7. Wykład 7: Rachunek λ z typami prostymi.

Przedmiotem niniejszego wykładu będzie **rachunek λ z typami prostymi,** który jest rozszerzeniem rachunku λ z **typami**. Jako że omówiony dotychczas rachunek λ nie używał typów, będziemy go odtąd nazywali **beztypowym** rachunkiem λ.

W odróżnieniu od beztypowego rachunku λ, w którym podstawowe typy (takie jak zmienne boole'owskie czy liczby całkowite) są symulowane za pomocą λ-abstrakcji, rachunek λ z typami prostymi zakłada ustalony zbiór typów z prostymi konstrukcjami. Na przykład, możemy umówić się, że do rachunku włączymy typ bool wraz ze stałymi boole'owskimi true oraz false i dodatkową konstrukcją if e then e_1 else e_2. Tym samym możemy myśleć o rachunku λ z typami prostymi nie tylko jak o "gołym" rachunku służącym do badania siły wyrazu, ale jak o podzbiorze pewnego języka funkcyjnego. Przy tym założeniu dowolne wyrażenie w rachunku λ z typami prostymi może być dosłownie przetłumaczone na język funkcyjny taki jak SML.

Tak jak to miało miejsce w przypadku beztypowego rachunku λ, zaczniemy od sformułowania abstrakcyjnego syntaksu i semantyki operacyjnej dla rachunku λ z typami prostymi. Różnica w semantyce operacyjnej jest czysto nominalna, albowiem typy nie odgrywają roli w redukowaniu wyrażeń. Główna różnica powstaje przez wprowadzenie **systemu typów,** a zatem zbioru osądów i reguł wnioskowania służących przypisywaniu typów wyrażeniom. Typ przypisany danemu wyrażeniu określa rodzaj wartości do której wyrażenie jest wartościowane. Na przykład wyrażenie typu bool może być wartościowane wyłącznie do true lub false, ale do niczego innego.

Ucząc się rachunku λ z typami prostymi główny nacisk położymy na **bezpieczeństwo typów**, czyli główną własność systemu typów polegającą na tym, że wyrażenie z prawidłowym typem, zwane też wyrażeniem **dobrze typowanym,** nie może spowodować błędu podczas kompilacji. Jako że wyrażeniom przypisywane są typy podczas kompilowania, zaś bezpieczeństwo typów gwarantuje brak wystąpienia błędów, nie musimy stosować metody prób i błędów celem zlokalizowania w programie krytycznych błędów takich jak dodawanie adresów pamięci, odejmowanie liczb od ciągów itp. Ponieważ to właśnie tego typu proste błędy są najczęstszym powodem zastojów i opóźnień podczas programowania, widzimy że bezpieczeństwo typów ma sporą przewagę nad językami programowania bez systemu typów lub też z systemami typów bez bezpieczeństwa. To właśnie bezpieczeństwo typów jest odpowiedzialne za to, że programy napisane w językach funkcyjnych, które uda się skompilować, na ogół poprawnie działają.

7.1. **Abstrakcyjny syntaks.** Abstrakcyjny syntaks rachunku λ z typami prostymi jest dany jak następuje:

typ	A	::=	$P \mid A \to A$
typ podstawowy	P	::=	bool
wyraenie	e	::=	$x \mid \lambda x : A.e \mid e\ e$
			true|false|if e then e else e
warto	v	::=	$\lambda x : A.e$|true|false

Typ jest albo **typem podstawowym** P albo **typem funkcyjnym** $A \to A'$. Typ podstawowy to typ, dla którego konstrukcje podstawowa są dane jako część definicji. Tutaj używamy typu boole'owskiego bool jako typu podstawowego z którym wiążemy trzy konstrukcje podstawowe: stałe boole'owskie true i false oraz konstrukcja warunkowa if e then e_1 else e_2. Typ funkcyjny $A \to A'$ opisuje funkcje, które przyjmują argument typu A i zwracają wartość typu A'. Będziemy używać metazmiennych A, B, C dla oznaczania typów.

Rachunek λ z typami prostymi nie blokuje możliwości wykorzystania podstawowych typów. Innymi słowy pełni rolę szkieletu dla języków funkcyjnych, których system typów można rozszerzać dodatkowymi

typami podstawowymi. Na przykład powyższa definicja wykorzystuje bool jako jedyny typ podstawowy, ale powinno także być jasne jak można ją rozszerzyć o inne typy podstawowy (na przykład typ int ze stałymi będącymi liczbami całkowitymi oraz z operacjami arytmetycznymi). Z drugiej strony rachunek λ z typami prostymi musi mieć przynajmniej jeden typ podstawowy. W przeciwnym razie zbiór typów bazowych P byłby pusty, co z kolei powodowałoby, że zbiór typów A byłby pusty – wówczas nigdy nie bylibyśmy w stanie skonstruować wyrażenia z poprawnym typem.

Tak jak w przypadku beztypowego rachunku λ, wyrażenia zawierają zmienne, λ-abstrakcje i aplikacje. λ-abstrakcja $\lambda x : A.e$ bezpośrednio specyfikuje teraz typ A swego formalnego argumentu x. Jeżeli $\lambda x : A.e$ zostaje zastosowane do wyrażenia o nowym typie A' (to znaczy $A' \neq A$), aplikacja nie weryfikuje typu i tym samym nie będzie miała przypisanego typu, jak już wkrótce zobaczymy. Powiemy, że zmienna x jest związana z typem A w λ-abstrakcji $\lambda x : A.e$, albo że λ-abstrakcja $\lambda x : A.e$ wiąże zmienną x do typu A.

7.2. **Semantyka operacyjna.** Semantykę operacyjną dla rachunku λ z typami prostymi możemy rozwinąć podobnie jak rozwijaliśmy semantykę dla beztypowego rachunku λ: definiujemy odwzorowanie $FV(e)$ do obliczania zbioru zmiennych wolnych w e, unikające przechwyceń podstawianie $[e'/x]e$ oraz osąd redukcyjny $e \mapsto e'$ za pomocą odpowiednich reguł redukcyjnych. Jako że rachunek λ z typami prostymi nie różni się od beztypowego rachunku λ inaczej niż systemem typów, jego semantyka operacyjna jest w zasadzie taka sama jak dla rachunku beztypowego, o ile tylko zignorujemy typy w wyrażeniach.

