SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" Università di Roma
Ingegneria dell'Informazione,
Informatica e Statistica
Dipartimento di Informatica

Linguaggi di Programmazione

Appunti integrati con il libro "TODO", TODO 1, Autore 2, ...

Author
Alessio Bandiera

11 ottobre 2023

Indice

Informazioni e Contatti

Prerequisiti consigliati:

- Algebra
- TODO

Segnalazione errori ed eventuali migliorie:

Per segnalare eventuali errori e/o migliorie possibili, si prega di utilizzare il **sistema di Issues fornito da GitHub** all'interno della pagina della repository stessa contenente questi ed altri appunti (link fornito al di sotto), utilizzando uno dei template già forniti compilando direttamente i campi richiesti.

Gli appunti sono in continuo aggiornamento, pertanto, previa segnalazione, si prega di controllare se l'errore sia ancora presente nella versione più recente.

Licenza di distribuzione:

Contatti dell'autore e ulteriori link:

- Github: **https://github.com/ph04**
- Email: **alessio.bandiera02@gmail.com**
- LinkedIn: **Alessio Bandiera**

1 Induzione

1.1 Algebre induttive

1.1.1 Assiomi di Peano

Definizione 1.1.1.1: Assiomi di Peano

Gli **assiomi di Peano** sono 5 assiomi che definiscono l'insieme $\mathbb{N}$, e sono i seguenti:

i) $0 \in \mathbb{N}$

ii) $\exists \text{succ} : \mathbb{N} \to \mathbb{N}$, o equivalentemente, $\forall x \in \mathbb{N} \quad \text{succ}(x) \in \mathbb{N}$

iii) $\forall x, y \in \mathbb{N} \quad x \neq y \implies \text{succ}(x) \neq \text{succ}(y)$

iv) $\nexists x \in \mathbb{N} \mid \text{succ}(x) = 0$

v) $\forall S \subseteq \mathbb{N} \quad (0 \in S \land (\forall x \in S \quad \text{succ}(x) \in S)) \implies S = \mathbb{N}$

Esempio 1.1.1.1 ($\mathbb{N}$ di von Neumann)**.** Una rappresentazione dell'insieme dei numeri naturali $\mathbb{N}$ alternativa alla canonica

$$\mathbb{N} := \{0, 1, 2, \ldots\}$$

è stata fornita da John von Neumann. Indicando tale rappresentazione con $\aleph$, si ha che, per Neumann

$$\begin{gathered} 0_\aleph := \varnothing = \{\} \\ 1_\aleph := \{0_\aleph\} = \{\{\}\} \\ 2_\aleph := \{0_\aleph, 1_\aleph\} = \{\{\}, \{\{\}\}\} \\ \vdots \end{gathered}$$

e la funzione $\text{succ}_\aleph$ è definita come segue

$$\text{succ}_\aleph : \aleph \to \aleph : x_\aleph \mapsto x_\aleph \cup \{x_\aleph\} = \{\mu_\aleph \in \aleph \mid |\mu_\aleph| \leq |x_\aleph|\}$$

ed in particolare $\forall x_{\aleph} \in \aleph \quad |x_{\aleph}| + 1 = |\text{succ}_{\aleph}(x_{\aleph})|$.

È possibile verificare che tale rappresentazione di $\mathbb{N}$ soddisfa gli assiomi di Peano, in quanto

i) $0_{\aleph} := \varnothing \in \aleph$;

ii) $\exists \text{succ}_{\aleph} : \aleph \to \aleph$, definita precedentemente;

iii) $\forall x_{\aleph}, y_{\aleph} \in \aleph \quad x_{\aleph} \neq y_{\aleph} \implies |x_{\aleph}| \neq |y_{\aleph}| \implies |\text{succ}_{\aleph}(x_{\aleph})| \neq |\text{succ}_{\aleph}(y_{\aleph})| \implies \text{succ}_{\aleph}(x_{\aleph}) \neq \text{succ}_{\aleph}(y_{\aleph})$;

iv) per assurdo, sia $x_{\aleph} \in \aleph$ tale che $\text{succ}_{\aleph}(x_{\aleph}) = 0_{\aleph} := \varnothing$; per definizione $\text{succ}_{\aleph}(x_{\aleph}) := \{\mu_{\aleph} \in \aleph \mid |\mu_{\aleph}| \leq |x_{\aleph}|\}$, ma non esiste $\mu_{\aleph} \in \aleph$ con cardinalità minore o uguale 0, e dunque $\nexists x_{\aleph} \in \aleph \mid \text{succ}_{\aleph}(x_{\aleph}) = 0_{\aleph}$;

v) per assurdo, sia $S \subseteq \aleph$ tale che $0_{\aleph} \in S$ e $\forall x_S \in S \quad \text{succ}_{\aleph}(x_S) \in S$ ma $S \neq \aleph \iff \aleph - S \neq \varnothing \implies \exists \zeta_{\aleph} \in \aleph - S$, ed in particolare $\zeta_{\aleph} \neq 0_{\aleph}$; $\aleph$ è chiuso su $\text{succ}_{\aleph}$ per il secondo assioma di Peano, e dunque $\zeta_{\aleph} \neq 0_{\aleph} \implies \exists \zeta'_{\aleph} \in \aleph \mid \text{succ}_{\aleph}(\zeta'_{\aleph}) = \zeta_{\aleph}$, e sicuramente $\zeta'_{\aleph} \notin S$, poiché altrimenti $\zeta_{\aleph} \in S$ anch'esso in quanto S è chiuso rispetto a $\text{succ}_{\aleph}$; allora, ripetendo il ragionamento analogo per l'intera catena di predecessori, S risulterebbe essere vuoto, ma ciò è impossibile poiché $0_{\aleph} \in S$ in ipotesi $\lightning$.

