Λάμβδα Λογισμός με Τύπους - II

Γιάννης Κασσιός

Δομή της Σημερινής Διάλεξης

- Αναδρομικοί Τύποι
  - παραδείγματα εκφραστικότητας αναδρομικών τύπων
- Εξαγωγή Τύπου σε σύστημα όμοιο με τη Haskell
  - αλγόριθμος Hindley-Milner
- Πολυμορφισμός
  - let-πολυμορφισμός (Haskell, ML)
  - σύστημα F
  - σύστημα $F_{<:}$

Δομή της Σημερινής Διάλεξης

- Αναδρομικοί Τύποι
  - παραδείγματα εκφραστικότητας αναδρομικών τύπων
- Εξαγωγή Τύπου σε σύστημα όμοιο με τη Haskell
  - αλγόριθμος Hindley-Milner
- Πολυμορφισμός
  - let-πολυμορφισμός (Haskell, ML)
  - σύστημα F
  - σύστημα $F_{<:}$

Δομή της Σημερινής Διάλεξης

- Αναδρομικοί Τύποι
  - παραδείγματα εκφραστικότητας αναδρομικών τύπων
- Εξαγωγή Τύπου σε σύστημα όμοιο με τη Haskell
  - αλγόριθμος Hindley-Milner
- Πολυμορφισμός
  - let-πολυμορφισμός (Haskell, ML)
  - σύστημα F
  - σύστημα $F_{<:}$

Αναδρομικοί Τύποι

- Αναδρομικος τύπος (recursive type) = τύπος του οποίου ο ορισμός περιέχει τον εαυτό του
  - άπειροι τύποι με επαναλαμβανόμενη δομή (π.χ. λίστες, δέντρα κτλ.)
- Στη Haskell: αλγεβρικοί τύποι (σε συνδυασμό με εναλλακτικούς τύπους και πολυμορφισμό)
- Γενικότερα: συνδυάζονται και με άλλους τύπους της γλώσσας όπως εγγραφές, συναρτήσεις
- Γράφουμε $\mu X.F\ X$ για τον τύπο που ορίζεται $T = F\ T$, όπου F μία συνάρτηση τύπων

Αναδρομικοί Τύποι

- Αναδρομικος τύπος (recursive type) = τύπος του οποίου ο ορισμός περιέχει τον εαυτό του
  - άπειροι τύποι με επαναλαμβανόμενη δομή (π.χ. λίστες, δέντρα κτλ.)
- Στη Haskell: αλγεβρικοί τύποι (σε συνδυασμό με εναλλακτικούς τύπους και πολυμορφισμό)
- Γενικότερα: συνδυάζονται και με άλλους τύπους της γλώσσας όπως εγγραφές, συναρτήσεις
- Γράφουμε $\mu X.F\ X$ για τον τύπο που ορίζεται $T = F\ T$, όπου F μία συνάρτηση τύπων

Αναδρομικοί Τύποι

- Αναδρομικος τύπος (recursive type) = τύπος του οποίου ο ορισμός περιέχει τον εαυτό του
  - άπειροι τύποι με επαναλαμβανόμενη δομή (π.χ. λίστες, δέντρα κτλ.)
- Στη Haskell: αλγεβρικοί τύποι (σε συνδυασμό με εναλλακτικούς τύπους και πολυμορφισμό)
- Γενικότερα: συνδυάζονται και με άλλους τύπους της γλώσσας όπως εγγραφές, συναρτήσεις
- Γράφουμε $\mu X.F\ X$ για τον τύπο που ορίζεται $T = F\ T$, όπου F μία συνάρτηση τύπων

Αναδρομικοί Τύποι

- Αναδρομικος τύπος (recursive type) = τύπος του οποίου ο ορισμός περιέχει τον εαυτό του
  - άπειροι τύποι με επαναλαμβανόμενη δομή (π.χ. λίστες, δέντρα κτλ.)
- Στη Haskell: αλγεβρικοί τύποι (σε συνδυασμό με εναλλακτικούς τύπους και πολυμορφισμό)
- Γενικότερα: συνδυάζονται και με άλλους τύπους της γλώσσας όπως εγγραφές, συναρτήσεις
- Γράφουμε $\mu X.F\ X$ για τον τύπο που ορίζεται $T = F\ T$, όπου F μία συνάρτηση τύπων

Αναδρομικοί Τύποι - Παραδείγματα

Συνδυασμός με Εναλλακτικούς Τύπους (Αλγεβρικοί Τύποι): Λίστες Φυσικών

- Μία λίστα φυσικών αριθμών είναι:
  - η κενή λίστα ή
  - μία μη κενή λίστα που περιλαμβάνει ένα φυσικό αριθμό ακολουθούμενο από μία λίστα φυσικών αριθμών
- Συμβολίζουμε την κενή λίστα με την ετικέτα `nil` και τη μη κενή λίστα με την ετικέτα `cons`. Στη Haskell
  - `nil` $\leftrightarrow$ `[]`
  - `cons` $\leftrightarrow$ `:`
- Ορισμός:

$$\texttt{NatList} = \mu X.\langle \texttt{nil}:\{\}, \texttt{cons}:\{\texttt{h}:\texttt{Nat}, \texttt{t}:X\}\rangle$$

- Μία λίστα φυσικών αριθμών είναι:
  - η κενή λίστα ή
  - μία μη κενή λίστα που περιλαμβάνει ένα φυσικό αριθμό ακολουθούμενο από μία λίστα φυσικών αριθμών
- Συμβολίζουμε την κενή λίστα με την ετικέτα `nil` και τη μη κενή λίστα με την ετικέτα `cons`. Στη Haskell
  - `nil` $\leftrightarrow$ `[]`
  - `cons` $\leftrightarrow$ `:`
- Ορισμός:

$$\texttt{NatList} = \mu X.\langle \texttt{nil}:\{\}, \texttt{cons}:\{\texttt{h}:\texttt{Nat}, \texttt{t}:X\}\rangle$$

- Μία λίστα φυσικών αριθμών είναι:
  - η κενή λίστα ή
  - μία μη κενή λίστα που περιλαμβάνει ένα φυσικό αριθμό ακολουθούμενο από μία λίστα φυσικών αριθμών
- Συμβολίζουμε την κενή λίστα με την ετικέτα `nil` και τη μη κενή λίστα με την ετικέτα `cons`. Στη Haskell
  - `nil` $\leftrightarrow$ `[]`
  - `cons` $\leftrightarrow$ `:`
- Ορισμός:

$$\texttt{NatList} = \mu X.\texttt{<nil:\{\}, cons:\{h:Nat,t:X\}>}$$

- Η 'πεινασμένη' συνάρτηση παίρνει έναν φυσικό και επιστρέφει τον εαυτό της:

$$\texttt{hungry} = \lambda \texttt{n:Nat.hungry}$$

- Τύπος και ορισμός:

$$\texttt{Hungry} = \mu \texttt{X.Nat} \to \texttt{X}$$
$$\texttt{hungry} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Hungry.} \lambda \texttt{n:Nat.f})$$

- Όμοια, αλλά πιο χρήσιμα: ρεύματα (streams):

$$\mu X.P \to \{\texttt{result:}R, \texttt{next:}X\}$$

- Παράδειγμα:

$$\texttt{Sum} = \mu \texttt{X.Nat} \to \{\texttt{result:Nat,next:X}\}$$
$$\texttt{sum} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Sum.} \lambda \texttt{i:Nat.} \lambda \texttt{n:Nat.}$$
$$\{\texttt{result} = \texttt{i+n},\ \texttt{next} = \texttt{f(i+n)}\})\ 0$$

- Η 'πεινασμένη' συνάρτηση παίρνει έναν φυσικό και επιστρέφει τον εαυτό της:

$$\texttt{hungry} = \lambda \texttt{n:Nat.hungry}$$

- Τύπος και ορισμός:

$$\texttt{Hungry} = \mu \texttt{X.Nat} \to \texttt{X}$$
$$\texttt{hungry} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Hungry}.\lambda \texttt{n:Nat.f})$$

- Όμοια, αλλά πιο χρήσιμα: ρεύματα (streams):

$$\mu X.P \to \{\texttt{result:}R, \texttt{next:}X\}$$

- Παράδειγμα:

$$\texttt{Sum} = \mu \texttt{X.Nat} \to \{\texttt{result:Nat,next:X}\}$$
$$\texttt{sum} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Sum}.\lambda \texttt{i:Nat}.\lambda \texttt{n:Nat}.$$
$$\{\texttt{result} = \texttt{i+n},\ \texttt{next} = \texttt{f(i+n)}\})\ 0$$

- Η 'πεινασμένη' συνάρτηση παίρνει έναν φυσικό και επιστρέφει τον εαυτό της:

$$\texttt{hungry} = \lambda \texttt{n:Nat.hungry}$$

- Τύπος και ορισμός:

$$\texttt{Hungry} = \mu \texttt{X.Nat} \to \texttt{X}$$
$$\texttt{hungry} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Hungry}.\lambda \texttt{n:Nat.f})$$

- Όμοια, αλλά πιο χρήσιμα: ρεύματα (streams):

$$\mu X.P \to \{\texttt{result:}R, \texttt{next:}X\}$$

- Παράδειγμα:

$$\texttt{Sum} = \mu \texttt{X.Nat} \to \{\texttt{result:Nat,next:X}\}$$
$$\texttt{sum} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Sum}.\lambda \texttt{i:Nat}.\lambda \texttt{n:Nat}.$$
$$\{\texttt{result} = \texttt{i+n},\ \texttt{next} = \texttt{f(i+n)}\})\ 0$$

- Η 'πεινασμένη' συνάρτηση παίρνει έναν φυσικό και επιστρέφει τον εαυτό της:

$$\texttt{hungry} = \lambda \texttt{n:Nat.hungry}$$

- Τύπος και ορισμός:

$$\texttt{Hungry} = \mu \texttt{X.Nat} \to \texttt{X}$$
$$\texttt{hungry} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Hungry}.\lambda \texttt{n:Nat.f})$$

- Όμοια, αλλά πιο χρήσιμα: ρεύματα (streams):

$$\mu X.P \to \{\texttt{result:}R, \texttt{next:}X\}$$

- Παράδειγμα:

$$\texttt{Sum} = \mu \texttt{X.Nat} \to \{\texttt{result:Nat,next:X}\}$$
$$\texttt{sum} = \texttt{fix}(\lambda \texttt{f:Nat} \to \texttt{Sum}.\lambda \texttt{i:Nat}.\lambda \texttt{n:Nat}.$$
$$\{\texttt{result} = \texttt{i+n},\ \texttt{next} = \texttt{f(i+n)}\})\ \texttt{0}$$

Αναδρομικοί Τύποι - Παραδείγματα

Συναρτησιακά Αντικείμενα

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

```
Counter = μX.{get:Nat, inc:Nat→X}
counter = fix(λf:Nat→Counter.λn:Nat.
            {get=n, inc=λx:Nat.f(x+n)})
```