Odwzorowanie $FV(e)$ jest zdefiniowane następująco:

$$
\begin{array}{rcl}
FV(x) & = & \{x\} \\
FV(\lambda x : A.e) & = & FV(e) \setminus \{x\} \\
FV(e_1 e_2) & = & FV(e_1) \cup FV(e_2) \\
FV(\mathsf{true}) & = & \emptyset \\
FV(\mathsf{false}) & = & \emptyset \\
FV(\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) & = & FV(e) \cup FV(e_1) \cup FV(e_2)
\end{array}
$$

Tak jak w przypadku beztypowego rachunku λ powiemy, że dane wyrażenie jest domknięte o ile nie zawiera zmiennych wolnych.

Podstawianie unikające przechwytywania zmiennych $[e'/x]e$ jest zdefiniowane następująco:

$$
\begin{array}{rcll}
[e'/x]x & = & e' & \\
[e'/x]y & = & y & \text{jeśli } x \neq y \\
[e'/x]\lambda x : A.e & = & \lambda x : A.e & \\
[e'/x]\lambda y : A.e & = & \lambda y : A.[e'/x]e & \text{jeśli } x \neq y, y \notin FV(e') \\
[e'/x](e_1 e_2) & = & [e'/x]e_1 [e'/x]e_2 & \\
[e'/x]\mathsf{true} & = & \mathsf{true} & \\
[e'/x]\mathsf{false} & = & \mathsf{false} & \\
[e'/x]\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 & = & \mathsf{if}\ [e'/x]e\ \mathsf{then}\ [e'/x]e_1\ \mathsf{else}\ [e'/x]e_2 &
\end{array}
$$

Jeżeli w $[e'/x]\lambda y : A.e$ nastąpi przechwycenie zmiennej, zmieniamy nazwę zmiennej związanej y za pomocą relacji α-równoważności $\equiv_\alpha$. Pomijamy tu definicję $\equiv_\alpha$, jako że nie różni się niczym od definicji już podanej.

Tak jak w przypadku beztypowego rachunku λ, różne strategie redukcyjne wymagają zastosowania różnych reguł redukcyjnych dla osądu redukcyjnego $e \mapsto e'$. Zdecydowaliśmy się wybrać tu strategię wywołania według wartości, która wygodnie i przejrzyście rozszerza się na rachunek λ z typami prostymi

poprzez **efekty obliczeniowe** takie jak mutowalne referncje, wyjątki i kontynuacje (o których powiemy za jakiś czas). Tym samym będziemy używać następujących reguł redukcyjnych:

$$\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2}\,Lam \qquad \frac{e_2 \mapsto e_2'}{(\lambda x{:}A.e) e_2 \mapsto (\lambda x{:}A.e) e_2'}\,Arg \qquad \frac{}{(\lambda x{:}A.e) v \mapsto [v/x]e}\,App$$

$$\frac{e \mapsto e'}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto \mathsf{if}\ e'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2}\,If$$

$$\frac{}{\mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_1}\,If_{\mathsf{true}} \qquad \frac{}{\mathsf{if\ false\ then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_2}\,If_{\mathsf{false}}$$

Reguły *Lam*, *Arg* oraz *App* są dokładnie takie same jak w przypadku beztypowego rachunku λ z tą różnicą, że stosujemy je do λ-abstrakcji postaci $\lambda x : A.e$. Reguły *If*, If_{true} oraz If_{false} specyfikują w jaki sposób można redukować konstrukcję warunkową $\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:

- redukujemy e do true albo false,
- jeżeli e redukuje się do true, wybieramy w drzewku decyzyjnym gałąź then i dalej redukujemy e_1,
- jeżeli e redukuje się do false, wybieramy w drzewku decyzyjnym gałąź else i dalej redukujemy e_2.

Tak jak wcześniej, będziemy pisać $\mapsto^*$ dla oznaczenia domknięcia przechodnio-zwrotnego relacji $\mapsto$. Powiemy, że e wartościuje się do v jeżeli zachodzi $e \mapsto^* v$.

7.3. **System typów.** Celem tego fragmentu wykładu jest zbudowanie systemu reguł wnioskowania służących przypisywaniu typów do wyrażeń w rachunku λ z typami prostymi. Będziemy używać osądu zwanego **osądem typującym**, natomiast o regułach wnioskowania służących do wydedukowania osądu typującego będziemy mówili jak o **regułach typowania.** System, jaki w ten sposób stworzymy, będziemy nazywali **systemem typów** dla rachunku λ z typami prostymi.

Aby zrozumieć, jakiej postaci powinien być osąd typujący, rozważmy prosty przykład funkcji identycznościowej $\mathsf{id} = \lambda x : A.x$. Intuicyjnie funkcja id powinna mieć typ funkcyjny $A \to A$ ponieważ bierze na wejściu argument o typie A i zwraca to samo. W jaki sposób możemy zatem określić typ id? Jako że id jest λ-abstrakcją o argumencie typu A, musimy określić typ jej zawartości. Widzimy wszakże, że nie można zrobić tego w oderwaniu od typu argumentu: bez założeń o typie argumentu x nie sposób określić typu zawartości.

Powyższy przykład pokazuje, że nie da się uniknąć użycia założeń o typach zmiennych w osądach typujących. Tym samym będziemy zmuszeni wprowadzić pojęcie **kontekstu typującego** do oznaczenia zbioru założeń o typach zmiennch; będziemy używali **spójnika typującego** $x : A$ do oznaczenia tego, że zakładamy iż zmienna x jest typu A:

$$\text{kontekst typujący} \qquad \Gamma \ ::= \ \cdot \mid \Gamma, x : A$$

- $\cdot$ oznacza pusty kontekst typujący i będzie naszą notacją na oznaczenie zbioru pustego $\emptyset$;
- $\Gamma, x : A$ rozszerza Γ o spójnik typujący $x : A$ i będzie naszą notacją na oznaczenie $\Gamma \cup \{x : A\}$; będziemy stosowali skrót $x : A$ do oznaczenia$\cdot, x : A$ dla jednoelementowego kontekstu typującego $\{x : A\}$;
- będziemy używać notacji dla kontekstów typujących dość elastycznie pisząc, na przykład, $\Gamma, x : A, \Gamma'$ dla oznaczenia $\Gamma \cup \{x : A\} \cup \Gamma'$.