Principio 1.1.1.1: Principio di Induzione

Sia P una proprietà che vale per $n = 0$, e dunque $P(0)$ è vera; inoltre, per ogni $n \in \mathbb{N}$ si ha che $P(n) \implies P(n+1)$; allora, $P(n)$ è vera per ogni $n \in \mathbb{N}$.

In simboli, utilizzando la notazione della logica formale, si ha che

$$\frac{P(0) \quad \dfrac{\forall n \in \mathbb{N} \quad P(n)}{P(n+1)}}{\forall n \in \mathbb{N} \quad P(n)}$$

Osservazione 1.1.1.1: Quinto assioma di Peano

Si noti che il quinto degli assiomi di Peano (Definizione 1.1.1.1) equivale al principio di induzione (Principio 1.1.1.1). Infatti, il quinto assioma afferma che qualsiasi sottoinsieme S di $\mathbb{N}$ avente lo 0, e caratterizzato dalla chiusura sulla funzione di successore succ, coincide con $\mathbb{N}$ stesso.

1.1.2 Algebre induttive

Definizione 1.1.2.1: Segnatura di una funzione

Data una funzione f, si definisce

$$f : A \to B$$

come **segnatura della funzione** f, dove A è detto **dominio**, denotato con $\mathrm{dom}\,(f)$ e B **codominio** di f.

Definizione 1.1.2.2: Algebra

Una **struttura algebrica**, o più semplicemente **algebra**, consiste di un insieme *non vuoto*, talvolta chiamato **insieme sostegno** (*carrier set* o *domain*), fornito di una o più operazioni su tale insieme, quest'ultime caratterizzate da un numero finito di assiomi da soddisfare.

Se A è il carrier set, e $\gamma_1, \dots \gamma_n$ sono delle operazioni definite su A, allora con

$$(A, \gamma_1, \dots, \gamma_n)$$

si indica l'algebra costituita da tali componenti, e questo simbolismo prende il nome di **segnatura dell'algebra**.

Esempio 1.1.2.1 (Strutture algebriche con singola operazione)**.** Esempi di strutture algebriche con un'operazione binaria sono i seguenti:

- semigruppi
- monoidi
- gruppi
- gruppi abeliani

Esempio 1.1.2.2 (Strutture algebriche con due operazioni)**.** Esempi di strutture algebriche con due operazioni binarie sono i seguenti:

- semianelli
- anelli
- campi

Definizione 1.1.2.3: Insieme unità

Con **insieme unità** si intende un qualsiasi insieme tale che $|\mathbb{1}| = 1$, e verrà indicato attraverso il simbolo $\mathbb{1}$.

Definizione 1.1.2.4: Funzione nullaria

Dato un insieme A, con **funzione nullaria** si intende una qualsiasi funzione con segnatura

$$f : \mathbb{1} \to A$$

Osservazione 1.1.2.1: Iniettività della funzione nullaria

Si noti che ogni funzione nullaria è iniettiva, poiché il dominio è costituito da un solo elemento.

Definizione 1.1.2.5: Algebra induttiva

Sia A un insieme, e siano $\gamma_1, \dots, \gamma_n$ funzioni definite su A di arbitraria arietà; allora, $(A, \gamma_1, \dots, \gamma_n)$ è definita **algebra induttiva** se si verificano le seguenti:

i) $\gamma_1, \dots, \gamma_n$ sono iniettive

ii) $\forall i, j \in [1, n] \mid i \neq j \quad \text{im}(\gamma_i) \cap \text{im}(\gamma_j) = \varnothing$, ovvero, le immagini dei costruttori sono a due a due disgiunte

iii) $\forall S \subseteq A \quad (\forall i \in [1, n], a_1, \dots a_k \in S, k \in \mathbb{N} \quad \gamma_i(a_1, \dots, a_k) \in S) \implies S = A$, o equivalentemente, in A non devono essere contenute algebre induttive.

Le funzioni $\gamma_1, \dots, \gamma_n$ prendono il nome di **costruttori dell'algebra**.

Osservazione 1.1.2.2: Terzo assioma delle algebre induttive

Si noti che nel terzo assioma della Definizione 1.1.2.5 anche $S = \varnothing$ è un valido sottoinsieme di A, ma poiché non esistono $a_1, \dots, a_k \in \varnothing$, in esso ogni qualificazione è vera a vuoto. Di conseguenza, nel momento in cui si ammette $S = \varnothing$ nel terzo assioma, l'algebra risulta essere non induttiva necessariamente (a meno dell'algebra vuota).

Di conseguenza, questo terzo assioma forza la necessità della presenza di un costruttore nullario all'interno di ogni algebra induttiva, in modo da non poter ammettere $S = \varnothing$, poiché l'algebra deve essere chiusa su ognuno dei suoi costruttori.