- Χρήση:

```
    counter 0.inc 2.get
=   {get=0,inc=λx:Nat.counter(x+0)}
    .inc 2.get
=   (λx:Nat.counter(x+0))2.get
=   counter 2.get
=   {get=2,inc=λx:Nat.counter(x+2)}.get
=   2
```

Αναδρομικοί Τύποι - Παραδείγματα

Συναρτησιακά Αντικείμενα

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

$$\texttt{Counter} = \mu X.\{\texttt{get:Nat}, \texttt{inc:Nat} \to X\}$$
$$\texttt{counter} = \texttt{fix}(\lambda f:\texttt{Nat} \to \texttt{Counter}.\lambda n:\texttt{Nat}.\ \{\texttt{get}=n, \texttt{inc}=\lambda x:\texttt{Nat}.f(x+n)\})$$

- Χρήση:

$$\begin{aligned}
&\texttt{counter 0.inc 2.get}\\
=\ &\{\texttt{get}=0, \texttt{inc}=\lambda x:\texttt{Nat}.\texttt{counter}(x+0)\}\texttt{.inc 2.get}\\
=\ &(\lambda x:\texttt{Nat}.\texttt{counter}(x+0))2\texttt{.get}\\
=\ &\texttt{counter 2.get}\\
=\ &\{\texttt{get}=2, \texttt{inc}=\lambda x:\texttt{Nat}.\texttt{counter}(x+2)\}\texttt{.get}\\
=\ &2
\end{aligned}$$

Αναδρομικοί Τύποι - Παραδείγματα

Συναρτησιακά Αντικείμενα

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

$$\texttt{Counter} = \mu X.\{\texttt{get:Nat}, \texttt{inc:Nat} \to X\}$$
$$\texttt{counter} = \texttt{fix}(\lambda f:\texttt{Nat} \to \texttt{Counter}.\lambda n:\texttt{Nat}.\ \{\texttt{get}=n, \texttt{inc}=\lambda x:\texttt{Nat}.f(x+n)\})$$

- Χρήση:

$$\begin{aligned} & \texttt{counter 0.inc 2.get} \\ = \ & \{\texttt{get}=0, \texttt{inc}=\lambda x:\texttt{Nat}.\texttt{counter}(x+0)\} \\ & \texttt{.inc 2.get} \\ = \ & (\lambda x:\texttt{Nat}.\texttt{counter}(x+0))2\texttt{.get} \\ = \ & \texttt{counter 2.get} \\ = \ & \{\texttt{get}=2, \texttt{inc}=\lambda x:\texttt{Nat}.\texttt{counter}(x+2)\}\texttt{.get} \\ = \ & 2 \end{aligned}$$

Αναδρομικοί Τύποι - Παραδείγματα

Συναρτησιακά Αντικείμενα

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

$$\texttt{Counter} = \mu X.\{\texttt{get:Nat}, \texttt{inc:Nat} \to X\}$$
$$\texttt{counter} = \texttt{fix}(\lambda f:\texttt{Nat} \to \texttt{Counter}.\lambda n:\texttt{Nat}.\ \{\texttt{get=n}, \texttt{inc=}\lambda x:\texttt{Nat}.f(x+n)\})$$

- Χρήση:

$$\begin{aligned} & \texttt{counter 0.inc 2.get} \\ = \ & \texttt{\{get=0,inc=}\lambda x:\texttt{Nat.counter(x+0)\}} \\ & \texttt{.inc 2.get} \\ = \ & (\lambda x:\texttt{Nat.counter(x+0)})\texttt{2.get} \\ = \ & \texttt{counter 2.get} \\ = \ & \texttt{\{get=2,inc=}\lambda x:\texttt{Nat.counter(x+2)\}.get} \\ = \ & \texttt{2} \end{aligned}$$

Αναδρομικοί Τύποι - Παραδείγματα

Συναρτησιακά Αντικείμενα

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

$$\begin{aligned}&\texttt{Counter} = \mu\texttt{X.\{get:Nat, inc:Nat}\rightarrow\texttt{X\}}\\&\texttt{counter} = \texttt{fix}(\lambda\texttt{f:Nat}\rightarrow\texttt{Counter.}\lambda\texttt{n:Nat.}\\&\qquad\texttt{\{get=n, inc=}\lambda\texttt{x:Nat.f(x+n)\}})\end{aligned}$$

- Χρήση:

$$\begin{aligned}&\texttt{counter 0.inc 2.get}\\=\;&\texttt{\{get=0,inc=}\lambda\texttt{x:Nat.counter(x+0)\}}\\&\texttt{.inc 2.get}\\=\;&(\lambda\texttt{x:Nat.counter(x+0))2.get}\\=\;&\texttt{counter 2.get}\\=\;&\texttt{\{get=2,inc=}\lambda\texttt{x:Nat.counter(x+2)\}.get}\\=\;&\texttt{2}\end{aligned}$$

Αναδρομικοί Τύποι - Παραδείγματα

Συναρτησιακά Αντικείμενα

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

$$\begin{aligned}&\texttt{Counter} = \mu\texttt{X.\{get:Nat, inc:Nat}\rightarrow\texttt{X\}}\\&\texttt{counter} = \texttt{fix}(\lambda\texttt{f:Nat}\rightarrow\texttt{Counter.}\lambda\texttt{n:Nat.}\\&\qquad\texttt{\{get=n, inc=}\lambda\texttt{x:Nat.f(x+n)\}})\end{aligned}$$

- Χρήση:

$$\begin{aligned}&\texttt{counter 0.inc 2.get}\\=\;&\texttt{\{get=0,inc=}\lambda\texttt{x:Nat.counter(x+0)\}}\\&\texttt{.inc 2.get}\\=\;&(\lambda\texttt{x:Nat.counter(x+0))2.get}\\=\;&\texttt{counter 2.get}\\=\;&\texttt{\{get=2,inc=}\lambda\texttt{x:Nat.counter(x+2)\}.get}\\=\;&\texttt{2}\end{aligned}$$

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

$$\begin{aligned}&\texttt{Counter} = \mu\texttt{X.\{get:Nat, inc:Nat}\rightarrow\texttt{X\}}\\&\texttt{counter} = \texttt{fix}(\lambda\texttt{f:Nat}\rightarrow\texttt{Counter.}\lambda\texttt{n:Nat.}\\&\qquad\texttt{\{get=n, inc=}\lambda\texttt{x:Nat.f(x+n)\}})\end{aligned}$$

- Χρήση:

$$\begin{aligned}&\texttt{counter 0.inc 2.get}\\=\;&\texttt{\{get=0,inc=}\lambda\texttt{x:Nat.counter(x+0)\}}\\&\texttt{.inc 2.get}\\=\;&(\lambda\texttt{x:Nat.counter(x+0))2.get}\\=\;&\texttt{counter 2.get}\\=\;&\texttt{\{get=2,inc=}\lambda\texttt{x:Nat.counter(x+2)\}.get}\\=\;&2\end{aligned}$$

Αναδρομικοί Τύποι - Παραδείγματα

Συναρτησιακά Αντικείμενα

- Στο συναρτησιακό αντικειμενοστρεφές μοντέλο, αντί να αλλάζουμε κατάσταση επιστρέφουμε ένα νέο αντικείμενο
- Αντικείμενο μετρητής:

$$\begin{aligned}&\texttt{Counter} = \mu\texttt{X.\{get:Nat, inc:Nat}\rightarrow\texttt{X\}}\\&\texttt{counter} = \texttt{fix}(\lambda\texttt{f:Nat}\rightarrow\texttt{Counter.}\lambda\texttt{n:Nat.}\\&\qquad\texttt{\{get=n, inc=}\lambda\texttt{x:Nat.f(x+n)\}})\end{aligned}$$

- Χρήση:

$$\begin{aligned}&\texttt{counter 0.inc 2.get}\\=\;&\texttt{\{get=0,inc=}\lambda\texttt{x:Nat.counter(x+0)\}}\\&\texttt{.inc 2.get}\\=\;&(\lambda\texttt{x:Nat.counter(x+0))2.get}\\=\;&\texttt{counter 2.get}\\=\;&\texttt{\{get=2,inc=}\lambda\texttt{x:Nat.counter(x+2)\}.get}\\=\;&\texttt{2}\end{aligned}$$

Αναδρομικοί Τύποι - Παραδείγματα

Τύπος του Y

- Ο όρος Y δεν έχει τύπο στον λ-λογισμό με απλούς τύπους:

$$Y = \lambda \mathtt{f}.(\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))$$

  - ο τύπος T_x της τυπικής $\mathtt{x}$ θα έπρεπε να ικανοποιεί:

$$T_x = T_x \rightarrow T_r$$

- Με αναδρομικούς τύπους, ο τύπος αυτός υπάρχει:

$$\mu \mathtt{X}.\mathtt{X} \rightarrow T_r$$

- Ο όρος $\mathtt{fix}_T$ για αναδρομικό υπολογισμό σε τύπο T είναι:

$$\mathtt{fix}_T : (T \rightarrow T) \rightarrow T$$
$$R_T = \mu \mathtt{X}.\mathtt{X} \rightarrow T$$
$$\mathtt{fix}_T = \lambda \mathtt{f}{:}T \rightarrow T.\ (\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))(\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))$$

Αναδρομικοί Τύποι - Παραδείγματα

Τύπος του Y

- Ο όρος Y δεν έχει τύπο στον λ-λογισμό με απλούς τύπους:

$$Y = \lambda \mathtt{f}.(\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))$$

  - ο τύπος T_x της τυπικής $\mathtt{x}$ θα έπρεπε να ικανοποιεί:

$$T_x = T_x \rightarrow T_r$$

- Με αναδρομικούς τύπους, ο τύπος αυτός υπάρχει:

$$\mu \mathtt{X}.\mathtt{X} \rightarrow T_r$$

- Ο όρος $\mathtt{fix}_T$ για αναδρομικό υπολογισμό σε τύπο T είναι:

$$\mathtt{fix}_T : (T \rightarrow T) \rightarrow T$$
$$R_T = \mu \mathtt{X}.\mathtt{X} \rightarrow T$$
$$\mathtt{fix}_T = \lambda \mathtt{f}{:}T \rightarrow T.\ (\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))(\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))$$

Αναδρομικοί Τύποι - Παραδείγματα

Τύπος του Y

- Ο όρος Y δεν έχει τύπο στον λ-λογισμό με απλούς τύπους:

$$Y = \lambda \mathtt{f}.(\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))$$

  - ο τύπος T_x της τυπικής $\mathtt{x}$ θα έπρεπε να ικανοποιεί:

$$T_x = T_x \rightarrow T_r$$

- Με αναδρομικούς τύπους, ο τύπος αυτός υπάρχει:

$$\mu \mathtt{X}.\mathtt{X} \rightarrow T_r$$

- Ο όρος $\mathtt{fix}_T$ για αναδρομικό υπολογισμό σε τύπο T είναι:

$$\mathtt{fix}_T : (T \rightarrow T) \rightarrow T$$
$$R_T = \mu \mathtt{X}.\mathtt{X} \rightarrow T$$
$$\mathtt{fix}_T = \lambda \mathtt{f}{:}T \rightarrow T.\ (\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))(\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))$$

