \{\varphi \wedge \neg b\}\ S_2\ \{\psi\}}{\{\varphi\}\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \{\psi\}}$$

$$\frac{\{\varphi \wedge b\}\ S\ \{\varphi\}}{\{\varphi\}\ \textbf{while}\ b\ \textbf{do}\ S\ \{\varphi \wedge \neg b\}}$$

$$\frac{\varphi' \Rightarrow \varphi \quad \{\varphi\}\ S\ \{\psi\} \quad \psi \Rightarrow \psi'}{\{\varphi'\}\ S\ \{\psi'\}}$$

Twierdzenie:

System dowodowy Hoare'a jest poprawny, tzn.:

$$\textit{jeśli} \quad \boxed{\mathcal{TH}(\mathcal{A}) \vdash \{\varphi\}\, S\, \{\psi\}} \quad \textit{to} \quad \boxed{\models_{\mathcal{A}} \{\varphi\}\, S\, \{\psi\}}$$

Dowód:

— — — jak poprzednio — — —

Ku pełności

Musimy zagwarantować, że wszystkie potrzebne w dowodach asercje można wyrazić w logice asercji.

Dla $S \in \mathbf{Stmt}_\Sigma$ oraz $\psi \in \mathbf{Form}_\Sigma$, definiujemy:

$$wpre_{\mathcal{A}}(S, \psi) = \{s \in \mathbf{State}_{\mathcal{A}} \mid \text{ jeśli } \mathcal{S}_{\mathcal{A}}[\![S]\!]\, s = s' \in \mathbf{State}_{\mathcal{A}} \text{ to } \mathcal{F}_{\mathcal{A}}[\![\psi]\!]\, s' = \mathbf{tt}\}$$

Definicja:

Logika pierwszego rzędu jest ekspresywna nad $\mathcal{A}$ dla $\textsc{Tiny}_{\mathcal{A}}$ ($\mathcal{A}$ jest ekspresywny) jeśli dla każdych $S \in \mathbf{Stmt}_\Sigma$ oraz $\psi \in \mathbf{Form}_\Sigma$, istnieje najsłabszy liberalny warunek wstępny dla S i ψ, czyli formuła $\varphi_0 \in \mathbf{Form}_\Sigma$ taka, że

$$\{\varphi_0\}_{\mathcal{A}} = wpre_{\mathcal{A}}(S, \psi)$$

Relatywna pełność logiki Hoare'a

(pełność w sensie Cooka)

Twierdzenie:

Jeśli $\mathcal{A}$ jest ekspresywne, to system dowodzenia Hoare'a jest poprawny i relatywnie pełny, czyli:

$$\boxed{\mathcal{TH}(\mathcal{A}) \vdash \{\varphi\}\, S\, \{\psi\}} \quad \textit{iff} \quad \boxed{\models_{\mathcal{A}} \{\varphi\}\, S\, \{\psi\}}$$

Dowód: Przez indukcję strukturalną po S. Dzięki założeniu o ekspresywności i możliwości swobodnego korzystania z faktów z $\mathcal{TH}(\mathcal{A})$, wszystkie przypadki są łatwe!

Twierdzenie:

*$\mathcal{A}$ jest ekspresywne wtedy i tylko wtedy, gdy albo w $S \in$ **Stmt**$_\Sigma$ daje się zdefiniować standardowy model arytmetyki Peano, albo dla każdego $S \in$ **Stmt**$_\Sigma$, istnieje skończone ograniczenie na liczbę stanów osiągalnych w dowolnym obliczeniu S.*

Wychodząc poza TINY

Procedury: Dla danej **proc** p **is** (S_p):

$$\frac{\{\varphi\}\ \textbf{call}\ p\ \{\psi\} \vdash \{\varphi\}\ S_p\ \{\psi\}}{\{\varphi\}\ \textbf{call}\ p\ \{\psi\}}$$

Nie do końca dobrze:
relatywna pełność wymaga dodatkowych reguł
do manipulowania zmiennymi pomocniczymi

Zmienne: Dla nowej zmiennej y:

$$\frac{\{\varphi \wedge y = ??\}\ S[x \mapsto y]\ \{\psi\}}{\{\varphi\}\ \textbf{begin var}\ x\ S\ \textbf{end}\ \{\psi\}}$$

itd...