Esempio 1.1.2.3 (Numeri naturali)**.** $(\mathbb{N}, +)$ non è un algebra induttiva, poiché esistono $x_1, x_2, x_3, x_4 \in \mathbb{N}$ con $x_1 \neq x_3$ e $x_2 \neq x_4$ tali che $x_1 + x_2 = x_3 + x_4$; ad esempio, $2 + 3 = 5 = 1 + 4$, e $2 \neq 1$, $3 \neq 4$.

Esempio 1.1.2.4 (Algebra di Boole)**.** Dato l'insieme $B = \{\text{true}, \text{false}\}$, e la funzione $\neg$ definita come segue

$$\neg : B \to B : x \mapsto \begin{cases} \text{false} & x = \text{true} \\ \text{true} & x = \text{false} \end{cases}$$

è possibile dimostrare che l'algebra $(B, \neg)$ non è induttiva; infatti, nonostante $\neg$ sia iniettiva, e la seconda proprietà della Definizione 1.1.2.5 sia vera a vuoto, $(B, \neg)$ non

presenta costruttore nullario, e dunque non può costituire un'algebra induttiva (si noti l'Osservazione 1.1.2.2).

Esempio 1.1.2.5 (Algebre induttive)**.** Sia zero la funzione definita come segue

$$\text{zero} : \mathbb{1} \to \mathbb{N} : x \mapsto 0$$

e si prenda in esame l'algebra $(\mathbb{N}, \text{succ}, \text{zero})$; allora si ha che

i) succ e zero sono iniettive, poiché
- succ è iniettiva per il terzo assioma di Peano (Definizione 1.1.1.1)
- zero è iniettiva per l'Osservazione 1.1.2.1

ii) $\text{im}(\text{succ}) \cap \text{im}(\text{zero}) = (\mathbb{N} - \{0\}) \cap \{0\} = \varnothing$

iii) TODO

Definizione 1.1.2.6: Omomorfismo

Un **omomorfismo** è una funzione, tra due algebre dello stesso tipo, tale da preservarne le strutture.

Formalmente, siano $(A, \mu_1, \dots, \mu_n)$ e $(B, \delta_1, \dots, \delta_n)$ due algebre tali che ogni funzione μ_i abbia la stessa arietà e lo stesso numero di parametri esterni (denotati con k) di δ_i, pari rispettivamente ad η_i ed a ν_i, per qualche $i \in [1, n]$; allora, una funzione $f : A \to B$ è detta essere un **omomorfismo** tra le due algebre, se e solo se

$$\begin{aligned} \forall a_1, \dots, a_{\eta_1} \quad & f(\mu_1(a_1, \dots, a_{\eta_n}), k_1, \dots, k_{\nu_1}) = \delta_1(f(a_1), \dots, f(a_{\eta_1}), k_1, \dots, k_{\nu_1}) \\ & \vdots \\ \forall a_1, \dots, a_{\eta_n} \quad & f(\mu_n(a_1, \dots, a_{\eta_n}), k_1, \dots, k_{\nu_n}) = \delta_n(f(a_1), \dots, f(a_{\eta_n}), k_1, \dots, k_{\nu_n}) \end{aligned}$$

Esempio 1.1.2.6 (Omomorfismi)**.** Si considerino i gruppi $(\mathbb{R}, +)$ e $(\mathbb{R}_{>0}, \cdot)$, e sia f definita come segue:

$$f : \mathbb{R} \to \mathbb{R}_{>0} : x \mapsto e^x$$

allora, si ha che

$$\forall x, y \in \mathbb{R} \quad f(x) \cdot f(y) = e^x \cdot e^y = e^{x+y} = f(x + y)$$

dunque f è un omomorfismo di gruppi.

Definizione 1.1.2.7: Isomorfismo

Un **isomorfismo** è un omomorfismo biettivo.

Esempio 1.1.2.7 (Isomorfismi)**.** Si consideri l'omomorfismo dell'Esempio 1.1.2.6; si noti che

$$\forall x, y \in \mathbb{R} \mid x \neq \mathbb{R} \quad e^x \neq e^y \implies f(x) \neq f(y)$$

e dunque f è iniettiva; inoltre

$$\forall y \in \mathbb{R}_{>0} \quad \exists x \in \mathbb{R} \mid f(x) = e^x = y \iff y = \ln(x)$$

e dunque f è suriettiva. Allora, f è biettiva, e poiché è un omomorfismo, risulta essere un isomorfismo.

1.2 Strutture dati induttive

1.2.1 Liste

Definizione 1.2.1.1: Liste

Una **lista** è una collezione ordinata di elementi, e l'insieme delle liste di lunghezza finita viene denotato con `List<T>`, dove `T` è il tipo degli elementi che le liste contengono, ed il simobolo `T` verrà identificato con l'insieme di tutti gli oggetti aventi tipo `T`.