Αναδρομικοί Τύποι - Παραδείγματα

Τύπος του Y

- Ο όρος Y δεν έχει τύπο στον λ-λογισμό με απλούς τύπους:

$$Y = \lambda \mathtt{f}.(\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{x}\ \mathtt{x}))$$

  - ο τύπος T_x της τυπικής $\mathtt{x}$ θα έπρεπε να ικανοποιεί:

$$T_x = T_x \rightarrow T_r$$

- Με αναδρομικούς τύπους, ο τύπος αυτός υπάρχει:

$$\mu \mathtt{X}.\mathtt{X} \rightarrow T_r$$

- Ο όρος $\mathtt{fix}_T$ για αναδρομικό υπολογισμό σε τύπο T είναι:

$$\mathtt{fix}_T : (T \rightarrow T) \rightarrow T$$
$$R_T = \mu \mathtt{X}.\mathtt{X} \rightarrow T$$
$$\mathtt{fix}_T = \lambda \mathtt{f}{:}T \rightarrow T.\ (\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))(\lambda \mathtt{x}{:}R_T.\ \mathtt{f}(\mathtt{x}\ \mathtt{x}))$$

Αναδρομικοί Τύποι - Παραδείγματα

Τύποι για Ολόκληρο το λ-λογισμό χωρίς Τύπους

- Όλες οι τιμές είναι συναρτήσεις. Τύπος όλων των τιμών:

$$\mathtt{D}=\mu\mathtt{X}\,.\,\mathtt{X}\to\mathtt{X}$$

- Ομοίως μπορούμε να δώσουμε τύπους σε λ-λογισμό + πρωτογενείς τιμές
  - δηλωτική σημασιολογία για λ-λογισμούς
- Παράδοξο; $\mathtt{D}=\mathtt{D}\to\mathtt{D}$
  - στη θεωρία συνόλων δεν ισχύει για κανένα αριθμήσιμο σύνολο D:

$$|D \to D| = |D|^{|D|}$$

  - εδώ ισχύει: $X \to Y$ είναι το σύνολο των υπολογίσιμων συναρτήσεων από X προς Y (λ-όροι)

$$|\mathtt{D}| = |\mathtt{D}\to\mathtt{D}| = \aleph_0$$

Αναδρομικοί Τύποι - Παραδείγματα

Τύποι για Ολόκληρο το λ-λογισμό χωρίς Τύπους

- Όλες οι τιμές είναι συναρτήσεις. Τύπος όλων των τιμών:

$$\mathsf{D}=\mu\mathsf{X}\,.\,\mathsf{X}\to\mathsf{X}$$

- Ομοίως μπορούμε να δώσουμε τύπους σε λ-λογισμό + πρωτογενείς τιμές
  - δηλωτική σημασιολογία για λ-λογισμούς
- Παράδοξο; $\mathsf{D}=\mathsf{D}\to\mathsf{D}$
  - στη θεωρία συνόλων δεν ισχύει για κανένα αριθμήσιμο σύνολο D:

$$|D \to D| = |D|^{|D|}$$

  - εδώ ισχύει: $X \to Y$ είναι το σύνολο των υπολογίσιμων συναρτήσεων από X προς Y (λ-όροι)

$$|\mathsf{D}| = |\mathsf{D}\to\mathsf{D}| = \aleph_0$$

- Όλες οι τιμές είναι συναρτήσεις. Τύπος όλων των τιμών:

$$\mathtt{D}=\mu\mathtt{X}\,.\,\mathtt{X}\to\mathtt{X}$$

- Ομοίως μπορούμε να δώσουμε τύπους σε λ-λογισμό + πρωτογενείς τιμές
  - δηλωτική σημασιολογία για λ-λογισμούς
- Παράδοξο; $\mathtt{D}=\mathtt{D}\to\mathtt{D}$
  - στη θεωρία συνόλων δεν ισχύει για κανένα αριθμήσιμο σύνολο D:

$$|D \to D| = |D|^{|D|}$$

  - εδώ ισχύει: $X \to Y$ είναι το σύνολο των υπολογίσιμων συναρτήσεων από X προς Y (λ-όροι)

$$|\mathtt{D}| = |\mathtt{D}\to\mathtt{D}| = \aleph_0$$

- Όλες οι τιμές είναι συναρτήσεις. Τύπος όλων των τιμών:

$$\mathtt{D}=\mu\mathtt{X}\,.\,\mathtt{X}\to\mathtt{X}$$

- Ομοίως μπορούμε να δώσουμε τύπους σε λ-λογισμό + πρωτογενείς τιμές
  - δηλωτική σημασιολογία για λ-λογισμούς
- Παράδοξο; $\mathtt{D}=\mathtt{D}\to\mathtt{D}$
  - στη θεωρία συνόλων δεν ισχύει για κανένα αριθμήσιμο σύνολο D:

$$|D \to D| = |D|^{|D|}$$

  - εδώ ισχύει: $X \to Y$ είναι το σύνολο των <u>υπολογίσιμων</u> συναρτήσεων από X προς Y (λ-όροι)

$$|\mathtt{D}| = |\mathtt{D}\to\mathtt{D}| = \aleph_0$$

- Σύνταξη:

$$\begin{array}{rll} T ::= & \ldots & \\ & X & \text{μεταβλητή τύπων} \\ & \mu X.T & \text{αναδρομικός τύπος} \end{array}$$

- Κανόνες Τύπων:

$$\frac{\Gamma \vdash t : T[X := \mu X.T]}{\Gamma \vdash t : \mu X.T}(\text{αν.τ.-fold})$$

$$\frac{\Gamma \vdash t : \mu X.T}{\Gamma \vdash t : T[X := \mu X.T]}(\text{αν.τ.-unfold})$$

- Σύνταξη:

$$\begin{array}{rll} T ::= & \ldots & \\ & X & \text{μεταβλητή τύπων} \\ & \mu X.T & \text{αναδρομικός τύπος} \end{array}$$

- Κανόνες Τύπων:

$$\frac{\Gamma \vdash t : T[X := \mu X.T]}{\Gamma \vdash t : \mu X.T}\text{(αν.τ.-fold)}$$

$$\frac{\Gamma \vdash t : \mu X.T}{\Gamma \vdash t : T[X := \mu X.T]}\text{(αν.τ.-unfold)}$$

Εξαγωγή Τύπων

- Εξαγωγή τύπων (type inference): δίνεται μία έκφραση λ-λογισμού χωρίς τύπους και ζητούνται όλοι οι τύποι που μπορεί να πάρει αυτή η έκφραση
  - ο προγραμματιστής δε χρειάζεται να γράφει τύπους: η γλώσσα τους ανακαλύπτει
  - εισαγωγικό θέμα στη μελέτη του πολυμορφισμού
- Ένας όρος σε λ-λογισμό χωρίς τύπους μπορεί να δέχεται 0,1 ή περισσότερους τύπους:
  - `if 5 then 2 else 0` `true`
    λ`x.x`

Εξαγωγή Τύπων

- Εξαγωγή τύπων (type inference): δίνεται μία έκφραση λ-λογισμού χωρίς τύπους και ζητούνται όλοι οι τύποι που μπορεί να πάρει αυτή η έκφραση
  - ο προγραμματιστής δε χρειάζεται να γράφει τύπους: η γλώσσα τους ανακαλύπτει
  - εισαγωγικό θέμα στη μελέτη του πολυμορφισμού
- Ένας όρος σε λ-λογισμό χωρίς τύπους μπορεί να δέχεται 0,1 ή περισσότερους τύπους:
  - `if 5 then 2 else 0` `true`
    `λx.x`

Εξαγωγή Τύπων

- Εξαγωγή τύπων (type inference): δίνεται μία έκφραση λ-λογισμού χωρίς τύπους και ζητούνται όλοι οι τύποι που μπορεί να πάρει αυτή η έκφραση
  - ο προγραμματιστής δε χρειάζεται να γράφει τύπους: η γλώσσα τους ανακαλύπτει
  - εισαγωγικό θέμα στη μελέτη του πολυμορφισμού
- Ένας όρος σε λ-λογισμό χωρίς τύπους μπορεί να δέχεται 0,1 ή περισσότερους τύπους:
  - `if 5 then 2 else 0` `true`
    `λx.x`

Εξαγωγή Τύπων

- Εξαγωγή τύπων (type inference): δίνεται μία έκφραση λ-λογισμού χωρίς τύπους και ζητούνται όλοι οι τύποι που μπορεί να πάρει αυτή η έκφραση
  - ο προγραμματιστής δε χρειάζεται να γράφει τύπους: η γλώσσα τους ανακαλύπτει
  - εισαγωγικό θέμα στη μελέτη του πολυμορφισμού
- Ένας όρος σε λ-λογισμό χωρίς τύπους μπορεί να δέχεται 0,1 ή περισσότερους τύπους:
  - `if 5 then 2 else 0` `true` λ`x.x`

Εξαγωγή Τύπων
Σύστημα υπό Εξέταση - I

- Παίρνουμε τον λ-λογισμό χωρίς τύπους + φυσικούς + αληθοτιμές
- Για ευκολία
  - χρήση συνηθισμένου συμβολισμού `0, 1, 2...`
  - προσθέτουμε πράξεις `+ - * / < > =`
- Για ενδιαφέρον (πολυμορφισμός) προσθέτουμε λίστες:

$t ::=$	...		$v ::=$	...	
	`emptylist`	κενή λίστα		`emptylist`	
	`cons` t t	μη κενή λίστα		`cons` v v	
	`head` t	κεφαλή λίστας			
	`tail` t	ουρά λίστας			
	`isempty` t	έλ. κενότ.			

- Κανόνες αποτίμησης παραλείπονται

Εξαγωγή Τύπων

Σύστημα υπό Εξέταση - I

- Παίρνουμε τον λ-λογισμό χωρίς τύπους + φυσικούς + αληθοτιμές
- Για ευκολία
  - χρήση συνηθισμένου συμβολισμού `0, 1, 2...`
  - προσθέτουμε πράξεις `+ - * / < > =`
- Για ενδιαφέρον (πολυμορφισμός) προσθέτουμε λίστες:

$t ::=$	...	$v ::=$	...
`emptylist`	κενή λίστα	`emptylist`	
`cons` $t\ t$	μη κενή λίστα	`cons` $v\ v$	
`head` t	κεφαλή λίστας		
`tail` t	ουρά λίστας		
`isempty` t	έλ. κενότ.		

- Κανόνες αποτίμησης παραλείπονται

Εξαγωγή Τύπων

Σύστημα υπό Εξέταση - I

- Παίρνουμε τον λ-λογισμό χωρίς τύπους + φυσικούς + αληθοτιμές
- Για ευκολία
  - χρήση συνηθισμένου συμβολισμού `0, 1, 2...`
  - προσθέτουμε πράξεις `+ - * / < > =`
- Για ενδιαφέρον (πολυμορφισμός) προσθέτουμε λίστες:

$t ::=$	...		$v ::=$	...	
	`emptylist`	κενή λίστα		`emptylist`	
	`cons` $t\ t$	μη κενή λίστα		`cons` $v\ v$	
	`head` t	κεφαλή λίστας			
	`tail` t	ουρά λίστας			
	`isempty` t	έλ. κενότ.			