Ale istnieją ograniczenia...

Twierdzenie:

Nie istnieje poprawny i relatywnie pełny w sensie Cooka system dowodowy Hoare'a dla języka programowania dopuszczającego rekurencyjne procedury z parametrami proceduralnymi, procedurami lokalnymi i globalnymi zmiennymi z wiązaniem statycznym.

Klucz do dowodu:

Twierdzenie:

Dla programów w takim języku problem stopu jest nierozstrzygalny nawet dla skończonych typów danych $\mathcal{A}$ (z co najmniej dwoma elementami).

Co w przypadku języka $\textsc{Tiny}_{\mathcal{A}}$*?*

DOBRE WIEŚCI:
Dowodzenie poprawności za pomocą relacji dobrze ufundowanych nadal jest możliwe!

Dla przypomnienia, podstawowa reguła:

$$\frac{(nat(l) \land \varphi(l+1)) \Rightarrow b \qquad [nat(l) \land \varphi(l+1)]\; S\; [\varphi(l)] \qquad \varphi(0) \Rightarrow \neg b}{[\exists l.nat(l) \land \varphi(l)]\; \textbf{while}\; b\; \textbf{do}\; S\; [\varphi(0)]}$$

Problem?

Dla danej sygnatury Σ, niech Σ^+ będzie jej rozszerzeniem o język arytmetyki (Peano): predykaty $nat(_)$ i $_ \leqslant _$, stałe 0, 1, operacje $_ + _$, $_ - _$, $_ * _$.
Niech $\mathcal{A}$ będzie Σ^+-strukturą, w której interpretacja $nat(_)$ jest domknięta na stałe i operacje arytmetyczne.
Niestety:

reguła dla pętli nie musi być poprawna w języku $\textsc{Tiny}_{\mathcal{A}}$

Na przykład, zazwyczaj otrzymujemy:

$$\mathcal{TH}(\mathcal{A}) \vdash [nat(x)]\ \textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\ [\textbf{true}]$$

Ale: Semantycznie ta własność nie zachodzi, gdy na przykład $\mathcal{A}$ jest niestandardowym modelem arytmetyki.

Poważny problem?

Σ^+-struktura $\mathcal{A}$ jest *arytmetyczna* jeśli zbiór elementów $a \in |\mathcal{A}|$, dla których w $\mathcal{A}$ zachodzi $nat(a)$, z operacjami i relacjami arytmetycznymi określonymi jak w $\mathcal{A}$, tworzy *standardowy model arytmetyki*.

Twierdzenie:

Jeśli $\mathcal{A}$ jest arytmetyczny, to

Poprawność

$$\text{jeśli } \boxed{\mathcal{TH}(\mathcal{A}) \vdash [\varphi]\, S\, [\psi]} \text{ to } \boxed{\models_{\mathcal{A}} [\varphi]\, S\, [\psi]}$$

Jeśli ponadto skończone ciągi elementów z $|\mathcal{A}|$ można zakodować za pomocą formuły jako pojedyncze elementy zbioru $|\mathcal{A}|$, to

$$\boxed{\mathcal{TH}(\mathcal{A}) \vdash [\varphi]\, S\, [\psi]} \text{ wtw } \boxed{\models_{\mathcal{A}} [\varphi]\, S\, [\psi]}$$