Dati $a_1, \ldots, a_n \in \texttt{T}$, una lista $l \in \texttt{List<T>}$ contenente tali elementi può essere rappresentata come segue:

$$[a_1, \ldots, a_n]$$

Definizione 1.2.1.2: Algebra delle liste finite

L'**algebra delle liste finite** è definita come segue:

$$(\texttt{List<T>}, \text{empty}, \text{cons})$$

dove i costruttori sono i seguenti:

$$\text{empty} : \mathbb{1} \to \texttt{List<T>} : x \mapsto []$$
$$\text{cons} : \texttt{List<T>} \times \texttt{T} \to \texttt{List<T>} : ([a_1, \ldots, a_n], x) \mapsto [a_1, \ldots, a_n, x]$$

Proposizione 1.2.1.1: Liste finite induttive

L'algebra delle liste finite è induttiva.

Dimostrazione. Si noti che:

- empty ha dominio in $\mathbb{1}$, e poiché questo contiene un solo elemento, empty è necessariamente iniettiva;
- $\forall l, l' \in \texttt{List<T>}, x, x' \in \texttt{T} \quad \text{cons}(l, x) = \text{cons}(l', x') \implies \begin{cases} l = l' \\ x = x' \end{cases}$ altrimenti l ed l' avrebbero avuto lunghezze diverse, oppure avrebbero contenuto diversi elementi;
- $\text{im}\,(\text{empty}) \cap \text{im}\,(\text{cons}) = \varnothing$, poiché solo empty può restituire `[]`, in quanto cons restituisce sempre una lista contenente almeno l'elemento fornito in input;

- sia $S \subseteq$ `List<T>` tale da essere chiuso rispetto a cons, e contenente la lista vuota; per assurdo, sia `List<T>` $- S \neq \varnothing \implies \exists l \in$ `List<T>` $- S$, ma `List<T>` è chiuso rispetto a cons, ed in particolare $\exists x \in$ `T`$, l' \in$ `List<T>` $\mid \text{cons}(l', x) = l$, ma poichè $l \notin S$, allora necessariamente $l' \notin S$, poiché S è chiuso rispetto a cons, e quindi $l' \in S \implies l \in S$, ma l è stato scelto in `List<T>` $- S$; ripetendo tale ragionamento induttivamente, si ottiene che S è vuoto, ma questo è impossibile poiché `[]` $\in$ `List<T>` in ipotesi $\lightning$.

Dunque, l'algebra delle liste finite risulta essere induttiva. □

Osservazione 1.2.1.1: Algebra delle liste infinite

Se all'algebra delle liste finite venissero aggiunte anche le liste infinite, l'algebra risultante non sarebbe induttiva, in quanto conterrebbe l'algebra delle liste finite, la quale è induttiva per la Proposizione 1.2.1.1, e verrebbe dunque contraddetto il terzo assioma della Definizione 1.1.2.5.

Osservazione 1.2.1.2: Concatenazione di liste finite

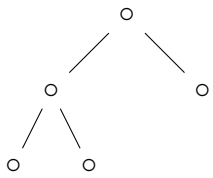
È possibile estendere l'algebra delle liste finite per supportare l'operazione di concatenazione tra liste, come segue:

$$\text{concat} : \texttt{List<T>} \times \texttt{List<T>} \to \texttt{List<T>} : (l, l') \mapsto \begin{cases} l' & l = \texttt{[]} \\ \text{cons}(x, \text{concat}(t, l')) & \exists x \in \texttt{T}, t \in \texttt{List<T>} \mid l = \text{cons}(t, x) \end{cases}$$

1.2.2 Alberi binari

Definizione 1.2.2.1: Albero binario

Un **albero binario** è una struttura dati che è possibile rappresentare graficamente come segue:



Il primo nodo, poiché non è figlio di nessuno, è detto **radice**, e poiché l'albero è *binario*, ogni nodo ha 0 — nel qual caso è definito **foglia** — oppure 2 figli. L'insieme degli alberi binari viene denotato con `B-tree`.

Definizione 1.2.2.2: Algebra degli alberi binari finiti

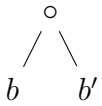
L'**algebra degli alberi binari finiti** è definita come segue:

$$(\texttt{B-tree}, \text{leaf}, \text{branch})$$

dove i costruttori sono i seguenti:

$$\text{leaf} : \mathbb{1} \to \texttt{B-tree} : x \mapsto \circ$$

$$\text{branch} : \texttt{B-tree} \times \texttt{B-tree} \to \texttt{B-tree} : (b, b') \mapsto$$



Proposizione 1.2.2.1: Alberi binari finiti induttivi

L'algebra degli alberi binari finiti è induttiva.

Dimostrazione. Omessa. □

Osservazione 1.2.2.1: Algebra degli alberi binari infiniti

Analogamente all'Osservazione 1.2.1.1, l'algebra degli alberi binari finiti ed infiniti non è induttiva.

Osservazione 1.2.2.2: Nodi di un albero binario finito

È possibile estendere l'algebra degli alberi binari finiti per supportare l'operazione per contare i nodi di un albero, come segue:

$$\text{nodes} : \texttt{B-tree} \to \mathbb{N} : b \mapsto \begin{cases} 1 & b = \circ \\ 1 + \text{nodes}(t) + \text{nodes}(t') & \exists t, t' \in \texttt{B-tree} \mid b = \text{branch}(t, t') \end{cases}$$

Osservazione 1.2.2.3: Foglie di un albero binario finito

È possibile estendere l'algebra degli alberi binari finiti per supportare l'operazione per contare le foglie di un albero, come segue:

$$\text{leaves} : \texttt{B-tree} \to \mathbb{N} : b \mapsto \begin{cases} 1 & b = \circ \\ \text{leaves}(t) + \text{leaves}(t') & \exists t, t' \in \texttt{B-tree} \mid b = \text{branch}(t, t') \end{cases}$$

Teorema 1.2.2.1: Relazione tra foglie e nodi

Ogni albero binario finito, avente n foglie, ha $2n - 1$ nodi.