- Κανόνες αποτίμησης παραλείπονται

Εξαγωγή Τύπων

Σύστημα υπό Εξέταση - I

- Παίρνουμε τον λ-λογισμό χωρίς τύπους + φυσικούς + αληθοτιμές
- Για ευκολία
  - χρήση συνηθισμένου συμβολισμού `0, 1, 2...`
  - προσθέτουμε πράξεις `+ - * / < > =`
- Για ενδιαφέρον (πολυμορφισμός) προσθέτουμε λίστες:

$t ::=$	...		$v ::=$	...	
	`emptylist`	κενή λίστα		`emptylist`	
	`cons` $t\ t$	μη κενή λίστα		`cons` $v\ v$	
	`head` t	κεφαλή λίστας			
	`tail` t	ουρά λίστας			
	`isempty` t	έλ. κενότ.			

- Κανόνες αποτίμησης παραλείπονται

- Παίρνουμε τον λ-λογισμό χωρίς τύπους + φυσικούς + αληθοτιμές
- Για ευκολία
  - χρήση συνηθισμένου συμβολισμού `0, 1, 2...`
  - προσθέτουμε πράξεις `+ - * / < > =`
- Για ενδιαφέρον (πολυμορφισμός) προσθέτουμε λίστες:

$t ::=$	...		$v ::=$	...
	`emptylist`	κενή λίστα		`emptylist`
	`cons` t t	μη κενή λίστα		`cons` v v
	`head` t	κεφαλή λίστας		
	`tail` t	ουρά λίστας		
	`isempty` t	έλ. κενότ.		

- Κανόνες αποτίμησης παραλείπονται

- Παίρνουμε τον λ-λογισμό χωρίς τύπους + φυσικούς + αληθοτιμές
- Για ευκολία
  - χρήση συνηθισμένου συμβολισμού `0, 1, 2...`
  - προσθέτουμε πράξεις `+ - * / < > =`
- Για ενδιαφέρον (πολυμορφισμός) προσθέτουμε λίστες:

$t ::=$	...		$v ::=$	...
	`emptylist`	κενή λίστα		`emptylist`
	`cons` t t	μη κενή λίστα		`cons` v v
	`head` t	κεφαλή λίστας		
	`tail` t	ουρά λίστας		
	`isempty` t	έλ. κενότ.		

- Κανόνες αποτίμησης παραλείπονται

- Τύποι λιστών:

$$\frac{}{\Gamma \vdash \texttt{emptylist} : [T]}\,(\text{τ.κενής λ.})$$

$$\frac{\Gamma \vdash t_1 : T \qquad \Gamma \vdash t_2 : [T]}{\Gamma \vdash \texttt{cons}\ t_1\ t_2 : [T]}\,(\text{τ.μη κενής λ.})$$

$$\frac{\Gamma \vdash t : [T]}{\Gamma \vdash \texttt{head}\ t : T}\,(\text{τ.κεφαλής})$$

$$\frac{\Gamma \vdash t : [T]}{\Gamma \vdash \texttt{tail}\ t : [T]}\,(\text{τ.ουράς})$$

$$\frac{\Gamma \vdash t : [T]}{\Gamma \vdash \texttt{isempty}\ t : \texttt{Bool}}\,(\text{τ.ελ.κενότ.})$$

- Προσθέτουμε μεταβλητές τύπων
  - Σύνολο TypeExp εκφράσεων τύπων δίνεται από:

$$TX ::= X | \texttt{Nat} | \texttt{Bool} | TX \to TX | [TX]$$

    (X μεταβλητές τύπων)
  - Σύνολο μεταβλητών τύπου: TypeVar $= \{A, B...\}$
- Αντικατάσταση τύπων: συνάρτηση στο TypeVar $\to_f$ TypeExp
- Έστω έκφραση τύπων E και αντικατάσταση σ. Η έκφραση $\sigma(E)$ δίνεται από ταυτόχρονη αντικατάσταση:

$$X \notin \text{Dom}\sigma \Rightarrow \sigma(X) = X$$
$$\sigma(\texttt{Nat}) = \texttt{Nat}$$
$$\sigma(\texttt{Bool}) = \texttt{Bool}$$
$$\sigma(T_1 \to T_2) = \sigma(T_1) \to \sigma(T_2)$$
$$\sigma([T]) = [\sigma(T)]$$

- Προσθέτουμε μεταβλητές τύπων
  - Σύνολο TypeExp εκφράσεων τύπων δίνεται από:

$$TX ::= X | \texttt{Nat} | \texttt{Bool} | TX \to TX | [TX]$$

    (X μεταβλητές τύπων)
  - Σύνολο μεταβλητών τύπου: TypeVar $= \{A, B...\}$
- Αντικατάσταση τύπων: συνάρτηση στο TypeVar $\to_f$ TypeExp
- Έστω έκφραση τύπων E και αντικατάσταση σ. Η έκφραση $\sigma(E)$ δίνεται από ταυτόχρονη αντικατάσταση:

$$X \notin \text{Dom}\sigma \Rightarrow \sigma(X) = X$$
$$\sigma(\texttt{Nat}) = \texttt{Nat}$$
$$\sigma(\texttt{Bool}) = \texttt{Bool}$$
$$\sigma(T_1 \to T_2) = \sigma(T_1) \to \sigma(T_2)$$
$$\sigma([T]) = [\sigma(T)]$$

- Προσθέτουμε μεταβλητές τύπων
  - Σύνολο TypeExp εκφράσεων τύπων δίνεται από:

$$TX ::= X | \texttt{Nat} | \texttt{Bool} | TX \to TX | [TX]$$

    (X μεταβλητές τύπων)
  - Σύνολο μεταβλητών τύπου: TypeVar $= \{A, B...\}$
- Αντικατάσταση τύπων: συνάρτηση στο TypeVar $\to_f$ TypeExp
- Έστω έκφραση τύπων E και αντικατάσταση σ. Η έκφραση $\sigma(E)$ δίνεται από ταυτόχρονη αντικατάσταση:

$$\begin{aligned}
&X \notin \text{Dom}\sigma \;\Rightarrow\; \sigma(X) = X \\
&\sigma(\texttt{Nat}) = \texttt{Nat} \\
&\sigma(\texttt{Bool}) = \texttt{Bool} \\
&\sigma(T_1 \to T_2) = \sigma(T_1) \to \sigma(T_2) \\
&\sigma([T]) = [\sigma(T)]
\end{aligned}$$

- Έστω κλειστός όρος χωρίς τύπους t. Αν ο t παίρνει τύπο, τότε υπάρχει μία έκφραση τύπων P_t ώστε κάθε τύπος του t να είναι της μορφής $\sigma(P_t)$ για κάποια αντικατάσταση σ
- Η E ονομάζεται πιο γενικός πολυμορφικός τύπος για τον t
- Η διαδικασία εξαγωγής τύπου παίρνει έναν όρο t και επιστρέφει
  - τον πιο γενικό πολυμορφικό τύπο P_t ή
  - σφάλμα σε περίπτωση που ο t δεν είναι κλειστός ή ο P_t δεν υπάρχει

- Έστω κλειστός όρος χωρίς τύπους t. Αν ο t παίρνει τύπο, τότε υπάρχει μία έκφραση τύπων P_t ώστε κάθε τύπος του t να είναι της μορφής $\sigma(P_t)$ για κάποια αντικατάσταση σ
- Η E ονομάζεται <u>πιο γενικός πολυμορφικός τύπος</u> για τον t
- Η διαδικασία εξαγωγής τύπου παίρνει έναν όρο t και επιστρέφει
  - τον πιο γενικό πολυμορφικό τύπο P_t ή
  - σφάλμα σε περίπτωση που ο t δεν είναι κλειστός ή ο P_t δεν υπάρχει

- Έστω κλειστός όρος χωρίς τύπους t. Αν ο t παίρνει τύπο, τότε υπάρχει μία έκφραση τύπων P_t ώστε κάθε τύπος του t να είναι της μορφής $\sigma(P_t)$ για κάποια αντικατάσταση σ
- Η E ονομάζεται πιο γενικός πολυμορφικός τύπος για τον t
- Η διαδικασία εξαγωγής τύπου παίρνει έναν όρο t και επιστρέφει
  - τον πιο γενικό πολυμορφικό τύπο P_t ή
  - σφάλμα σε περίπτωση που ο t δεν είναι κλειστός ή ο P_t δεν υπάρχει

Εξαγωγή Τύπων

Βήμα I - Ανάθεση Τύπων

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

```
λf.cons emptylist (f(λx.x+1))
```

- `λf.cons emptylist (f(λx.x+1)) : A`
- `f : B`
- `cons emptylist (f(λx.x+1)) : C`
- `emptylist : D`
- `f(λx.x+1) : E`
- `f : F`
- `λx.x+1 : G`
- `x : H`
- `x+1 : I`
- `x : J`
- `1 : K`

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1)) : A$
- $f : B$
- $\text{cons emptylist } (f(\lambda x.x+1)) : C$
- $\text{emptylist} : D$
- $f(\lambda x.x+1) : E$
- $f : F$
- $\lambda x.x+1 : G$
- $x : H$
- $x+1 : I$
- $x : J$
- $1 : K$

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1)) : A$
- $f : B$
- $\text{cons emptylist } (f(\lambda x.x+1)) : C$
- $\text{emptylist} : D$
- $f(\lambda x.x+1) : E$
- $f : F$
- $\lambda x.x+1 : G$
- $x : H$
- $x+1 : I$
- $x : J$
- $1 : K$

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\texttt{cons emptylist (f(}\lambda x.\texttt{x+1))}$$

  - $\lambda f.\texttt{cons emptylist (f(}\lambda x.\texttt{x+1)) : A}$
  - $\texttt{f : B}$
  - $\texttt{cons emptylist (f(}\lambda x.\texttt{x+1)) : C}$
  - $\texttt{emptylist : D}$
  - $\texttt{f(}\lambda x.\texttt{x+1) : E}$
  - $\texttt{f : F}$
  - $\lambda x.\texttt{x+1 : G}$
  - $\texttt{x : H}$
  - $\texttt{x+1 : I}$
  - $\texttt{x : J}$
  - $\texttt{1 : K}$

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$: A
- f : B
- cons emptylist $(f(\lambda x.x+1))$: C
- emptylist : D
- $f(\lambda x.x+1)$: E
- f : F
- $\lambda x.x+1$: G
- x : H
- x+1 : I
- x : J
- 1 : K

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\texttt{cons emptylist (f(}\lambda x.\texttt{x+1))}$$

- $\lambda f.\texttt{cons emptylist (f(}\lambda x.\texttt{x+1))} : \texttt{A}$
- $\texttt{f} : \texttt{B}$
- $\texttt{cons emptylist (f(}\lambda x.\texttt{x+1))} : \texttt{C}$
- $\texttt{emptylist} : \texttt{D}$
- $\texttt{f(}\lambda x.\texttt{x+1)} : \texttt{E}$
- $\texttt{f} : \texttt{F}$
- $\lambda x.\texttt{x+1} : \texttt{G}$
- $\texttt{x} : \texttt{H}$
- $\texttt{x+1} : \texttt{I}$
- $\texttt{x} : \texttt{J}$
- $\texttt{1} : \texttt{K}$