Dla prostoty będziemy zawsze zakładać, że zmienne w danym kontekście typującym są parami rozłączne. Tym samym kontekst $\Gamma, x : A$ nie jest zdefiniowany, jeśli Γ zawiera inny spójnik typujący postaci $x : A'$.

System typów będzie wykorzystywał następującą postać osądów typujących:

$$\Gamma \vdash e : A \Leftrightarrow \text{ wyrażenie } e \text{ jest typu } A \text{ przy kontekście typującym } \Gamma$$

$\Gamma \vdash e : A$ oznacza, że jeśli zastosujemy każdy ze spójników typujących $x : A$ w Γ jako założenie. to będziemy w stanie pokazać, że wyrażenie e jest typu A. Prostym sposobem zrozumienia roli Γ jest traktowanie jej jako zbioru spójników typujących dla zmiennych wolnych w e, jakkolwiek Γ może zawierać też spójniki typujące dla zmiennych, które nie występują w e. Na przykład, wyrażenie domknięte e typu A wymaga osądu typującego $\cdot \vdash e : A$ z pustym kontekstem typującym (ponieważ nie zawiera zmiennych wolnych), podczas gdy wyrażenie e' ze zmienną wolną x wymaga osądu typującego $\Gamma \vdash e' : A'$, gdzie Γ zawiera przynajmniej jeden spójnik typujący $x : B$, dla pewnego typu B.

Po tych wstępnych wyjaśnieniach, możemy teraz przedstawić reguły typujące dla rachunku λ z typami prostymi:

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash x{:}A}\,\mathsf{Var} \qquad \frac{\Gamma, x{:}A \vdash e{:}B}{\Gamma \vdash \lambda x{:}A.e{:}A \to B}\,\to \mathsf{I} \qquad \frac{\Gamma \vdash e{:}A \to B \;\; \Gamma \vdash e'{:}A}{\Gamma \vdash e e'{:}B}\,\to E$$

$$\frac{}{\Gamma \vdash \mathsf{true}{:}\mathsf{bool}}\,\mathsf{True} \qquad \frac{}{\Gamma \vdash \mathsf{false}{:}\mathsf{bool}}\,\mathsf{False} \qquad \frac{\Gamma \vdash e{:}\mathsf{bool} \;\; \Gamma \vdash e_1{:}A \;\; \Gamma \vdash e_2{:}A}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2{:}A}\,\mathsf{If}$$

- Reguła Var powiada, że spójnik typujący w kontekście typującym jest założeniem. Inaczej możemy zapisać tę samą regułę następująco:

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A}\,\mathsf{Var}$$

- Reguła $\to \mathsf{I}$ powiada, że jeśli e jest typu B pod założeniem, że x jest typu A, to wówczas $\lambda x : A.e$ jest typu $A \to B$. Jeśli odczytamy regułę $\to \mathsf{I}$ od przesłanki począwszy, a na wniosku skończywszy (czyli "od góry do dołu"), to "wprowadzimy" typ funkcyjny $A \to B$ z osądu w przesłance, co jest powodem dla którego regułę nazywamy "$\to$ Indukcją". Zauważmy, że jeśli Γ zawiera już spójnik typujący dla zmiennej x (czyli $x : A' \in \Gamma$), to zmieniamy nazwę x na nową za pomocą α-konwersji. Tym samym możemy bez straty ogólności założyć, że w regule $\to \mathsf{I}$ nigdy nie dochodzi do konfliktów oznaczeń.
- Reguła $\to \mathsf{E}$ powiada, że jeśli e jest typu $A \to B$ oraz e' jest typu A, to wówczas ee' jest typu B. Jeśli odczytamy tę regułę "od góry do dołu", to "wyeliminujemy" typ funkcyjny $A \to B$ celem stworzenia wyrażenia o mniejszym typie B, co jest powodem dla którego regułę tę nazywamy "$\to$ Eliminacją".
- Reguły True oraz False przypisują podstawowy typ bool stałym boole'owskim true i false. Zauważmy, że kontekst typujący Γ nie jest tu używany, ponieważ w true i false nie występują zmienne wolne.
- Reguła If powiada, że jeśli e jest typu bool oraz zarówno e_1 jak i e_2 są tego samego typu A, to wówczas $\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ jest typu A.

Drzewko dowodowe dla osądu typującego będziemy nazywać **dowodem typującym**. Oto kilka przykładów poprawnych dowodów typujących. Pierwszy przykład pozwala wywnioskować typ funkcji identycznościowej (gdzie nie używamy przesłanki w regule Var):

$$\frac{\dfrac{}{\Gamma, x : A \vdash x : A}\,\mathsf{Var}}{\Gamma \vdash \lambda x : A.x : A \to A}\,\to \mathsf{I}$$

Jako że $\lambda x : A.x$ jest domknięta, możemy użyć pustego kontekstu typującego $\cdot$ dla Γ:

$$\frac{\dfrac{}{x : A \vdash x : A}\,\mathsf{Var}}{\cdot \vdash \lambda x : A.x : A \to A}\,\to \mathsf{I}$$

W drugim z przykładów skrócimy zapis $\Gamma, x : \mathsf{bool}, y_1 : A, y_2 : A$ do Γ'. Zauważmy także, że $\mathsf{bool} \to A \to A \to A$ jest równoważne z $\mathsf{bool} \to (A \to (A \to A))$, ponieważ $\to$ jest prawostronnie łączne:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x : \mathsf{bool} \in \Gamma'}{\Gamma' \vdash x : \mathsf{bool}}\,\mathsf{Var} \quad \dfrac{y_1 : A \in \Gamma'}{\Gamma' \vdash y_1 : A}\,\mathsf{Var} \quad \dfrac{y_2 : A \in \Gamma'}{\Gamma' \vdash y_2 : A}\,\mathsf{Var}}{\Gamma, x : \mathsf{bool}, y_1 : A, y_2 : A \vdash \mathsf{if}\ x\ \mathsf{then}\ y_1\ \mathsf{else}\ y_2 : A}\,\mathsf{If}}{\Gamma, x : \mathsf{bool}, y_1 : A \vdash \lambda y_2 : A.\mathsf{if}\ x\ \mathsf{then}\ y_1\ \mathsf{else}\ y_2 : A \to A}\to \mathsf{I}}{\Gamma, x : \mathsf{bool} \vdash \lambda y_1 : A.\lambda y_2 : A.\mathsf{if}\ x\ \mathsf{then}\ y_1\ \mathsf{else}\ y_2 : A \to A \to A}\to \mathsf{I}}{\Gamma \vdash \lambda x : \mathsf{bool}.\lambda y_1 : A.\lambda y_2 : A.\mathsf{if}\ x\ \mathsf{then}\ y_1\ \mathsf{else}\ y_2 : A \to A \to A \to A}\to \mathsf{I}$$