Dimostrazione. La seguente dimostrazione procede per *induzione strutturale*, dunque effettuando l'induzione sulla morfologia della struttura dati, e non sul numero n di foglie.

Caso base. Il caso base è costituito dunque da $\circ$, l'albero ottenuto attraverso il costruttore nullario leaf, ed infatti si ha che

$$\text{leaves}(\circ) = 1 \implies 2 \cdot 1 - 1 = 1$$

e $\circ$ ha esattamente 1 nodo.

Ipotesi induttiva. Un albero binario finito, avente n foglie, ha $2n - 1$ nodi.

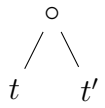
Passo induttivo. Sia $b \in$ `B-tree` tale che esistano $t, t' \in$ `B-tree` tali che $\text{branch}(t, t') = b$, e siano

$$\begin{cases} \text{leaves}(t) = n \\ \text{leaves}(t') = n' \end{cases}$$

Si noti che, per ipotesi induttiva, si ha che

$$\begin{cases} \text{nodes}(t) = 2n - 1 \\ \text{nodes}(t') = 2n' - 1 \end{cases}$$

ed inoltre, poiché $b = \text{branch}(t, t')$, b ha la forma seguente



dunque, per definizione di leaves si ha che

$$\text{leaves}(b) = \text{leaves}(t) + \text{leaves}(t') = n + n'$$

e, dalla morfologia di b, segue che

$$\text{nodes}(b) = \text{nodes}(t) + \text{notes}(t') + 1 = 2n - 1 + 2n' - 1 + 1 = 2(n + n') - 1$$

ed è quindi verificata la tesi, poiché

$$\text{leaves}(b) = n + n' \implies \text{nodes}(b) = 2(n + n') - 1$$

□

2 Paradigma funzionale

2.1 Grammatiche

2.1.1 Definizioni

Definizione 2.1.1.1: Grammatica

Una **grammatica** è un insieme di regole che definiscono come manipolare un insieme di stringhe, agendo su elementi sintattici detti **termini**.

Definizione 2.1.1.2: Forma di Backus-Naur (BNF)

La **forma di Backus-Naur** (*Backus-Naur Form*) è una notazione utilizzata per descrivere la sintassi di grammatiche, ed è definita come segue:

$$\texttt{<symbol>}, \ldots, \texttt{<symbol>} ::= \texttt{\_expression\_} \mid \ldots \mid \texttt{\_expression\_}$$

dove

- `<symbol>` è una *metavariabile non terminale*, ovvero, può essere sostituita con regole definite dalla grammatica; si noti che le regole possono essere utilizzate ricorsivamente;
- il simbolo ::= indica che ciò che è posto alla sua sinistra deve essere sostituito con ciò che è alla sua destra;
- `_expression_` è un espressione che verrà usata per rimpiazzare le metavariabili non terminali, attraverso le regole definite dalla grammatica; le *metavarabili* che compongono le espressioni possono essere **costanti**, **variabili**, **termini** o **espressioni** contenenti combinazioni delle precedenti, presentando eventualmente anche operazioni sintattiche specifiche.

Esempio 2.1.1.1 (Grammatica *Exp*)**.** Sia *Exp* la seguente grammatica:

$$M, N ::= \texttt{0} \mid \texttt{1} \mid \ldots \mid x \mid M \texttt{ + } N \mid M \texttt{ * } N$$

essa definisce le regole per utilizzare i numeri in $\mathbb{N}$, ammettendo inoltre le operazioni sintattiche di `+` e `*`.

All'interno di questa grammatica dunque, le metavariabili utilizzate sono le seguenti:

- *costanti*: $\texttt{0}, \texttt{1}, \ldots$
- *variabili*: x
- *termini*: M ed N
- *espressioni*: M `+` N e M `*` N (tecnicamente anche le precedenti sono espressioni, ma queste in particolare comprendono anche operazioni sintattiche)

Definizione 2.1.1.3: Variabili

Data una grammatica di G, con Var si indica l'**insieme delle variabili** di G.

Definizione 2.1.1.4: Valori

Data una grammatica, con Val si indica l'**insieme dei valori** che ogni termine della grammatica può assumere.

Esempio 2.1.1.2 (Variabili e valori di *Exp*)**.** Si prenda in considerazione la grammatica *Exp* dell'Esempio 2.2.1.1; in essa, si ha che

$$\text{Var} = \{x\}$$
$$\text{Val} = \{\texttt{0}, \texttt{1}, \ldots\}$$

Definizione 2.1.1.5: Linguaggio di una grammatica

Sia G una grammatica; allora, il suo **linguaggio** è l'insieme delle stringhe che è possibile costruire attraverso le regole dettate da G.