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$: A
- f : B
- cons emptylist $(f(\lambda x.x+1))$: C
- emptylist : D
- $f(\lambda x.x+1)$: E
- f : F
- $\lambda x.x+1$: G
- x : H
- x+1 : I
- x : J
- 1 : K

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1)) : A$
- $f : B$
- $\text{cons emptylist } (f(\lambda x.x+1)) : C$
- $\text{emptylist} : D$
- $f(\lambda x.x+1) : E$
- $f : F$
- $\lambda x.x+1 : G$
- $x : H$
- $x+1 : I$
- $x : J$
- $1 : K$

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$: A
- f : B
- cons emptylist $(f(\lambda x.x+1))$: C
- emptylist : D
- $f(\lambda x.x+1)$: E
- f : F
- $\lambda x.x+1$: G
- x : H
- x+1 : I
- x : J
- 1 : K

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$: A
- f : B
- cons emptylist $(f(\lambda x.x+1))$: C
- emptylist : D
- $f(\lambda x.x+1)$: E
- f : F
- $\lambda x.x+1$: G
- x : H
- x+1 : I
- x : J
- 1 : K

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$: A
- f : B
- cons emptylist $(f(\lambda x.x+1))$: C
- emptylist : D
- $f(\lambda x.x+1)$: E
- f : F
- $\lambda x.x+1$: G
- x : H
- x+1 : I
- x : J
- 1 : K

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$: A
- f : B
- cons emptylist $(f(\lambda x.x+1))$: C
- emptylist : D
- $f(\lambda x.x+1)$: E
- f : F
- $\lambda x.x+1$: G
- x : H
- x+1 : I
- x : J
- 1 : K

- Για κάθε υποέκφραση και κάθε δεσμεύουσα μεταβλητή στον όρο μας, ορίζουμε μία μεταβλητή τύπου.
- Παράδειγμα:

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

- $\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$: A
- f : B
- cons emptylist $(f(\lambda x.x+1))$: C
- emptylist : D
- $f(\lambda x.x+1)$: E
- f : F
- $\lambda x.x+1$: G
- x : H
- x+1 : I
- x : J
- 1 : K

- Περιορισμός (constraint) = μία εξίσωση μεταξύ εκφράσεων τύπων
- Ο αλγόριθμος εξαγωγής τύπου εξάγει ένα σύνολο περιορισμών C που περιέχει τους τύπους των υποεκφράσεων
- Κανόνες εξαγωγής περιορισμών (για κάθε υποέκφραση E και αντίστοιχη μεταβλητή τύπου X)
  - μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα I:

$$E = x : X \Rightarrow X = Y$$

  - σταθερά c τύπου $T = \texttt{Bool}$ ή $T = \texttt{Nat}$:

$$E = c : X \Rightarrow X = T$$

  - δυαδικός τελεστής b τύπου $T \to T \to U$

$$\left\{ \begin{array}{l} E = b\ t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Y = T \\ Z = T \\ X = U \end{array} \right\}$$

- Περιορισμός (constraint) = μία εξίσωση μεταξύ εκφράσεων τύπων
- Ο αλγόριθμος εξαγωγής τύπου εξάγει ένα σύνολο περιορισμών C που περιέχει τους τύπους των υποεκφράσεων
- Κανόνες εξαγωγής περιορισμών (για κάθε υποέκφραση E και αντίστοιχη μεταβλητή τύπου X)
  - μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα I:

$$E = x : X \Rightarrow X = Y$$

  - σταθερά c τύπου $T = \texttt{Bool}$ ή $T = \texttt{Nat}$:

$$E = c : X \Rightarrow X = T$$

  - δυαδικός τελεστής b τύπου $T \to T \to U$

$$\left\{ \begin{array}{l} E = b\ t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Y = T \\ Z = T \\ X = U \end{array} \right\}$$

- Περιορισμός (constraint) = μία εξίσωση μεταξύ εκφράσεων τύπων
- Ο αλγόριθμος εξαγωγής τύπου εξάγει ένα σύνολο περιορισμών C που περιέχει τους τύπους των υποεκφράσεων
- Κανόνες εξαγωγής περιορισμών (για κάθε υποέκφραση E και αντίστοιχη μεταβλητή τύπου X)
  - μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα I:

$$E = x : X \quad \Rightarrow \quad X = Y$$

  - σταθερά c τύπου $T = \texttt{Bool}$ ή $T = \texttt{Nat}$:

$$E = c : X \quad \Rightarrow \quad X = T$$

  - δυαδικός τελεστής b τύπου $T \longrightarrow T \longrightarrow U$

$$\left\{ \begin{array}{l} E = b\ t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{c} Y = T \\ Z = T \\ X = U \end{array} \right\}$$

- Περιορισμός (constraint) = μία εξίσωση μεταξύ εκφράσεων τύπων
- Ο αλγόριθμος εξαγωγής τύπου εξάγει ένα σύνολο περιορισμών C που περιέχει τους τύπους των υποεκφράσεων
- Κανόνες εξαγωγής περιορισμών (για κάθε υποέκφραση E και αντίστοιχη μεταβλητή τύπου X)
  - μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα Ι:

$$E = x : X \quad \Rightarrow \quad X = Y$$

  - σταθερά c τύπου $T = \texttt{Bool}$ ή $T = \texttt{Nat}$:

$$E = c : X \quad \Rightarrow \quad X = T$$

  - δυαδικός τελεστής b τύπου $T \longrightarrow T \longrightarrow U$

$$\left\{ \begin{array}{l} E = b\ t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{c} Y = T \\ Z = T \\ X = U \end{array} \right\}$$

- Περιορισμός (constraint) = μία εξίσωση μεταξύ εκφράσεων τύπων
- Ο αλγόριθμος εξαγωγής τύπου εξάγει ένα σύνολο περιορισμών C που περιέχει τους τύπους των υποεκφράσεων
- Κανόνες εξαγωγής περιορισμών (για κάθε υποέκφραση E και αντίστοιχη μεταβλητή τύπου X)
  - μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα I:

$$E = x : X \quad \Rightarrow \quad X = Y$$

  - σταθερά c τύπου $T = \texttt{Bool}$ ή $T = \texttt{Nat}$:

$$E = c : X \quad \Rightarrow \quad X = T$$

  - δυαδικός τελεστής b τύπου $T \longrightarrow T \longrightarrow U$

$$\left\{ \begin{array}{l} E = b\ t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{c} Y = T \\ Z = T \\ X = U \end{array} \right\}$$

- Περιορισμός (constraint) = μία εξίσωση μεταξύ εκφράσεων τύπων
- Ο αλγόριθμος εξαγωγής τύπου εξάγει ένα σύνολο περιορισμών C που περιέχει τους τύπους των υποεκφράσεων
- Κανόνες εξαγωγής περιορισμών (για κάθε υποέκφραση E και αντίστοιχη μεταβλητή τύπου X)
  - μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα I:

$$E = x : X \quad \Rightarrow \quad X = Y$$

  - σταθερά c τύπου $T = \texttt{Bool}$ ή $T = \texttt{Nat}$:

$$E = c : X \quad \Rightarrow \quad X = T$$

  - δυαδικός τελεστής b τύπου $T \longrightarrow T \longrightarrow U$

$$\left\{ \begin{array}{l} E = b\ t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{c} Y = T \\ Z = T \\ X = U \end{array} \right\}$$

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής `if then else`:

$$\left\{ \begin{array}{l} E = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : X \\ t_1 : Y \\ t_2 : Z \\ t_3 : W \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Y = \texttt{Bool} \\ X = Z \\ Z = W \end{array} \right\}$$

  - Κενή λίστα:

$$E = \texttt{emptylist} : X \Rightarrow X = [Y]$$

    - Y: αχρησιμοποίητη μεταβλητή τύπων
  - Μη κενή λίστα:

$$\left\{ \begin{array}{l} E = \texttt{cons } t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Z = [Y] \\ X = Z \end{array} \right\}$$

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής `if then else`:

$$\left\{ \begin{array}{l} E = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : X \\ t_1 : Y \\ t_2 : Z \\ t_3 : W \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Y = \texttt{Bool} \\ X = Z \\ Z = W \end{array} \right\}$$

  - Κενή λίστα:

$$E = \texttt{emptylist} : X \Rightarrow X = [Y]$$

    - Y: αχρησιμοποίητη μεταβλητή τύπων
  - Μη κενή λίστα:

$$\left\{ \begin{array}{l} E = \texttt{cons } t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Z = [Y] \\ X = Z \end{array} \right\}$$

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής `if then else`:

$$\left\{ \begin{array}{l} E = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : X \\ t_1 : Y \\ t_2 : Z \\ t_3 : W \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Y = \texttt{Bool} \\ X = Z \\ Z = W \end{array} \right\}$$

  - Κενή λίστα:

$$E = \texttt{emptylist} : X \Rightarrow X = [Y]$$

    - Y: αχρησιμοποίητη μεταβλητή τύπων
  - Μη κενή λίστα:

$$\left\{ \begin{array}{l} E = \texttt{cons } t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Z = [Y] \\ X = Z \end{array} \right\}$$

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής `if then else`:

$$\left\{ \begin{array}{l} E = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : X \\ t_1 : Y \\ t_2 : Z \\ t_3 : W \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Y = \texttt{Bool} \\ X = Z \\ Z = W \end{array} \right\}$$

  - Κενή λίστα:

$$E = \texttt{emptylist} : X \Rightarrow X = [Y]$$

    - Y: αχρησιμοποίητη μεταβλητή τύπων
  - Μη κενή λίστα:

$$\left\{ \begin{array}{l} E = \texttt{cons } t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Z = [Y] \\ X = Z \end{array} \right\}$$

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής `head`:

$$\left\{ \begin{array}{l} E = \texttt{head}\ t : X \\ t : Y \end{array} \right\} \Rightarrow Y = [X]$$

  - τελεστής `tail`:

$$\left\{ \begin{array}{l} E = \texttt{tail}\ t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = Y \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων
  - τελεστής `isempty`:

$$\left\{ \begin{array}{l} E = \texttt{isempty}\ t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = \texttt{Bool} \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής head:

$$\left\{ \begin{array}{l} E = \text{head } t : X \\ t : Y \end{array} \right\} \Rightarrow Y = [X]$$

  - τελεστής tail:

$$\left\{ \begin{array}{l} E = \texttt{tail } t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = Y \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων
  - τελεστής isempty:

$$\left\{ \begin{array}{l} E = \texttt{isempty } t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = \texttt{Bool} \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής `head`:

$$\left\{ \begin{array}{l} E = \texttt{head}\ t : X \\ t : Y \end{array} \right\} \Rightarrow Y = [X]$$