Poprawność & pełność

Zadanie programistyczne

Dla danego warunku wstępnego φ i warunku końcowego ψ
zbuduj program $S^?$, taki że

$$[\varphi]\ S^?\ [\psi]$$

Aby zrealizować zadanie programistyczne

Dla danego warunku wstępnego φ i warunku końcowego ψ
zbuduj program $S^?$, taki że
$[\varphi]\ S^?\ [\psi]$

- najpierw wymyśl dobre rozwiązanie (lub wykorzystaj to, czego nauczono Cię na studiach), a następnie

  napisz program S

- po czym sprawdź, że $\models [\varphi]\ S\ [\psi]$
  - czyli udowodnij $\vdash [\varphi]\ S\ [\psi]$

Zadanie jest zakończone
dopiero po wykonaniu tych kroków.

Oczywiście jeśli:
udało się je wykonać...

Przykład

Zbuduj instrukcję $S^?$, taką że

$$[n \geqslant 0]\ S^?\ [rt^2 \leqslant n \wedge n < (rt+1)^2]$$

(i n nie podlega zmianom w $S^?$)

Przykładowe, sensownie wyglądające rozwiązanie:

$rt := 0;\ sqr := 0$
while $sqr \leqslant n$ **do**
    $rt := rt + 1;\ sqr := sqr + 2 * rt + 1$

Nie, to jeszcze nie koniec!

Trzeba pokazać:

$$\models^{?} \begin{array}{l} [n \geqslant 0] \\ \quad rt := 0;\ sqr := 0 \\ \quad \textbf{while}\ sqr \leqslant n\ \textbf{do} \\ \qquad\quad rt := rt + 1;\ sqr := sqr + 2 * rt + 1 \\ [rt^2 \leqslant n \wedge n < (rt+1)^2] \end{array}$$

Próba dowodu

Na przykład:

$$\vdash \begin{array}{l} [n \geqslant 0] \\ \quad rt := 0;\ sqr := 0 \\ \quad [rt = 0 \wedge sqr = 0] \\ \quad \textbf{while } [sqr = rt^2]\ sqr \leqslant n \textbf{ do decr } n + 1 - rt \textbf{ in Nat wrt } > \\ \qquad\quad rt := rt + 1;\ sqr := sqr + 2 * rt + 1 \\ [rt^2 \leqslant n \wedge n < (rt+1)^2] \end{array}$$

ŹLE!

Co jest źle?

Są dwie możliwości (obie równie przykre):

- program nie jest zgodny ze specyfikacją
- nie potrafimy udowodnić, że program jest zgodny ze specyfikacją

W tym przypadku:

$$\not\models \quad \begin{array}{l} [n \geqslant 0] \\ \quad rt := 0; sqr := 0 \\ \quad \textbf{while } sqr \leqslant n \textbf{ do} \\ \qquad\quad rt := rt + 1; sqr := sqr + 2 * rt + 1 \\ [rt^2 \leqslant n \wedge n < (rt + 1)^2] \end{array}$$

(co można pokazać, na przykład, testując program)

Co teraz?

Druga próba

$$
\begin{array}{l}
rt := 0;\ sqr := 1\\
\textbf{while}\ sqr \leqslant n\ \textbf{do}\\
\qquad rt := rt + 1;\ sqr := sqr + 2 * rt + 1
\end{array}
$$

Nie, to jeszcze nie koniec!

Musimy pokazać:

$$
\models^{?}
\begin{array}{l}
[n \geqslant 0]\\
\quad rt := 0;\ sqr := 1\\
\quad \textbf{while}\ sqr \leqslant n\ \textbf{do}\\
\qquad\quad rt := rt + 1;\ sqr := sqr + 2 * rt + 1\\
[rt^2 \leqslant n \wedge n < (rt + 1)^2]
\end{array}
$$

Próba dowodu

Na przykład:

$$\vdash \begin{array}{l} [n \geqslant 0] \\ \quad rt := 0;\ sqr := 1 \\ \quad [rt = 0 \wedge sqr = 1] \\ \quad \textbf{while}\ [sqr = (rt+1)^2]\ sqr \leqslant n\ \textbf{do} \\ \qquad \textbf{decr}\ n + 1 - rt\ \textbf{in Nat wrt} > \\ \qquad rt := rt + 1;\ sqr := sqr + 2 * rt + 1 \\ [rt^2 \leqslant n \wedge n < (rt+1)^2] \end{array}$$

I ZNOWU ŹLE!