Trzeci przykład pozwala wywnioskować typ funkcji składającej dwie funkcje f i g, przy czym skrócimy zapis $\Gamma, f : A \to B, g : B \to C, x : A$ do Γ':

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{g : B \to C \in \Gamma'}{\Gamma' \vdash g : B \to C}\,\mathsf{Var} \quad \dfrac{\dfrac{f : A \to B \in \Gamma'}{\Gamma' \vdash f : A \to B}\,\mathsf{Var} \quad \dfrac{x : A \in \Gamma'}{\Gamma' \vdash x : A}\,\mathsf{Var}}{\Gamma' \vdash f\ x : B}\to \mathsf{E}}{\Gamma, f : A \to B, g : B \to C, x : A \vdash g(f\ x) : C}\to \mathsf{E}}{\Gamma, f : A \to B, g : B \to C \vdash \lambda x : A.g(f\ x) : A \to C}\to \mathsf{I}}{\Gamma, f : A \to B \vdash \lambda g : B \to C.\lambda x : A.g(f\ x) : (B \to C) \to (A \to C)}\to \mathsf{I}}{\Gamma \vdash \lambda f : A \to B.\lambda g : B \to C.\lambda x : A.g(f\ x) : (A \to B) \to (B \to C) \to (A \to C)}\to \mathsf{I}$$

Zakończymy ten fragment naszych rozważań dowodami dwóch własności osądów typujących: **permutacji** oraz **osłabiania**. Własność permutacji powiada, że kontekst typujący Γ jest zbiorem, nie zaś ciągiem, a więc kolejność nie ma w nim znaczenia i dwa konteksty typujące są takie same z dokładnością do permutacji. Na przykład $\Gamma, x : A, y : B$ można identyfikować z $\Gamma, y : B, x : A$. Z kolei własność osłabiania orzeka, iż jeśli potrafimy udowodnić, że wyrażenie e jest typu A przy założeniu kontekstu Γ, to możemy to samo udowodnić przy założeniu kontekstu Γ' upraszczającego Γ o spójnik typujący $x : A$ (albowiem możemy po prostu zignorować nowy spójnik typujący). Własności te będziemy też nazywać **własnościami strukturalnymi** osądów typujących, jako że opisują one przede wszystkim strukturę osądów typujących, nie zaś ich dowody typujące.

Stwierdzenie 1 (własność permutacji)**.** *Jeśli* $\Gamma \vdash e : A$ *oraz* Γ' *jest permutacją* Γ*, to wówczas* $\Gamma' \vdash e : A$*.*

Dowód. Wystarczy zastosować indukcję reguł względem osądu $\Gamma \vdash e : A$. □

Stwierdzenie 2 (własność osłabiania)**.** *Jeśli* $\Gamma \vdash e : C$*, to wówczas* $\Gamma, x : A \vdash e : A$*.*

Dowód. Wystarczy zastosować indukcję reguł względem osądu $\Gamma \vdash e : C$. Zajmiemy się tu trzema przypadkami, pozostałe dowodzi się w podobny sposób.

Przypadek $\frac{y:C \in \Gamma}{\Gamma \vdash y:C}\mathsf{Var}$ gdzie $e = y$:

$y : C \in \Gamma, x : A$ wobec $y : C \in \Gamma$

$\Gamma, x : A \vdash y : C$ wobec reguły Var.

Przypadek $\frac{\Gamma, y:C_1 \vdash e':C_2}{\Gamma \vdash \lambda y:C_1.e':C_1 \to C_2} \to \mathsf{I}$ gdzie $e = \lambda y : C_1.e'$ oraz $C = C_1 \to C_2$:

W tym przypadku $\Gamma \vdash e : C$ dowodzimy z zastosowaniem reguły $\to \mathsf{I}$. Innymi słowy, ostatnia reguła wnioskowania użyta w dowodzie $\Gamma \vdash e : C$ to reguła $\to \mathsf{I}$. Wówczas e musi być postaci $\lambda y : C_1.e'$ dla pewnego typu $C = C_1 \to C_2$, w przeciwnym bowiem razie reguła $\to \mathsf{I}$ nie może być zastosowana. W tym przypadku przesłanka jest wyznaczona jednoznacznie jako $\Gamma, y : C_1 \vdash e' : C_2$.

$\Gamma, y : C_1, x : A \vdash e' : C_2$ wobec założenia indukcyjnego

$\Gamma, x : A, y : C_1 \vdash e' : C_2$ wobec własności permutacji
$\Gamma, x : A \vdash \lambda y : C_1.e' : C_1 \to C_2$ wobec reguły $\to \mathsf{I}$.

Przypadek $\frac{\Gamma \vdash e_1 : B \to C \;\; \Gamma \vdash e_2 : B}{\Gamma \vdash e_1 \;\; e_2 : C} \to \mathsf{E}$ gdzie $e = e_1 \; e_2$:

W tym przypadku $\Gamma \vdash e : C$ dowodzimy z zastosowaniem reguły $\to \mathsf{E}$. Wówczas e musi być postaci $e_1 \; e_2$ i dwie przesłanki są jednoznacznie wyznaczone dla pewnego typu B, w przeciwnym bowiem razie reguła $\to \mathsf{E}$ nie może być zastosowana.