  - τελεστής `tail`:

$$\left\{ \begin{array}{l} E = \texttt{tail}\ t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = Y \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων
  - τελεστής `isempty`:

$$\left\{ \begin{array}{l} E = \texttt{isempty}\ t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = \texttt{Bool} \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - τελεστής `head`:

$$\left\{ \begin{array}{l} E = \texttt{head}\ t : X \\ t : Y \end{array} \right\} \Rightarrow Y = [X]$$

  - τελεστής `tail`:

$$\left\{ \begin{array}{l} E = \texttt{tail}\ t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = Y \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων
  - τελεστής `isempty`:

$$\left\{ \begin{array}{l} E = \texttt{isempty}\ t : X \\ t : Y \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} X = \texttt{Bool} \\ Y = [Z] \end{array} \right\}$$

    - Z: αχρησιμοποίητη μεταβλητή τύπων

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - Εφαρμογή:

$$\left\{ \begin{array}{l} E = t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \quad Y = Z \rightarrow X$$

  - λ-αφαίρεση σε δεσμεύουσα μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα I:

$$\left\{ \begin{array}{l} E = \lambda x.t : X \\ t : Z \end{array} \right\} \Rightarrow \quad X = Y \rightarrow Z$$

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - Εφαρμογή:

$$\left\{ \begin{array}{l} E = t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \quad Y = Z \rightarrow X$$

  - λ-αφαίρεση σε δεσμεύουσα μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα Ι:

$$\left\{ \begin{array}{l} E = \lambda x.t : X \\ t : Z \end{array} \right\} \Rightarrow \quad X = Y \rightarrow Z$$

- Κανόνες εξαγωγής περιορισμών (συνέχεια)
  - Εφαρμογή:

$$\left\{ \begin{array}{l} E = t_1\ t_2 : X \\ t_1 : Y \\ t_2 : Z \end{array} \right\} \Rightarrow \quad Y = Z \rightarrow X$$

  - λ-αφαίρεση σε δεσμεύουσα μεταβλητή x. Έστω Y η μεταβλητή τύπου που ανέθεσε στη x το βήμα I:

$$\left\{ \begin{array}{l} E = \lambda x.t : X \\ t : Z \end{array} \right\} \Rightarrow \quad X = Y \rightarrow Z$$

$$\lambda f.\texttt{cons emptylist (f}(\lambda x.\texttt{x+1}))$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = \texttt{Nat} \\ J = \texttt{Nat} & K = \texttt{Nat} \\ J = H & K = \texttt{Nat} \end{array}$$

$$\lambda f.\texttt{cons emptylist } (f(\lambda x.x+1))$$

$$\lambda f.\texttt{cons emptylist } (f(\lambda x.x+1)) : \texttt{A}$$

$$\texttt{f} : \texttt{B}$$

$$\texttt{cons emptylist } (f(\lambda x.x+1)) : \texttt{C}$$

Περιορισμοί:

$$\texttt{A} = \texttt{B} \rightarrow \texttt{C} \quad \texttt{E} = [\texttt{D}]$$
$$\texttt{C} = \texttt{E} \quad \texttt{D} = [\texttt{L}]$$
$$\texttt{F} = \texttt{G} \rightarrow \texttt{E} \quad \texttt{F} = \texttt{B}$$
$$\texttt{G} = \texttt{H} \rightarrow \texttt{I} \quad \texttt{I} = \texttt{Nat}$$
$$\texttt{J} = \texttt{Nat} \quad \texttt{K} = \texttt{Nat}$$
$$\texttt{J} = \texttt{H} \quad \texttt{K} = \texttt{Nat}$$

λf.cons emptylist (f(λx.x+1))

cons emptylist (f(λx.x+1)) : C

emptylist : D

f(λx.x+1) : E

Περιορισμοί:

A = B$\rightarrow$C E = [D]
C = E D = [L]
F = G$\rightarrow$E F = B
G = H$\rightarrow$I I = Nat
J = Nat K = Nat
J = H K = Nat

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

$$\text{emptylist} : \quad D$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = \text{Nat} \\ J = \text{Nat} & K = \text{Nat} \\ J = H & K = \text{Nat} \end{array}$$

$$\lambda f.\mathtt{cons\ emptylist}\ (f(\lambda x.x{+}1))$$

$$f(\lambda x.x{+}1) : E$$

$$f : F$$

$$\lambda x.x{+}1 : G$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = Nat \\ J = Nat & K = Nat \\ J = H & K = Nat \end{array}$$

$$\lambda f.\texttt{cons emptylist } (f(\lambda x.x+1))$$

$$f : F$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = Nat \\ J = Nat & K = Nat \\ J = H & K = Nat \end{array}$$

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

$$\lambda x.x+1 : G$$

$$x : H$$

$$x+1 : I$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = \text{Nat} \\ J = \text{Nat} & K = \text{Nat} \\ J = H & K = \text{Nat} \end{array}$$

$$\lambda f.\text{cons emptylist } (f(\lambda x.x+1))$$

$$x+1 : I$$

$$x : J$$

$$1 : K$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = \text{Nat} \\ J = \text{Nat} & K = \text{Nat} \\ J = H & K = \text{Nat} \end{array}$$

$$\lambda f.\text{cons emptylist } (f(\lambda x.x{+}1))$$

$$x : J$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = Nat \\ J = Nat & K = Nat \\ J = H & K = Nat \end{array}$$

$$\lambda f.\texttt{cons emptylist (f}(\lambda x.x+1))$$

$$1 : K$$

Περιορισμοί:

$$\begin{array}{ll} A = B \to C & E = [D] \\ C = E & D = [L] \\ F = G \to E & F = B \\ G = H \to I & I = Nat \\ J = Nat & K = Nat \\ J = H & K = Nat \end{array}$$

- Παίρνουμε το σύνολο περιορισμών C από το προηγούμενο βήμα και εφαρμόζουμε ενοποίηση (unification) για την εξαγωγή του πιο γενικού τύπου
- Αλγόριθμος ενοποίησης:
  - εισάγουμε νέο περιορισμό $A' = A$. Έστω a αυτός ο περιορισμός
  - για κάθε περιορισμό c (εκτός από τον a)
    - αν ο c βρίσκει αντίφαση, σταματάμε με αποτυχία
    - αλλιώς, εφαρμόζουμε τον c σε όλους τους άλλους περιορισμούς
    - βγάζουμε τον c από τον C
  - αν δεν έχουμε αποτυχία μέχρι το τέλος, ο περιορισμός a δίνει τον πιο γενικό τύπο της έκφρασης

- Παίρνουμε το σύνολο περιορισμών C από το προηγούμενο βήμα και εφαρμόζουμε ενοποίηση (unification) για την εξαγωγή του πιο γενικού τύπου
- Αλγόριθμος ενοποίησης:
  - εισάγουμε νέο περιορισμό $A' = A$. Έστω a αυτός ο περιορισμός
  - για κάθε περιορισμό c (εκτός από τον a)
    - αν ο c βρίσκει αντίφαση, σταματάμε με αποτυχία
    - αλλιώς, εφαρμόζουμε τον c σε όλους τους άλλους περιορισμούς
    - βγάζουμε τον c από τον C
  - αν δεν έχουμε αποτυχία μέχρι το τέλος, ο περιορισμός a δίνει τον πιο γενικό τύπο της έκφρασης

- Παίρνουμε το σύνολο περιορισμών C από το προηγούμενο βήμα και εφαρμόζουμε ενοποίηση (unification) για την εξαγωγή του πιο γενικού τύπου
- Αλγόριθμος ενοποίησης:
  - εισάγουμε νέο περιορισμό $A' = A$. Έστω a αυτός ο περιορισμός
  - για κάθε περιορισμό c (εκτός από τον a)
    - αν ο c βρίσκει αντίφαση, σταματάμε με αποτυχία
    - αλλιώς, εφαρμόζουμε τον c σε όλους τους άλλους περιορισμούς
    - βγάζουμε τον c από τον C
  - αν δεν έχουμε αποτυχία μέχρι το τέλος, ο περιορισμός a δίνει τον πιο γενικό τύπο της έκφρασης

- Παίρνουμε το σύνολο περιορισμών C από το προηγούμενο βήμα και εφαρμόζουμε ενοποίηση (unification) για την εξαγωγή του πιο γενικού τύπου
- Αλγόριθμος ενοποίησης:
  - εισάγουμε νέο περιορισμό $A' = A$. Έστω a αυτός ο περιορισμός
  - για κάθε περιορισμό c (εκτός από τον a)
    - αν ο c βρίσκει αντίφαση, σταματάμε με αποτυχία
    - αλλιώς, εφαρμόζουμε τον c σε όλους τους άλλους περιορισμούς
    - βγάζουμε τον c από τον C
  - αν δεν έχουμε αποτυχία μέχρι το τέλος, ο περιορισμός a δίνει τον πιο γενικό τύπο της έκφρασης

- Παίρνουμε το σύνολο περιορισμών C από το προηγούμενο βήμα και εφαρμόζουμε ενοποίηση (unification) για την εξαγωγή του πιο γενικού τύπου
- Αλγόριθμος ενοποίησης:
  - εισάγουμε νέο περιορισμό $A' = A$. Έστω a αυτός ο περιορισμός
  - για κάθε περιορισμό c (εκτός από τον a)
    - αν ο c βρίσκει αντίφαση, σταματάμε με αποτυχία
    - αλλιώς, εφαρμόζουμε τον c σε όλους τους άλλους περιορισμούς
    - βγάζουμε τον c από τον C
  - αν δεν έχουμε αποτυχία μέχρι το τέλος, ο περιορισμός a δίνει τον πιο γενικό τύπο της έκφρασης

- Παίρνουμε το σύνολο περιορισμών C από το προηγούμενο βήμα και εφαρμόζουμε ενοποίηση (unification) για την εξαγωγή του πιο γενικού τύπου
- Αλγόριθμος ενοποίησης:
  - εισάγουμε νέο περιορισμό $A' = A$. Έστω a αυτός ο περιορισμός
  - για κάθε περιορισμό c (εκτός από τον a)
    - αν ο c βρίσκει αντίφαση, σταματάμε με αποτυχία
    - αλλιώς, εφαρμόζουμε τον c σε όλους τους άλλους περιορισμούς
    - βγάζουμε τον c από τον C
  - αν δεν έχουμε αποτυχία μέχρι το τέλος, ο περιορισμός a δίνει τον πιο γενικό τύπο της έκφρασης

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

- Κανόνες χειρισμού περιορισμών:
  - περιορισμός $X = T$ και $X \notin FV(T)$: εφαρμογή της μετατροπής $[X := T]$ σε κάθε άλλο περιορισμό του C
  - περιορισμός $T = X$: όμοια με $X = T$
  - περιορισμός $[T_1] = [T_2]$: εισαγωγή περιορισμού $T_1 = T_2$
  - περιορισμός $S_1 \to T_1 = S_2 \to T_2$: εισαγωγή περιορισμών $S_1 = S_2$ και $T_1 = T_2$
  - περιορισμός $T = T$ για T σταθερό τύπο ή μεταβλητή: καμία ενέργεια
  - κάθε άλλη περίπτωση: αποτυχία
- Στο τέλος μένουμε με τον περιορισμό a που είναι της μορφής $A' = T$. Ο T είναι ο πιο γενικός τύπος που ψάχνουμε