Tym razem niewłaściwie przeprowadzaliśmy dowód...

Silniejszy niezmiennik pętli
$[sqr = (rt+1)^2 \wedge rt^2 \leqslant n]$
umożliwia zakończenie dowodu

Przykład

Buduj program stopniowo,
postępując w każdym kroku tak,
aby poprawność wyniku była zapewniona
przez poprawność pojedynczych kroków

Zbuduj instrukcję $S^?$, taką że

$$[n \geqslant 0]\ S^?\ [rt^2 \leqslant n \wedge n < (rt+1)^2]$$

(i n nie podlega modyfikacjom w $S^?$)

Krok 1

Skorzystajmy z następującego schematu:

$$[n \geqslant 0] \; S_1^? \; [rt = 0 \wedge sqr = 1] \; S_2^? \; [rt^2 \leqslant n \wedge n < (rt+1)^2]$$

Chcemy zatem:

- zbudować $S_1^?$ tak, aby $[n \geqslant 0] \; S_1^? \; [rt = 0 \wedge sqr = 1]$
- niezależnie od tego, zbudować $S_2^?$ tak, aby $[rt = 0 \wedge sqr = 1] \; S_2^? \; [rt^2 \leqslant n \wedge n < (rt+1)^2]$
- a następnie zdefiniować $S^? \equiv S_1^?; S_2^?$

Poprawność wynika z asercji pośredniej $[rt = 0 \wedge sqr = 1]$

Krok 2

Zbuduj $S_1^?$ tak, aby

$$[n \geqslant 0]\ S_1^?\ [rt = 0 \wedge sqr = 1]$$

(i n nie podlega modyfikacjom w $S_1^?$)

ŁATWE!

Po prostu jako $S_1^?$ wystarczy wziąć

$$rt := 0;\ sqr := 1$$

Weryfikacja jest natychmiastowa!

Krok 3

Zbuduj $S_2^?$, tak aby

$$[rt = 0 \wedge sqr = 1]\ S_2^?\ [rt^2 \leqslant n \wedge n < (rt+1)^2]$$

(i n nie podlegała modyfikacjom w $S_2^?$)

Decyzja projektowa: korzystamy z następującego schematu

$[rt = 0 \wedge sqr = 1]$

while $[\varphi^?]$ $b^?$ **do decr** $e^?$ **in** $W^?$ **wrt** $\succ^?$

$S_3^?$

$[rt^2 \leqslant n \wedge n < (rt+1)^2]$

Zatem:

Wybieramy $W^?$ i dobrze ufundowaną relację $\succ^? \subseteq W^? \times W^?$ oraz niezmiennik $\varphi^?$, wyrażenie logiczne $b^?$, wyrażenie $e^?$, a następnie budujemy $S_3^?$ tak, aby:

- $(rt = 0 \wedge sqr = 1) \implies \varphi^?$
- $(\varphi^? \wedge \neg b^?) \implies (rt^2 \leqslant n \wedge n < (rt+1)^2)$
- $[\varphi^? \wedge b^?]\ S_3^?\ [\varphi^?]$
- $\mathcal{E}[\![e^?]\!]\ s \succ^? \mathcal{E}[\![e^?]\!]\ (\mathcal{S}[\![S_3^?]\!]\ s)$ dla wszystkich stanów $s \in \{\varphi^? \wedge b^?\}$

Eureka!