$\Gamma, x : A \vdash e_1 : B \to C$ wobec założenia indukcyjnego względem $\Gamma \vdash e_1 : B \to C$
$\Gamma, x : A \vdash e_2 : B$ wobec założenia indukcyjnego względem $\Gamma \vdash e_2 : B$
$\Gamma \vdash e_1 \; e_2 : C$ wobec reguły $\to \mathsf{E}$. □

7.4. **Bezpieczeństwo typów.** Celem określenia własności wyrażeń zbudowaliśmy dwa systemy dla rachunku λ z typami prostymi: semantykę operacyjną i system typów. Semantyka operacyjna pozwala nam na rozpoznawanie własności dynamicznych, a konkretnie wartości, związanych z wyrażeniami. Wartości można rozpatrywać jako własności dynamiczne w takim sensie, że w ogólności mogą być określone tylko w czasie wykonywania programu. Z tego powodu semantyka operacyjna czasami nazywana jest też **semantyką dynamiczną**. Z drugiej strony, system typów pozwala nam na badanie własności statycznych wyrażeń, a więc ich typów. Typy są własnościami statycznymi w takim sensie, że są określane w czasie kompilacji i pozostają "statyczne" podczas wykonywania programu. Z tego względu system typów bywa też nazywany **semantyką statyczną.**

System typów skonstruowaliśmy niezależnie od semantyki operacyjnej. Wobec tego istnieje realne zagrożenie, że stworzona przez nas konstrukcja nie przestrzega zasad ustalonych przez semantykę operacyjną. Na przykład możemy wyobrazić sobie sytuację, w której dwa różne typy zostaną przypisane dwóm wyrażeniom e oraz e' takim, że $e \mapsto e'$, co nie jest naturalne, albowiem nie spodziewamy się, aby w procesie redukcji ulegały zmianie typy. Podobnie, mogłoby się zdarzyć, że poprawny typ zostanie przypisany nonsensownemu wyrażeniu, co również wydaje się nienaturalne, albowiem spodziewamy się, że każde wyrażenie o popranym typie będzie poprawnym programem. Bezpieczeństwo typów, najbardziej podstawowa własność systemu typów, łączy system typów z semantyką operacyjną w taki sposób, żeby obydwa systemy funkcjonowały harmonijnie. Nieformalnie możemy powiedzieć, że bezpieczeństwo typów gwarantuje nam, że dobrze typowane wyrażenia nie mogą nie działać.

Na bezpieczeństwo typów składają się dwa twierdzenia: **twierdzenie o postępie** oraz **twierdzenie o zachowaniu**. Twierdzenie o postępie mówi, że dobrze typowane wyrażenie domknięte "nie zacina się": albo jest wartością, albo redukuje się do innego wyrażenia:

Twierdzenie 7 (o postępie)**.** *Jeśli* $\cdot \vdash e : A$ *dla pewnego typu* A*, to albo* e *jest wartością, albo istnieje* e' *takie, że* $e \mapsto e'$*.*

Twierdzenie o zachowaniu mówi z kolei, że gdy dobrze typowane wyrażenie redukuje się, to wyrażenie po redukcji jest również dobrze typowane i zachowuje typ:

Twierdzenie 8. *Jeśli* $\Gamma \vdash e : A$ *oraz* $e \mapsto e'$*, to wówczas* $\Gamma \vdash e' : A$*.*

Zauważmy, że twierdzenie o postępie zakłada pusty kontekst typujący (dlatego dotyczy **domkniętych** dobrze typowanych wyrażeń e), podczas gdy twierdzenie o zachowaniu nie zakłada. Takie podejście ma sens jeśli rozważamy czy dany osąd redukcyjny $e \mapsto e'$ jest częścią wniosku, czy też jest dany jako założenie. W przypadku twierdzenia o postępie interesuje nas czy e redukuje się do innego wyrażenia, czy nie, o ile jest dobrze typowane. Tym samym używamy pustego kontekstu aby wykluczyć zmienne wolne w e, które mogłyby uniemożliwić redukcję. Gdybyśmy dopuścili dowolny kontekst typujący Γ, twierdzenie o

postępie byłoby fałszywe, co ilustruje prosty przykład wyrażenia $e = x$, które nie jest wartością i nie jest redukowalne. W przypadku twierdzenia o zachowaniu zaczynamy od założenia $e \mapsto e'$. Teraz nie mamy powodu aby nie pozwalać zmiennym wolnym występować w e, ponieważ już wiemy, że zredukuje się do e'. Tym samym używamy metazmiennej Γ (biorącej wartości z klasy wszystkich kontekstów) zamiast pustego kontekstu.

Omawiane dwa twierdzenia razem gwarantują, że dobrze typowane domknięte wyrażenie nigdy nie zredukuje się do wyrażenia, które "zatnie się": albo będzie wartością, albo zredukuje się do innego dobrze typowanego wyrażenia. Rozważmy dobrze typowane wyrażenie e takie, że $\cdot \vdash e : A$ dla pewnego typu A. Jeśli e jest wartością, nie ma potrzeby go redukować. Jeśli nie, twierdzenie o postępie zapewnia, że istnieje wyrażenie e' takie, że $e \mapsto e'$, które również jest dobrze typowane i o tym samym typie A wobec twierdzenia o zachowaniu.

Podamy teraz dowody twierdzeń z wykorzystaniem indukcji reguł. Okazuje się, że próba bezpośredniego dowodu przez indukcję reguł nie może się powieść i tym samym konieczne będzie skorzystanie z dwóch lematów. Lematy te – zwane **lematem o formach kanonicznych** oraz **lematem o podstawianiu** – są na tyle ważne w teorii języków programowania, że warto zapamiętać ich nazwy.

7.4.1. *Dowód twierdzenia o postępie.* Dowód twierdzenia o postępie jest stosunkowo bezpośredni: twierdzenie jest postaci *"jeśli zachodzi J, zachodzi też $P(J)$"* i wystarczy zastosować indukcję reguł względem osądu J, który jest osądem typującym $\cdot \vdash e : A$. Załóżmy zatem, że $\cdot \vdash e : A$. Jeśli e jest wartością, trywialnie spełnione jest też $P(J)$, ponieważ osąd *"e jest wartością"* jest prawdziwy. Załóżmy wobec tego, że $\cdot \vdash e : A$ oraz że e nie jest wartością. Naszym celem jest teraz przeanalizowanie struktury dowodu $\cdot \vdash e : A$, co sprowadza się do trzech przypadków:

$$\frac{x : A \in \cdot}{\cdot \vdash e : A}\mathsf{Var} \qquad \frac{\cdot \vdash e_1 : A \to B \quad \cdot \vdash e_2 : A}{\cdot \vdash e_1 e_2 : B} \to \mathsf{E} \qquad \frac{\cdot \vdash e_b : \mathsf{bool} \quad \cdot \vdash e_1 : A \quad \cdot \vdash e_2 : A}{\cdot \vdash \mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A}\mathsf{If}$$

Przypadek Var nie jest możliwy, ponieważ $x : A$ nie może być elementem pustego kontekstu typującego. Innymi słowy, przesłanka $x : A \in \cdot$ nigdy nie jest spełniona. Pozostają do przeanalizowania przypadki $\to \mathsf{E}$ oraz If. Przeanalizujmy szczegółowo przypadek $\to \mathsf{E}$. Wobec zasady indukcji reguł, hipoteza indukcyjna względem pierwszej przesłanki $\cdot \vdash e_1 : A \to B$ oferuje dwie możliwości:

(1) e_1 jest wartością;
(2) e_1 nie jest wartością i redukuje się do nowego wyrażenia e_1', to znaczy $e_1 \mapsto e_1'$.