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
J = Nat	K = Nat
J = H	K = Nat

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
J = Nat	K = Nat
J = H	K = Nat

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
J = Nat	Nat = Nat
J = H	

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
J = Nat	Nat = Nat
J = H	

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
H = Nat	Nat = Nat

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
H = Nat	Nat = Nat

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
H = Nat	

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = H$\rightarrow$I	I = Nat
H = Nat	

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = Nat$\rightarrow$I	I = Nat

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = Nat$\rightarrow$I	I = Nat

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = Nat$\rightarrow$Nat	

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = G$\rightarrow$E	F = B
G = Nat$\rightarrow$Nat	

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = (Nat$\rightarrow$Nat)$\rightarrow$E	F = B

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C	E = [D]
C = E	D = [L]
F = (Nat$\rightarrow$Nat)$\rightarrow$E	F = B

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C
C = E
B = (Nat$\rightarrow$Nat)$\rightarrow$E

E = [D]
D = [L]

λf.cons emptylist (f(λx.x+1))

A = B$\rightarrow$C
C = E
B = (Nat$\rightarrow$Nat)$\rightarrow$E

E = [D]
D = [L]

λf.cons emptylist (f(λx.x+1))

A = ((Nat$\rightarrow$Nat)$\rightarrow$E)$\rightarrow$C E = [D]
C = E D = [L]

λf.cons emptylist (f(λx.x+1))

A = ((Nat$\rightarrow$Nat)$\rightarrow$E)$\rightarrow$C E = [D]
C = E D = [L]

λf.cons emptylist (f(λx.x+1))

A = ((Nat$\rightarrow$Nat)$\rightarrow$E)$\rightarrow$C E = [[L]]
C = E

λf.cons emptylist (f(λx.x+1))

A = ((Nat$\rightarrow$Nat)$\rightarrow$E)$\rightarrow$C E = [[L]]

C = E

λf.cons emptylist (f(λx.x+1))

A = ((Nat$\rightarrow$Nat)$\rightarrow$E)$\rightarrow$E E = [[L]]

$$\lambda f.\mathtt{cons}\ \mathtt{emptylist}\ (f(\lambda x.x+1))$$

$$A = ((\mathtt{Nat}\to\mathtt{Nat})\to E)\to E \qquad E = [[\bot]]$$

$$\lambda f.\texttt{cons emptylist } (f(\lambda x.x+1))$$

$$A = ((\texttt{Nat} \to \texttt{Nat}) \to [[L]]) \to [[L]]$$

- Δομές `let,where` και ορισμοί (+αναδρομή)

```
([]++"a",[]++[1])
\l->(l++"a",l++[1])
```

- Πλειάδες, αλγεβρικοί τύποι
  - προφανής επέκταση
- Υπερφόρτωση και κλάσεις
- Μοτίβα

- Δομές `let,where` και ορισμοί (+αναδρομή)

```
([]++"a",[]++[1])
\l->(l++"a",l++[1])
```

- Πλειάδες, αλγεβρικοί τύποι
  - προφανής επέκταση
- Υπερφόρτωση και κλάσεις
- Μοτίβα

- Δομές `let,where` και ορισμοί (+αναδρομή)

```
([]++"a",[]++[1])
\l->(l++"a",l++[1])
```

- Πλειάδες, αλγεβρικοί τύποι
  - προφανής επέκταση
- Υπερφόρτωση και κλάσεις
- Μοτίβα

- Δομές `let,where` και ορισμοί (+αναδρομή)

```
([]++"a",[]++[1])
\l->(l++"a",l++[1])
```

- Πλειάδες, αλγεβρικοί τύποι
  - προφανής επέκταση
- Υπερφόρτωση και κλάσεις
- Μοτίβα

- Δομές `let,where` και ορισμοί (+αναδρομή)

```
([]++"a",[]++[1])
\l->(l++"a",l++[1])
```

- Πλειάδες, αλγεβρικοί τύποι
  - προφανής επέκταση
- Υπερφόρτωση και κλάσεις
- Μοτίβα

- Επιτρέπονται εκφράσεις χωρίς τύπους
  - (ή τύποι με μεταβλητές τύπων)
- Εκφράσεις μπορούν να χρησιμοποιηθούν με οποιονδήποτε από τους τύπους τους:

```
map ::  (a->b)->[a]->[b]
```

```
map (\x-> x+1) [2,3] ⟶ [3,4]
```

```
map (\x-> x++[1]) [[2],[3]] ⟶ [[2,1],[3,1]]
```

- Το σύστημα τύπων δεν επηρεάζεται. Εκφράσεις όπως `[1,False]` δεν παίρνουν τύπο και δεν επιτρέπονται
- Ένας πολυμορφικός τύπος είναι ένα σύνολο μονομορφικών τύπων
  - οι μεταβλητές αντικαθίστανται μόνο από μονομορφικούς τύπους

- Επιτρέπονται εκφράσεις χωρίς τύπους
  - (ή τύποι με μεταβλητές τύπων)
- Εκφράσεις μπορούν να χρησιμοποιηθούν με οποιονδήποτε από τους τύπους τους:

```
map ::  (a->b)->[a]->[b]
```

```
map (\x-> x+1) [2,3] ⟶ [3,4]
```

```
map (\x-> x++[1]) [[2],[3]] ⟶ [[2,1],[3,1]]
```

- Το σύστημα τύπων δεν επηρεάζεται. Εκφράσεις όπως `[1,False]` δεν παίρνουν τύπο και δεν επιτρέπονται
- Ένας πολυμορφικός τύπος είναι ένα σύνολο μονομορφικών τύπων
  - οι μεταβλητές αντικαθίστανται μόνο από μονομορφικούς τύπους

- Επιτρέπονται εκφράσεις χωρίς τύπους
  - (ή τύποι με μεταβλητές τύπων)
- Εκφράσεις μπορούν να χρησιμοποιηθούν με οποιονδήποτε από τους τύπους τους:

```
map ::  (a->b)->[a]->[b]
```

```
map (\x-> x+1) [2,3] ⟶ [3,4]
```

```
map (\x-> x++[1]) [[2],[3]] ⟶ [[2,1],[3,1]]
```

- Το σύστημα τύπων δεν επηρεάζεται. Εκφράσεις όπως `[1,False]` δεν παίρνουν τύπο και δεν επιτρέπονται
- Ένας πολυμορφικός τύπος είναι ένα σύνολο μονομορφικών τύπων
  - οι μεταβλητές αντικαθίστανται μόνο από μονομορφικούς τύπους

- Επιτρέπονται εκφράσεις χωρίς τύπους
  - (ή τύποι με μεταβλητές τύπων)
- Εκφράσεις μπορούν να χρησιμοποιηθούν με οποιονδήποτε από τους τύπους τους:

```
map ::  (a->b)->[a]->[b]
```

```
map (\x-> x+1) [2,3] ⟶ [3,4]
```

```
map (\x-> x++[1]) [[2],[3]] ⟶ [[2,1],[3,1]]
```

- Το σύστημα τύπων δεν επηρεάζεται. Εκφράσεις όπως `[1,False]` δεν παίρνουν τύπο και δεν επιτρέπονται
- Ένας πολυμορφικός τύπος είναι ένα σύνολο μονομορφικών τύπων
  - οι μεταβλητές αντικαθίστανται μόνο από μονομορφικούς τύπους

- Σύστημα *F*: οι πολυμορφικοί τύποι συμπεριλαμβάνονται στο σύστημα τύπων
- Ένας πολυμορφικός τύπος αντιμετωπίζεται ως ένας τύπος από μόνος του και όχι ως σύνολο μονομορφικών τύπων
- Οι μεταβλητές τύπων αντικαθίστανται και από πολυμορφικούς τύπους
- Παράδειγμα πολυμορφικού τύπου: $\forall X.[X]$
- αν $e:\forall X.[X]$, τότε, για κάθε τύπο X είναι $e\langle\langle X\rangle\rangle:[X]$
- παράδειγμα:

```
emptylist :  ∀X.[X]
emptylist<<Int>> :  [Int]
emptylist<<∀X.X→X>> :  [∀X.X→X]
[emptylist] :  [∀X.[X]]
```

- Εξαγωγή τύπου μη αποφασίσιμη

- Σύστημα *F*: οι πολυμορφικοί τύποι συμπεριλαμβάνονται στο σύστημα τύπων
- Ένας πολυμορφικός τύπος αντιμετωπίζεται ως ένας τύπος από μόνος του και όχι ως σύνολο μονομορφικών τύπων
- Οι μεταβλητές τύπων αντικαθίστανται και από πολυμορφικούς τύπους
- Παράδειγμα πολυμορφικού τύπου: $\forall X.[X]$
- αν $e:\forall X.[X]$, τότε, για κάθε τύπο X είναι $e\langle\langle X\rangle\rangle:[X]$
- παράδειγμα:

```
emptylist :  ∀X.[X]
emptylist<<Int>> :  [Int]
emptylist<<∀X.X→X>> :  [∀X.X→X]
[emptylist] :  [∀X.[X]]
```

- Εξαγωγή τύπου μη αποφασίσιμη

- Σύστημα *F*: οι πολυμορφικοί τύποι συμπεριλαμβάνονται στο σύστημα τύπων
- Ένας πολυμορφικός τύπος αντιμετωπίζεται ως ένας τύπος από μόνος του και όχι ως σύνολο μονομορφικών τύπων
- Οι μεταβλητές τύπων αντικαθίστανται και από πολυμορφικούς τύπους
- Παράδειγμα πολυμορφικού τύπου: $\forall X.[X]$
- αν $e:\forall X.[X]$, τότε, για κάθε τύπο X είναι $e\langle\langle X\rangle\rangle:[X]$
- παράδειγμα:

```
emptylist : ∀X.[X]
emptylist<<Int>> : [Int]
emptylist<<∀X.X→X>> : [∀X.X→X]
[emptylist] : [∀X.[X]]
```

- Εξαγωγή τύπου μη αποφασίσιμη

- Σύστημα *F*: οι πολυμορφικοί τύποι συμπεριλαμβάνονται στο σύστημα τύπων
- Ένας πολυμορφικός τύπος αντιμετωπίζεται ως ένας τύπος από μόνος του και όχι ως σύνολο μονομορφικών τύπων
- Οι μεταβλητές τύπων αντικαθίστανται και από πολυμορφικούς τύπους
- Παράδειγμα πολυμορφικού τύπου: $\forall X.[X]$
- αν $e:\forall X.[X]$, τότε, για κάθε τύπο X είναι $e\langle\langle X\rangle\rangle:[X]$
- παράδειγμα:

```
emptylist : ∀X.[X]
emptylist<<Int>> : [Int]
emptylist<<∀X.X→X>> : [∀X.X→X]
[emptylist] : [∀X.[X]]
```