Wybieramy:

$$\varphi^? \equiv (sqr = (rt+1)^2 \wedge rt^2 \leqslant n)$$

Wówczas:

- Pierwsze wymaganie wynika natychmiast.
- Bierzemy $b^? \equiv (sqr \leqslant n)$ — i łatwo wynika drugie wymaganie.
- Wybieramy:
  - $W^? = \mathbf{Nat}$ z relacją (dobrze ufundowaną) $\succ^? = >$
  - $e^? = n - rt$

Przechodzimy następnie do kolejnych kroków zadania...

Krok 4

Zbuduj $S_3^?$ tak, aby

$$[sqr = (rt+1)^2 \land rt^2 \leqslant n \land sqr \leqslant n]$$
$$S_3^?$$
$$[sqr = (rt+1)^2 \land rt^2 \leqslant n]$$

(i n nie podlegała zmianom w $S_3^?$)

Decyzja projektowa: korzystamy z następującego schematu

$$[sqr = (rt+1)^2 \leqslant n \land sqr \leqslant n]$$
$$S_4^?$$
$$[sqr = rt^2 \leqslant n]$$
$$S_5^?$$
$$[sqr = (rt+1)^2 \land rt^2 \leqslant n]$$

Własność stopu

Nie wolno zapomnieć:
własność stopu to także część wymagań

Dla $S_3^?$ wymagamy też, aby:

- $\mathcal{E}[\![n - rt]\!]\ s > \mathcal{E}[\![n - rt]\!]\ (\mathcal{S}[\![S_3^?]\!]\ s)$ dla $s \in \{sqr = (rt+1)^2 \leqslant n \wedge rt^2 \leqslant n\}$

Aby to zapewnić:

- $\mathcal{E}[\![n - rt]\!]\ s \geqslant \mathcal{E}[\![n - rt]\!]\ (\mathcal{S}[\![S_4^?]\!]\ s)$ dla $s \in \{sqr = (rt+1)^2 \leqslant n \wedge rt^2 \leqslant n\}$
- $\mathcal{E}[\![n - rt]\!]\ s \geqslant \mathcal{E}[\![n - rt]\!]\ (\mathcal{S}[\![S_5^?]\!]\ s)$ dla $s \in \{sqr = rt^2 \leqslant n\}$

przy czym co najmniej jedna z nierówności musi być ostra

Kroki 5 & 6

$S_4^?$ definiujemy jako

$$rt := rt + 1$$

$S_5^?$ zaś definiujemy jako

$$sqr := sqr + 2 * rt + 1$$

Weryfikacja jest natychmiastowa
(łącznie z warunkami stopu)

ŁATWE!

Łącząc wszystkie kroki

$$
\begin{array}{l}
[n \geqslant 0] \\
\quad rt := 0;\ sqr := 1 \\
\quad [rt = 0 \wedge sqr = 1] \\
\quad \textbf{while } [sqr = (rt+1)^2 \wedge rt^2 \leqslant n]\ sqr \leqslant n \textbf{ do} \\
\qquad\qquad \textbf{decr } n - rt \textbf{ in Nat wrt } > \\
\qquad\quad (rt := rt + 1\ [sqr = rt^2 \leqslant n]\ sqr := sqr + 2 * rt + 1) \\
[rt^2 \leqslant n \wedge n < (rt+1)^2]
\end{array}
$$

Poprawność przez konstrukcję!!!

. . . dowody są od razu gotowe!

Po co są specyfikacje?

Z punktu widzenia użytkownika systemu: specyfikacja opisuje własności systemu, na których użytkownik może polegać.

Z punktu widzenia twórcy systemu: specyfikacja opisuje wszystkie wymagania, jakie system musi spełniać.

Inżynieria wymagań

Budowanie specyfikacji: ustalenie pożądanych własności systemu i zbudowanie opisującej je specyfikacji.

Walidacja specyfikacji: sprawdzenie, czy specyfikacja rzeczywiście opisuje oczekiwane właściwości systemu.