Jeśli zachodzi drugi przypadek, to znaleźliśmy wyrażenie, do którego $e_1 e_2$ się redukuje, a mianowicie $e_1' e_2$:

$$\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2} Lam.$$

Co się będzie jednak działo w pierwszym przypadku? Ponieważ e_1 jest typu $A \to B$, istnieje spora szansa, że jest λ-abstrakcją, a wówczas hipoteza indukcyjna względem drugiej przesłanki $\cdot \vdash e_2 : A$ daje dwie nowe możliwości, dla których możemy wykorzystać bądź to *Arg* bądź *App* celem udowodnienia własności postępu. Niestety, nie dysponujemy formalnym dowodem tego, że e_1 jest λ-abstrakcją; wiemy jedynie, że e_1 jest typu $A \to B$ przy założeniu pustego kontekstu typującego. Nasza intuicja podpowiada nam wszelako, że e_1 musi być λ-abstrakcją z uwagi na swój typ. Następujący lemat formalizuje naszą intuicję względem poprawnych – czy też "kanonicznych" – form dobrze typowanej wartości:

Lemat 4 (o formach kanonicznych)**.** (1) *Jeśli v jest wartością typu* bool*, to wówczas v jest* true *albo* false.
(2) *Jeśli v jest wartością typu $A \to B$, to wówczas v jest λ-abstrakcją $\lambda x : A.e$*

Dowód. Dowód prowadzimy przez analizę v.

Jeśli v jest wartością typu bool, to wówczas jedyne reguły typowania przypisujące typ boole'owski do danej wartości to True i False. Tym samym v jest stałą boole'owską true lub false. Zauważmy, że reguły Var, $\to \mathsf{E}$ oraz If mogą przypisać typ boole'owski, ale nigdy do wartości.

Jeśli v jest wartością typu $A \to B$, to wówczas jedyną regułą typowania przypisującą typ funkcyjny do danej wartości jest $\to \mathsf{I}$. Tym samym v musi być λ-abstrakcją postaci $\lambda x : A.e$. □

Możemy teraz przystąpić do dowodu twierdzenia o postępie.

Dowód. Dowód prowadzimy przez indukcję względem osądu $\cdot \vdash e : A$.

Jeśli e jest już wartością, dowód jest zakończony. Załóżmy zatem, że e nie jest wartością. Musimy rozważyć trzy przypadki.

Przypadek $\frac{x:A \in \cdot}{\cdot \vdash e:A}\mathsf{Var}$ gdzie $e = x$:
niemożliwe wobec $x : A \notin \cdot$

Przypadek $\frac{\cdot \vdash e_1:B \to A \quad \cdot \vdash e_2:B}{\cdot \vdash e_1 e_2:A} \to \mathsf{E}$ gdzie $e = e_1 e_2$:
e_1 jest wartością lub instnieje e_1' takie, że $e_1 \mapsto e_1'$ wobec założenia indukcyjnego względem $\cdot \vdash e_1 : B \to A$
e_2 jest wartością lub instnieje e_2' takie, że $e_2 \mapsto e_2'$ wobec założenia indukcyjnego względem $\cdot \vdash e_2 : B$.

Podprzypadek e_1 jest wartością i e_2 jest wartością:
$e_1 = \lambda x : B.e_1'$ wobec Lematu 4
$e_2 = v_2$ ponieważ e_2 jest wartością
$e_1 e_2 \mapsto [v_2/x]e_1'$ wobec reguły *App*
Niech $e' = [v_2/x]e_1'$

Podprzypadek e_1 jest wartością i istnieje e_2' takie, że $e_2 \mapsto e_2'$:
$e_1 = \lambda x : B.e_1'$ wobec Lematu 4
$e_1 e_2 \mapsto (\lambda x : B.e_1')e_2'$ wobec reguły *Arg*
Niech $e' = (\lambda x : B.e_1')e_2'$

Podprzypadek istnieje e_1' takie, że $e_1 \mapsto e_1'$:
$e_1 e_2 \mapsto e_1' e_2$ wobec reguły *Lam*
Niech $e' = e_1' e_2$

Przypadek $\frac{\cdot \vdash e_b:\mathsf{bool} \quad \cdot \vdash e_1:A \quad \cdot \vdash e_2:A}{\cdot \vdash \mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2:A}\mathsf{If}$ gdzie $e = \mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$:
e_b jest wartością lub istnieje e_b' takie, że $e_b \mapsto e_b'$ wobec założenia indukcyjnego względem $\cdot \vdash e_b : \mathsf{bool}$

Podprzypadek e_b jest wartością:
e_b jest true albo false wobec Lematu 4
$\mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_1$ albo $\mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_2$ wobec reguły $\mathit{If}_{\mathsf{true}}$ albo $\mathit{If}_{\mathsf{false}}$
Niech $e' = e_1$ albo $e' = e_2$

Podprzypadek istnieje e'_b takie, że $e_b \mapsto e'_b$:
$\mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto \mathsf{if}\ e'_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ wobec reguły *If*
Niech $e' = \mathsf{if}\ e'_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$. □

7.4.2. *Dowód twierdzenia o zachowaniu.* Dowód twierdzenia o zachowaniu nie jest tak bezpośredni jak dowód twierdzenia o postępie, ponieważ poprzednik implikacji *"Jeśli..."* w twierdzeniu zawiera dwa osądy: $\Gamma \vdash e : A$ oraz $e \mapsto e'$. Musimy zatem zadecydować do którego z osądów $\Gamma \vdash e : A$ oraz $e \mapsto e'$ chcemy zastosować indukcję reguł. Okazuje się, że twierdzenie o zachowaniu jest jednym z tych specjalnych przypadków, w których taki wybór nie ma znaczenia.

Przypuśćmy, że zechcielibyśmy zastosować indukcję reguł do osądu $e \mapsto e'$. Jako że mamy sześć reguł redukcyjnych, musielibyśmy rozważyć co najmniej sześć przypadków. Główne pytanie, jakie się nasuwa, brzmi: które z nich rozważymy jako pierwsze?