- Εξαγωγή τύπου μη αποφασίσιμη

- Σύστημα *F*: οι πολυμορφικοί τύποι συμπεριλαμβάνονται στο σύστημα τύπων
- Ένας πολυμορφικός τύπος αντιμετωπίζεται ως ένας τύπος από μόνος του και όχι ως σύνολο μονομορφικών τύπων
- Οι μεταβλητές τύπων αντικαθίστανται και από πολυμορφικούς τύπους
- Παράδειγμα πολυμορφικού τύπου: $\forall X.[X]$
- αν $e : \forall X.[X]$, τότε, για κάθε τύπο X είναι $e\langle\langle X\rangle\rangle : [X]$
- παράδειγμα:

```
emptylist :  ∀X.[X]
emptylist<<Int>> :  [Int]
emptylist<<∀X.X→X>> :  [∀X.X→X]
[emptylist] :  [∀X.[X]]
```

- Εξαγωγή τύπου μη αποφασίσιμη

- Σύστημα *F*: οι πολυμορφικοί τύποι συμπεριλαμβάνονται στο σύστημα τύπων
- Ένας πολυμορφικός τύπος αντιμετωπίζεται ως ένας τύπος από μόνος του και όχι ως σύνολο μονομορφικών τύπων
- Οι μεταβλητές τύπων αντικαθίστανται και από πολυμορφικούς τύπους
- Παράδειγμα πολυμορφικού τύπου: $\forall X.[X]$
- αν $e:\forall X.[X]$, τότε, για κάθε τύπο X είναι $e\langle\langle X\rangle\rangle:[X]$
- παράδειγμα:

```
emptylist :  ∀X.[X]
emptylist<<Int>> :  [Int]
emptylist<<∀X.X→X>> :  [∀X.X→X]
[emptylist] :  [∀X.[X]]
```

- Εξαγωγή τύπου μη αποφασίσιμη

- Σύστημα *F*: οι πολυμορφικοί τύποι συμπεριλαμβάνονται στο σύστημα τύπων
- Ένας πολυμορφικός τύπος αντιμετωπίζεται ως ένας τύπος από μόνος του και όχι ως σύνολο μονομορφικών τύπων
- Οι μεταβλητές τύπων αντικαθίστανται και από πολυμορφικούς τύπους
- Παράδειγμα πολυμορφικού τύπου: $\forall X.[X]$
- αν $e:\forall X.[X]$, τότε, για κάθε τύπο X είναι $e\langle\langle X\rangle\rangle:[X]$
- παράδειγμα:

```
emptylist :  ∀X.[X]
emptylist<<Int>> :  [Int]
emptylist<<∀X.X→X>> :  [∀X.X→X]
[emptylist] :  [∀X.[X]]
```

- Εξαγωγή τύπου μη αποφασίσιμη

- Σύστημα $F_{<:}$: F + subtyping + subtyping στις παραμέτρους τύπων
- Περιορισμός της παραμέτρου σε υποτύπους δεδομένου τύπου: $\forall X<:\{c:Int\}.X \rightarrow X$
- Πανίσχυρο αλλά δύσχρηστο σύστημα
- Έλεγχος τύπων μη αποφασίσιμος στο πλήρες $F_{<:}$

- Σύστημα $F_{<:}$: F + subtyping + subtyping στις παραμέτρους τύπων
- Περιορισμός της παραμέτρου σε υποτύπους δεδομένου τύπου: $\forall X<:\{c:Int\}.X \rightarrow X$
- Πανίσχυρο αλλά δύσχρηστο σύστημα
- Έλεγχος τύπων μη αποφασίσιμος στο πλήρες $F_{<:}$

- Σύστημα $F_{<:}$: F + subtyping + subtyping στις παραμέτρους τύπων
- Περιορισμός της παραμέτρου σε υποτύπους δεδομένου τύπου: $\forall$X<:{c:Int}.X$\rightarrow$X
- Πανίσχυρο αλλά δύσχρηστο σύστημα
- Έλεγχος τύπων μη αποφασίσιμος στο πλήρες $F_{<:}$

- Σύστημα $F_{<:}$: F + subtyping + subtyping στις παραμέτρους τύπων
- Περιορισμός της παραμέτρου σε υποτύπους δεδομένου τύπου: $\forall$X<:{c:Int}.X$\rightarrow$X
- Πανίσχυρο αλλά δύσχρηστο σύστημα
- Έλεγχος τύπων μη αποφασίσιμος στο πλήρες $F_{<:}$

Επισκόπηση

- Οι αναδρομικοί τύποι έχουν πολύ μεγάλη εκφραστικότητα. Μπορεί να δώσει τύπο σε κάθε έκφραση του λ-λογισμού χωρίς τύπους
- Η χρήση μεταβλητών τύπων κάνει δυνατή την εξαγωγή τύπου και κάθε είδος πολυμορφισμού
- Σε ένα σύστημα τύπων όπως της Haskell, οι εκφράσεις έχουν ένα γενικότερο πολυμορφικό τύπο
- Ο αλγόριθμος εξαγωγής τύπου Hindley-Milner ανακαλύπτει τον πιο γενικό πολυμορφικό τύπο μίας έκφρασης χρησιμοποιώντας εξαγωγή περιορισμών και ενοποίηση
- Ο let- ή ML- πολυμορφισμός επιτρέπει τη χρήση εκφράσεων οπουδήποτε τουλάχιστον ένας από τους τύπους τους είναι έγκυρος
- Ισχυρότερα συστήματα πολυμορφισμού δε διαχωρίζουν τους πολυμορφικούς από τους μονομορφικούς τύπους

Επισκόπηση

- Οι αναδρομικοί τύποι έχουν πολύ μεγάλη εκφραστικότητα. Μπορεί να δώσει τύπο σε κάθε έκφραση του λ-λογισμού χωρίς τύπους
- Η χρήση μεταβλητών τύπων κάνει δυνατή την εξαγωγή τύπου και κάθε είδος πολυμορφισμού
- Σε ένα σύστημα τύπων όπως της Haskell, οι εκφράσεις έχουν ένα γενικότερο πολυμορφικό τύπο
- Ο αλγόριθμος εξαγωγής τύπου Hindley-Milner ανακαλύπτει τον πιο γενικό πολυμορφικό τύπο μίας έκφρασης χρησιμοποιώντας εξαγωγή περιορισμών και ενοποίηση
- Ο let- ή ML- πολυμορφισμός επιτρέπει τη χρήση εκφράσεων οπουδήποτε τουλάχιστον ένας από τους τύπους τους είναι έγκυρος
- Ισχυρότερα συστήματα πολυμορφισμού δε διαχωρίζουν τους πολυμορφικούς από τους μονομορφικούς τύπους

Επισκόπηση

- Οι αναδρομικοί τύποι έχουν πολύ μεγάλη εκφραστικότητα. Μπορεί να δώσει τύπο σε κάθε έκφραση του λ-λογισμού χωρίς τύπους
- Η χρήση μεταβλητών τύπων κάνει δυνατή την εξαγωγή τύπου και κάθε είδος πολυμορφισμού
- Σε ένα σύστημα τύπων όπως της Haskell, οι εκφράσεις έχουν ένα γενικότερο πολυμορφικό τύπο
- Ο αλγόριθμος εξαγωγής τύπου Hindley-Milner ανακαλύπτει τον πιο γενικό πολυμορφικό τύπο μίας έκφρασης χρησιμοποιώντας εξαγωγή περιορισμών και ενοποίηση
- Ο let- ή ML- πολυμορφισμός επιτρέπει τη χρήση εκφράσεων οπουδήποτε τουλάχιστον ένας από τους τύπους τους είναι έγκυρος
- Ισχυρότερα συστήματα πολυμορφισμού δε διαχωρίζουν τους πολυμορφικούς από τους μονομορφικούς τύπους

Επισκόπηση

- Οι αναδρομικοί τύποι έχουν πολύ μεγάλη εκφραστικότητα. Μπορεί να δώσει τύπο σε κάθε έκφραση του λ-λογισμού χωρίς τύπους
- Η χρήση μεταβλητών τύπων κάνει δυνατή την εξαγωγή τύπου και κάθε είδος πολυμορφισμού
- Σε ένα σύστημα τύπων όπως της Haskell, οι εκφράσεις έχουν ένα γενικότερο πολυμορφικό τύπο
- Ο αλγόριθμος εξαγωγής τύπου Hindley-Milner ανακαλύπτει τον πιο γενικό πολυμορφικό τύπο μίας έκφρασης χρησιμοποιώντας εξαγωγή περιορισμών και ενοποίηση
- Ο let- ή ML- πολυμορφισμός επιτρέπει τη χρήση εκφράσεων οπουδήποτε τουλάχιστον ένας από τους τύπους τους είναι έγκυρος
- Ισχυρότερα συστήματα πολυμορφισμού δε διαχωρίζουν τους πολυμορφικούς από τους μονομορφικούς τύπους

Επισκόπηση

- Οι αναδρομικοί τύποι έχουν πολύ μεγάλη εκφραστικότητα. Μπορεί να δώσει τύπο σε κάθε έκφραση του λ-λογισμού χωρίς τύπους
- Η χρήση μεταβλητών τύπων κάνει δυνατή την εξαγωγή τύπου και κάθε είδος πολυμορφισμού
- Σε ένα σύστημα τύπων όπως της Haskell, οι εκφράσεις έχουν ένα γενικότερο πολυμορφικό τύπο
- Ο αλγόριθμος εξαγωγής τύπου Hindley-Milner ανακαλύπτει τον πιο γενικό πολυμορφικό τύπο μίας έκφρασης χρησιμοποιώντας εξαγωγή περιορισμών και ενοποίηση
- Ο let- ή ML- πολυμορφισμός επιτρέπει τη χρήση εκφράσεων οπουδήποτε τουλάχιστον ένας από τους τύπους τους είναι έγκυρος
- Ισχυρότερα συστήματα πολυμορφισμού δε διαχωρίζουν τους πολυμορφικούς από τους μονομορφικούς τύπους

Επισκόπηση

- Οι αναδρομικοί τύποι έχουν πολύ μεγάλη εκφραστικότητα. Μπορεί να δώσει τύπο σε κάθε έκφραση του λ-λογισμού χωρίς τύπους
- Η χρήση μεταβλητών τύπων κάνει δυνατή την εξαγωγή τύπου και κάθε είδος πολυμορφισμού
- Σε ένα σύστημα τύπων όπως της Haskell, οι εκφράσεις έχουν ένα γενικότερο πολυμορφικό τύπο
- Ο αλγόριθμος εξαγωγής τύπου Hindley-Milner ανακαλύπτει τον πιο γενικό πολυμορφικό τύπο μίας έκφρασης χρησιμοποιώντας εξαγωγή περιορισμών και ενοποίηση
- Ο let- ή ML- πολυμορφισμός επιτρέπει τη χρήση εκφράσεων οπουδήποτε τουλάχιστον ένας από τους τύπους τους είναι έγκυρος
- Ισχυρότερα συστήματα πολυμορφισμού δε διαχωρίζουν τους πολυμορφικούς από τους μονομορφικούς τύπους