Esempio 2.1.1.3 (Linguaggio di *Exp*)**.** Sia *Exp* la grammatica dell'Esempio 2.1.1.1; in essa, considerando ad esempio le stringhe `"4"` e `"23"`, si può ottenere la stringa

$$\texttt{+("4", "23") = "4 + 23"}$$

dove la *polish notation* — alla sinistra dell'uguale — e la forma sintattica canonica — alla sua destra — verranno utilizzate intercambiabilmente, poiché puro *syntactic sugar*.

Osservazione 2.1.1.1: Valutazione di *Exp*

Si prenda in considerazione la grammatica *Exp* dell'Esempio 2.1.1.1; su di essa, è possibile definire ricorsivamente una funzione eval, in grado di valutare le stringhe che tale grammatica può produrre, come segue:

$$\begin{aligned} \text{eval}(\texttt{0}) &= \texttt{0} \\ \text{eval}(\texttt{1}) &= \texttt{1} \\ &\vdots \\ \text{eval}(M \texttt{ + } N) &= \text{eval}(M) \texttt{ + } \text{eval}(N) \\ \text{eval}(M \texttt{ * } N) &= \text{eval}(M) \texttt{ * } \text{eval}(N) \end{aligned}$$

Osservazione 2.1.1.2: Ambiguità di *Exp*

Si prenda in considerazione la grammatica *Exp* dell'Esempio 2.1.1.1; si noti che tale grammatica è ambigua, poichè ad esempio

$$\texttt{+}(\texttt{"5"}, \texttt{*}(\texttt{"6"}, \texttt{"7"})) = \texttt{"5 + 6 * 7"} = \texttt{*}(\texttt{+}(\texttt{"5"}, \texttt{"6"}), \texttt{"7"})$$

e da ciò segue anche che $\text{im}\,(\texttt{+}) \cap \text{im}\,(\texttt{*}) \neq \varnothing$.

Osservazione 2.1.1.3: Disambiguazione di *Exp*

Si noti che l'ambiguità trattata nell'Osservazione 2.1.1.2 non permetterebbe di poter definire la funzione eval, descritta nell'Osservazione 2.1.1.1. Dunque, per risolvere tale ambiguità, a meno di parentesi (che *non* sono definite all'interno della grammatica) o dell'esplicitazione della composizione di funzioni utilizzata, verrà sottintesa la normale precedenza degli operatori aritmetica durante la valutazione delle stringhe.

2.2 Assegnazioni

2.2.1 Definizioni

Definizione 2.2.1.1: Clausola *let*

Sia G una grammatica; allora, è possibile definire su G una funzione *let*, come segue:

$$let : \text{Var} \times G \times G \to G$$

e verrà utilizzata attraverso la sintassi

$$let\ \texttt{*variable*} = \texttt{\_expression\_}_1 \text{ in } \texttt{\_expression\_}_2$$

dove alla variabile `*variable*` verrà assegnata l'espressione `_expression_`$_1$ *durante la valutazione* di `_expression_`$_2$; la variabile `*variable*`, all'interno di `_expression_`$_2$, prende il nome di **variabile locale**.

Una variabile alla quale non è stata assegnata nessuna espressione prende il nome di **variable libera** (*unbound variable* in inglese); una variabile non libera è detta **variabile legata** (*bound variable*).

Esempio 2.2.1.1 (Estensione di *Exp*)**.** Sia *Exp* la seguente estensione della grammatica presente all'interno dell'Esempio 2.1.1.1:

$$M, N ::= k \mid x \mid M \texttt{ + } N \mid M \texttt{ * } N \mid let\ x = M \text{ in } N$$

In essa, sono presenti:

- *costanti*: indicate con k, che sta ad indicare che in *Exp* è ammessa qualsiasi costante; di fatto, è possibile pensare a k come una funzione definita come segue:

$$k : \mathbb{N} \to Exp : x \mapsto \texttt{"x"}$$

- *variabili*: x
- *termini*: M ed N
- *espressioni*: M `+` N, M `*` N e *let* $x = M$ in N

Esempio 2.2.1.2 (Clausole *let*)**.** Sia *Exp* la grammatica dell'Esempio 2.2.1.1; un esempio di espressione su *Exp*, che utilizza la clausola *let* della Definizione 2.2.1.1, è la seguente:

$$let\ x = 3 \text{ in } (x + 1)$$

e nel momento in cui viene valutata tale espressione, si ha che

$$x = 3 \implies x + 1 = 3 + 1 = 4$$

e dunque il valore dell'espressione è 4.

Esempio 2.2.1.3 (Variabili libere)**.** Sia Exp la grammatica dell'Esempio 2.2.1.1, ed ammettendo la variabile y in essa, si consideri la seguente espressione:

$$let\ x = 3\ \text{in}\ (x + y)$$

in essa, la variabile x è posta pari a 3, ma ad y non è stato assegnato alcuna espressione, e dunque risulta essere una variabile libera.

Osservazione 2.2.1.1: Ambiguità di *let*

Sia Exp la grammatica dell'Esempio 2.2.1.1, e si consideri la sua seguente espressione

$$let\ x = M\ \text{in}\ x + y$$

per qualche espressione $M \in Exp$, e due variabili $x, y \in \text{Var}$, ammettendo dunque y tra le variabili di Exp; si noti che tale espressione è ambigua, poiché potrebbe equivalere a

$$(let\ x = M\ \text{in}\ x) + y$$

oppure a

$$let\ x = M\ \text{in}\ (x + y)$$

Per convenzione, all'interno di questi appunti, in assenza di parentesi che descrivano la precedenza degli operatori, si assume la precedenza della seconda espressione mostrata.