Podejście bazujące na modelu: zadajemy model — system jest *poprawny* jeśli zachowuje się tak samo, jak przedstawiony model.

Podejście bazujące na własnościach: zadajemy listę wymagań — system jest *poprawny* jeśli spełnia wszystkie przedstawione wymagania.

Walidacja specyfikacji

- Dowodzenie twierdzeń
- Prototypowanie i testowanie

Języki budowania specyfikacji

Jest ich sporo... Trzeba jeden wybrać.

Oczywiście, zalecany wybór to **CASL** :-)

Nawet rzeczywiste, duże specyfikacje mogą być zrozumiałe!

Kluczowy pomysł: STRUKTURA

Dzięki niej można:

- tworzyć złożone specyfikacje, rozumieć je i wnioskować o ich własnościach
- (choć implementacja nie musi być z tą strukturą zgodna)

Zadanie programisty

Dla danej specyfikacji wymagań
zbuduj moduł, który ją poprawnie implementuje

Dla danej specyfikacji wymagań SP
zbuduj program P tak, aby
$SP \rightsquigarrow P$

Formalna definicja $SP \rightsquigarrow P$ wynika z *semantyki*
(języka budowania specyfikacji i języka programowania)

Podstawowy pomysł

$$SP \rightsquigarrow M$$

Nie w jednym kroku!

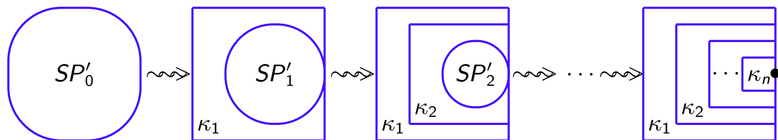
Raczej: budujemy program z jego specyfikacji krok po kroku, stopniowo dodając coraz więcej szczegółów i podejmując coraz więcej decyzji projektowych i implementacyjnych, aż do uzyskania specyfikacji, którą łatwo zaimplementować od razu

$$SP_0 \rightsquigarrow\!\!\!> SP_1 \rightsquigarrow\!\!\!> \cdots \rightsquigarrow\!\!\!> SP_n$$

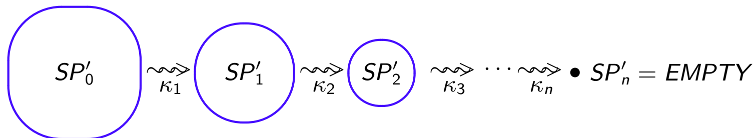
(*relacja uszczegóławiania*) jest definiowana formalnie, zapewniając:

$$\frac{SP_0 \rightsquigarrow\!\!\!> SP_1 \rightsquigarrow\!\!\!> \cdots \rightsquigarrow\!\!\!> SP_n \qquad SP_n \rightsquigarrow P}{SP_0 \rightsquigarrow P}$$

W praktyce, niektóre elementy programu zostają w danym kroku ustalone:



Oddzielamy je od części,
która pozostała tak naprawdę do zaimplementowania:



Dekompozycja zadania

Kroki dekompozycji zadania:

$$SP \rightsquigarrow\!\!\!> BR \begin{cases} SP_1 \\ \vdots \\ SP_n \end{cases}$$

Krok BR dekompozycji zadania programistycznego określać musi specyfikacje składowych oraz "*procedurę łączącą*" κ_{BR} (n-argumentowy *konstruktor*, odwzorujący programy w programy).
Poprawność dekompozycji wyraża następujące wymaganie:

$$\frac{SP_1 \rightsquigarrow P_1 \quad \cdots \quad SP_n \rightsquigarrow P_n}{SP \rightsquigarrow \kappa_{BR}(P_1, \ldots, P_n)}$$

Wielkie Wyzwanie

- Jest całkiem niezła teoria opisująca ten proces;
- I wiele złej praktyki . . .

Opracować praktyczne metody
specyfikacji i systematycznego konstruowania programów
z solidnymi podstawami formalnymi