Jako "regułę kciuka" możemy przyjąć, że gdy kiedykolwiek dowodzimy własności, o której spodziewamy się, że zachodzi, zaczynamy od rozważenia możliwie najtrudniejszego przypadku. W naszej sytuacji intuicyjnie najtrudniejszy jest przypadek, w którym $e \mapsto e'$ jest dowodzone za pomocą reguły *App*, jako że podstawianie może tu zamienić aplikację e w zupełnie inną formę wyrażenia, na przykład w konstrukcję warunkową.

Rozważmy zatem przypadek $(\lambda x : A.e)v \mapsto [v/x]e$. Naszym celem jest użycie założenia $\Gamma \vdash (\lambda x : A.e)v : C$ do dowodu $\Gamma \vdash [v/x]e : C$. Osąd typujący $\Gamma \vdash (\lambda x : A.e)v : C$ musi mieć następujące drzewko dowodowe:

$$\dfrac{\dfrac{\Gamma, x : A \vdash e : C}{\Gamma \vdash \lambda x : A.e : A \to C}\to \mathsf{I} \qquad \Gamma \vdash v : A}{\Gamma \vdash (\lambda x : A.e)v : C}\to \mathsf{E}$$

Wobec tego naszym nowym celem jest użycie dwóch założeń $\Gamma, x : A \vdash e : C$ oraz $\Gamma \vdash v : A$ do dowodu $\Gamma \vdash [v/x]e : C$. Następujący lemat uogólnia ten problem:

Lemat 5 (o podstawianiu)**.** *Jeśli* $\Gamma \vdash e : A$ *oraz* $\Gamma, x : A \vdash e' : C$*, to wówczas* $\Gamma \vdash [e/x]e' : C$*.*

Lemat o podstawianiu jest podobny do twierdzenia o zachowaniu w tym sensie, że poprzednik implikacji *"Jeśli..."* zawiera dwa osądy. W tym przypadku musimy jednak bardzo ostrożnie wybrać osąd do zastosowania indukcji reguł, albowiem zły wybór może doprowadzić do porażki w próbie dowodu. Kluczowa obserwacja jest taka, że $[e/x]e'$ analizuje strukturę e', a nie e. Innymi słowy, $[e/x]e'$ wyszukuje każde wystąpienie zmiennej x w e' tylko w celu zastąpienia jej przez e i w rezultacie nie wie nic o strukturze e. Tym samym właściwym osądem do zastosowania indukcji reguł jest $\Gamma, x : A \vdash e' : C$.

Dowód. Dowód prowadzimy metodą indukcji reguł ze względu na $\Gamma, x : A \vdash e' : C$. Przypomnijmy, że zakładamy, że wszystkie zmienne w kontekście typującym są parami rozłączne. Pokażemy tu cztery przypadki. Pierwsze dwa dotyczą sytuacji, gdzie e' jest zmienną. Pozostałe są podobne do przypadku dla reguły $\to \mathsf{E}$.

Przypadek $\frac{y:C \in \Gamma, x:A}{\Gamma, x:A \vdash y:C}\mathsf{Var}$ gdzie $e' = y$ oraz $y : C \in \Gamma$:
Jest to przypadek, w którym e' jest zmienną y inną niż x. Ponieważ $y \neq x$, przesłanka $y : C \in \Gamma, x : A$ pociąga dodatkowy warunek $y : C \in \Gamma$.
$\Gamma \vdash y : C$ wobec $y : C \in \Gamma$
$[e/x]y = y$ wobec $x \neq y$
$\Gamma \vdash [e/x]y : C$

Przypadek $\frac{}{\Gamma,x:A\vdash x:A}\mathsf{Var}$ gdzie $e' = x$ oraz $C = A$:
Jest to przypadek, w którym e' jest zmienną x.

$\Gamma \vdash e : A$	wobec założenia
$\Gamma \vdash [e/x]x : A$	$[e/x]x = e$

Przypadek $\frac{\Gamma,x:A,y:C_1\vdash e'':C_2}{\Gamma,x:A\vdash \lambda y:C_1.e'':C_1\to C_2}\to \mathsf{I}$ gdzie $e' = \lambda y : C_1.e''$ oraz $C = C_1 \to C_2$:
Tutaj możemy bez straty ogólności założyć, że y jest nową zmienną taką, że $y \notin FV(e)$ oraz $y \neq x$. Jeśli $y \in FV(e)$ albo $y = x$, możemy zawsze wybrać inną zmienną po zastosowaniu α-konwersji do $\lambda y : C_1.e''$.

$\Gamma, y : C_1 \vdash [e/x]e'' : C_2$	wobec założenia indukcyjnego
$\Gamma \vdash \lambda y : C_1.[e/x]e'' : C_1 \to C_2$	wobec reguły $\to \mathsf{I}$
$[e/x]\lambda y : C_1.e'' = \lambda y : C_1.[e/x]e''$	wobec $y \notin FV(e)$ oraz $x \neq y$
$\Gamma \vdash [e/x]\lambda y : C_1.e'' : C_1 \to C_2$	

Przypadek $\frac{\Gamma,x:A\vdash e_1:B\to C\ \ \Gamma,x:A\vdash e_2:B}{\Gamma,x:A\vdash e_1e_2:C}\to \mathsf{E}$ gdzie $e' = e_1e_2$:

$\Gamma \vdash [e/x]e_1 : B \to C$	wobec założenia indukcyjnego względem $\Gamma, x : A \vdash e_1 : B \to C$
$\Gamma \vdash [e/x]e_2 : B$	wobec założenia indukcyjnego względem $\Gamma, x : A \vdash e_2 : B$
$\Gamma \vdash [e/x]e_1[e/x]e_2 : C$	wobec reguły $\to \mathsf{E}$
$\Gamma \vdash [e/x]e_1e_2 : C$	wobec $[e/x]e_1e_2 = [e/x]e_1[e/x]e_2$. □