Osservazione 2.2.1.2: Variabili libere di un'espressione

Sia Exp la grammatica dell'Esempio 2.2.1.1; su di essa, è possibile definire, ricorsivamente, una funzione in grado di restituire le variabili unbound di una data espressione, come segue:

$$\text{free} : Exp \to \mathcal{P}(\text{Var}) : e \mapsto \begin{cases} \varnothing & \exists \eta \in \mathbb{N} \mid e = k(\eta) \\ \{x\} & \exists x \in \text{Var} \mid e = x \\ \text{free}(M) \cup \text{free}(N) & \exists M, N \in Exp \mid e = M + N \vee e = M * N \\ \text{free}(M) \cup (\text{free}(N) - \{x\}) & \exists x \in \text{Var}, M, N \in Exp \mid e = (let\ x = M\ \text{in}\ N) \end{cases}$$

2.2.2 Ambienti

Definizione 2.2.2.1: Ambiente di una grammatica

Data una grammatica tale che Val sia un insieme finito, un **ambiente** della grammatica è una funzione della forma

$$E : \text{Var} \xrightarrow{fin} \text{Val}$$

che associa dunque una variabile ad un possibile valore che può assumere (la notazione fin indica che E è una funzione *parziale*, dunque non necessariamente definita su tutto il dominio). L'insieme di tutti gli ambienti della grammatica è denotato con

$$\text{Env} := \{f \mid f : \text{Var} \xrightarrow{fin} \text{Val}\}$$

In simboli, gli ambienti verranno scritti come insiemi di coppie (x, k) con $x \in \text{Var}, k \in \text{Val}$, che descriveranno la mappa definita dall'ambiente stesso. Si noti che, per un certo ambiente $E \in \text{Env}$, $E(x)$ è indefinito per ogni $x \in \text{Var} - \text{dom}(E)$.

Esempio 2.2.2.1 (Ambienti di Exp)**.** Sia Exp la grammatica dell'Esempio 2.2.1.1; allora, un possibile ambiente di Exp, denotato con $E \in \text{Env}$, è il seguente:

$$E := \{(z, 3), (y, 9)\}$$

ed esso esprime la possibilità che in Exp z possa essere valutato pari a 3, mentre y pari a 9 (tecnicamente, le variabili z ed y andrebbero ammesse all'interno della grammatica, ma d'ora in avanti tale precisazione verrà sottintesa).

Definizione 2.2.2.2: Concatenazione di ambienti

Siano E_1 ed E_2 due ambienti di una grammatica; allora, si definisce **concatenazione** di E_1 ed E_2 la seguente funzione

$$E_1E_2 : \text{Env} \times \text{Env} \to \text{Env} : x \mapsto \begin{cases} E_2 & x \in \text{dom}\,(E_2) \vee x \in \text{dom}\,(E_1) \cap \text{dom}\,(E_2) \\ E_1(x) & x \in \text{dom}\,(E_1) \end{cases}$$

dunque, nella concatenazione E_2 sovrascrive le tuple che sono presenti anche in E_1.

Esempio 2.2.2.2 (Concatenazioni di ambienti)**.** Sia Exp la grammatica descritta all'interno del Esempio 2.2.1.1, e siano

$$E_1 := \{(z, 3), (y, 9)\}$$
$$E_2 := \{(z, 4)\}$$

due suoi ambienti; allora si ha che

$$E_1E_2 := \{(z, 4), (y, 9)\}$$

2.2.3 Semantica operazionale

Definizione 2.2.3.1: Semantica operazionale di una grammatica

Data una grammatica G, si definisce **semantica operazionale** della grammatica una relazione, indicata col simbolo $\rightsquigarrow$, definita come segue:

$$\rightsquigarrow \subseteq \text{Env} \times G \times \text{Val}$$

Un elemento $(E, M, v) \in \rightsquigarrow$ è detto **giudizio operazionale**, e viene scritto attraverso il seguente simbolismo:

$$E \vdash M \rightsquigarrow v$$

e si legge "valutando M, nell'ambiente E, si ottiene v".

Esempio 2.2.3.1 (Semantica operazionale di Exp)**.** Sia Exp la grammatica definita all'interno dell'Esempio 2.2.1.1, e sia E un suo ambiente; allora, sono vere le seguenti asserzioni:

$$[const]\ E \vdash k \rightsquigarrow k$$

$$\forall x \in \text{Var} \quad \exists v \in \text{Val} \mid E(x) = v \implies [var]\ E \vdash x \rightsquigarrow v$$

$$\forall v', v'' \in \text{Val}, M, N \in Exp \quad \exists v \in \text{Val} \mid v = v' + v'' \implies [plus]\ \frac{E \vdash M \rightsquigarrow v' \quad E \vdash N \rightsquigarrow v''}{E \vdash M + N \rightsquigarrow v}$$

$$\forall v', v'' \in \text{Val}, M, N \in Exp \quad \exists v \in \text{Val} \mid v = v' \cdot v'' \implies [times]\ \frac{E \vdash M \rightsquigarrow v' \quad E \vdash N \rightsquigarrow v''}{E \vdash M * N \rightsquigarrow v}$$

$$\forall v, v' \in \text{Val}, x \in \text{Var}, M, N \in Exp \quad [let]\ \frac{E \vdash M \rightsquigarrow v' \quad E\{(x, v')\} \vdash N \rightsquigarrow v}{E \vdash let\ x = M\ \text{in}\ N \rightsquigarrow v}$$