Możemy teraz przejść do dowodu twierdzenia o zachowaniu. Dowód będzie przebiegał przez indukcję reguł względem osądu $e \mapsto e'$. Wykorzytuje fakt, że istnieje tylko jedna reguła typująca dla każdej postaci wyrażenia. Na przykład, jedyny sposób udowodnienia $\Gamma \vdash e_1e_2 : A$ to zastosowanie reguły $\to \mathsf{E}$. Tym samym o systemie typów mówimy, że jest **kierowany syntaktycznie** w takim sensie, że **syntaktyczna** postać wyrażenia e w osądzie $\Gamma \vdash e : A$ decyduje – czy też **kieruje** – o regułach, jakie mają zostać zastosowane. Jako że syntaktyczne kierowanie systemem typów decyduje o jednoznaczności wyboru reguły typującej R do dowodu $\Gamma \vdash e : A$, o przesłankach reguły R możemy założyć, że są spełnione o ile tylko zachodzi $\Gamma \vdash e : A$. Na przykład, $\Gamma \vdash e_1e_2 : A$ może być udowodnione tylko za pomocą reguły $\to \mathsf{E}$, skąd możemy wnioskować, że dwie przesłanki $\Gamma \vdash e_1 : B \to A$ oraz $\Gamma \vdash e_2 : B$ zachodzą dla pewnego typu B. Obserwację tę będziemy nazywać **własnością odwracania**, która odwraca regułę typującą w taki sposób, że jej wniosek uzasadnia prawdziwość przesłanek. Własność odwracania sformułujemy jako osobny lemat.

Lemat 6 (własność odwracania)**.** *Załóżmy, że* $\Gamma \vdash e : C$.

(1) *Jeśli* $e = x$, *to* $x : C \in \Gamma$.
(2) *Jeśli* $e = \lambda x : A.e'$ *to* $C = A \to B$ *oraz* $\Gamma, x : A \vdash e' : B$ *dla pewnego typu* B.
(3) *Jeśli* $e = e_1e_2$ *to* $\Gamma \vdash e_1 : A \to C$ *oraz* $\Gamma \vdash e_2 : A$ *dla pewnego typu* A.
(4) *Jeśli* $e = \mathsf{true}$ *to* $C = \mathsf{bool}$.
(5) *Jeśli* $e = \mathsf{false}$ *to* $C = \mathsf{bool}$.
(6) *Jeśli* $e = \mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ *to* $\Gamma \vdash e_b : \mathsf{bool}$ *oraz* $\Gamma \vdash e_1 : C$ *i* $\Gamma \vdash e_2 : C$.

Dowód. Formalnie dowód należy przeprowadzić przez indukcję reguł względem osądu $\Gamma \vdash e : C$. Szczegóły zostawiamy jako nietrudne ćwiczenie. □

Przechodzimy do dowodu twierdzenia o zachowaniu.

Dowód. Dowód prowadzimy przez indukcję reguł względem osądu $e \mapsto e'$.

Przypadek $\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2} Lam$:

$\Gamma \vdash e_1 e_2 : A$ wobec założenia

$\Gamma \vdash e_1 : B \to A$ oraz $\Gamma \vdash e_2 : B$ dla pewnego typu B wobec Lematu 6

$\Gamma \vdash e_1' : B \to A$ wobec założenia indukcyjnego względem $e_1 \mapsto e_1'$ wraz z $\Gamma \vdash e_1 : B \to A$

$\Gamma \vdash e_1' e_2 : A$ wobec $\frac{\Gamma \vdash e_1' : B \to A \;\; \Gamma \vdash e_2 : B}{\Gamma \vdash e_1' e_2 : A} \to \mathsf{E}$

Przypadek $\frac{e_2 \mapsto e_2'}{(\lambda x{:}B.e_1') e_2 \mapsto (\lambda x{:}B.e_1') e_2'} Arg$:

$\Gamma \vdash (\lambda x : B.e_1') e_2 : A$ wobec założenia

$\Gamma \vdash \lambda x : B.e_1' : B \to A$ oraz $\Gamma \vdash e_2 : B$ wobec Lematu 6

$\Gamma \vdash e_2' : B$ wobec założenia indukcyjnego względem $e_2 \mapsto e_2'$ wraz z $\Gamma \vdash e_2 : B$

$\Gamma \vdash (\lambda x : B.e_1') e_2' : A$ wobec $\frac{\Gamma \vdash \lambda x{:}B.e_1' : B \to A \;\; \Gamma \vdash e_2' : B}{\Gamma \vdash (\lambda x{:}B.e_1') e_2' : A} \to \mathsf{E}$

Przypadek $\frac{}{(\lambda x{:}B.e_1') v \mapsto [v/x]e_1'} App$:

$\Gamma \vdash (\lambda x : B.e_1') v : A$ wobec założenia

$\Gamma \vdash \lambda x : B.e_1' : B \to A$ oraz $\Gamma \vdash v : B$ wobec Lematu 6

$\Gamma, x : B \vdash e_1' : A$ wobec Lematu 6 względem $\Gamma \vdash \lambda x : B.e_1' : B \to A$

$\Gamma \vdash [v/x]e_1' : A$ wobec Lematu 5 zastosowanego do $\Gamma \vdash v : B$ oraz $\Gamma, x : B \vdash e_1' : A$

Przypadek $\frac{e_b \mapsto e_b'}{\mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto \mathsf{if}\ e_b'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2} If$:

$\Gamma \vdash \mathsf{if}\ e_b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A$ wobec założenia

$\Gamma \vdash e_b : \mathsf{bool}$ oraz $\Gamma \vdash e_1 : A$ i $\Gamma \vdash e_2 : A$ wobec Lematu 6

$\Gamma \vdash e_b' : \mathsf{bool}$ wobec założenia indukcyjnego względem $e_b \mapsto e_b'$ wraz z $\Gamma \vdash e_b : \mathsf{bool}$

$\Gamma \vdash \mathsf{if}\ e_b'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A$ wobec $\frac{\Gamma \vdash e_b' : \mathsf{bool} \;\; \Gamma \vdash e_1 : A \;\; \Gamma \vdash e_2 : A}{\Gamma \vdash \mathsf{if}\ e_b'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A} \mathsf{If}$

Przypadek $\frac{}{\mathsf{if}\ \mathsf{true}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_1} If_{\mathsf{true}}$:

$\Gamma \vdash \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A$ wobec założenia

$\Gamma \vdash \mathsf{true} : \mathsf{bool}$ oraz $\Gamma \vdash e_1 : A$ i $\Gamma \vdash e_2 : A$ wobec Lematu 6

$\Gamma \vdash e_1 : A$

Przypadek $\frac{}{\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_2} If_{\mathsf{false}}$:

Podobnie jak przypadek If_{true}. $\square$