Definizione 2.2.3.2: Equivalenza operazionale

Sia G una grammatica, e siano M ed N due sue espressioni; queste sono dette **operazionalmente equivalenti**, se è vera la seguente proposizione:

$$\forall E \in \text{Env}, v \in \text{Val} \quad E \vdash M \rightsquigarrow v \iff E \vdash N \rightsquigarrow v$$

e viene indicato con $M \sim N$.

Definizione 2.2.3.3: Albero di valutazione

Con **albero di valutazione** di un'espressione e, si definisce l'albero, composto da inferenze logiche, ottenuto dalla valutazione di e.

Osservazione 2.2.3.1: Ambiente iniziale

Per qualsiasi grammatica — a meno di specifiche — si assume che, all'interno di una valutazione, l'ambiente iniziale sia $\varnothing \in \text{Env}$.

Esempio 2.2.3.2 (Alberi di valutazione su *Exp*)**.** Sia *Exp* la grammatica definita all'interno dell'Esempio 2.2.1.1; allora, l'albero di valutazione dell'espressione

$$let\ x = 3\ \text{in}\ x + 4$$

è il seguente

$$\dfrac{\varnothing \vdash 3 \rightsquigarrow 3 \quad \dfrac{\{(x,3)\} \vdash x \rightsquigarrow 3 \quad \{(x,3)\} \vdash 4 \rightsquigarrow 4}{\{(x,3)\} \vdash x + 4 \rightsquigarrow 7}}{\varnothing \vdash let\ x = 3\ \text{in}\ x + 4 \rightsquigarrow 7}$$

e l'espressione è valutabile poiché $x \in \text{dom}(\{(x,1)\}) = \{x\}$.

2.3 Valutazioni e scoping

2.3.1 Definizioni

Definizione 2.3.1.1: Valutazione eager

Data una grammatica, la **valutazione eager** (*call-by-name*) valuta una data espressione della grammatica non appena questa viene legata ad una variabile.

Definizione 2.3.1.2: Valutazione lazy

Data una grammatica, la **valutazione lazy** (*call-by-need*) valuta una data espressione della grammatica solo quando il suo valore viene richiesto da un'altra espressione.

Definizione 2.3.1.3: Scoping statico

Data una grammatica, la valutazione a **scoping statico** (*lexical scope*) valuta le variabili TODO

Definizione 2.3.1.4: Scoping dinamico

Data una grammatica, la valutazione a **scoping dinamico** valuta le variabili utilizzando l'ambiente definito in tempo di valutazione. TODO NON MI PIACE CAMBIA

Osservazione 2.3.1.1: *Exp* eager

Sia *Exp* la grammatica definita all'interno dell'Esempio 2.2.1.1; in essa, la valutazione eager statica e dinamica sono equivalenti, poiché nessun espressione della grammatica è influenzata dal tipo si scoping valutando in maniera eager; di conseguenza, non è necessario ridefinire le clausole viste nell'Esempio 2.2.3.1.

Esempio 2.3.1.1 (Valutazioni eager su Exp)**.** Sia Exp la grammatica definita all'interno dell'Esempio 2.2.1.1, e si consideri la seguente espressione:

$$let\ x = 3\ \text{in}\ (let\ y = x\ \text{in}\ (let\ x = 7\ \text{in}\ y + x))$$

essa, valutata attraverso valutazione eager, produce il seguente albero di derivazione:

$$\dfrac{\varnothing \vdash 3 \rightsquigarrow 3 \quad \dfrac{\{(x,3)\} \vdash x \rightsquigarrow 3 \quad \dfrac{\{(x,3),(y,3)\} \vdash 7 \rightsquigarrow 7 \quad \dfrac{\{(x,3),(y,3)\}\{(x,7)\} \vdash y \rightsquigarrow 3 \quad \{(x,3),(y,3)\}\{(x,7)\} \vdash x \rightsquigarrow 7}{\{(x,3),(y,3)\}\{(x,7)\} \vdash y + x \rightsquigarrow 10}}{\{(x,3),(y,3)\} \vdash let\ x = 7\ \text{in}\ y + x \rightsquigarrow 10}}{\{(x,3)\} \vdash let\ y = x\ \text{in}\ (let\ x = 7\ \text{in}\ y + x) \rightsquigarrow 10}}{\varnothing \vdash let\ x = 3\ \text{in}\ (let\ y = x\ \text{in}\ (let\ x = 7\ \text{in}\ y + x)) \rightsquigarrow 10}$$

Osservazione 2.3.1.2: Exp lazy dinamica

Sia Exp la grammatica definita nell'Esempio 2.2.1.1; per poter valutare le sue espressioni in maniera lazy dinamica, è necessario ridefinire alcune regole di inferenza definite all'interno dell'Esempio 2.2.3.1.

- l'insieme degli ambienti di Exp viene ridefinito come segue:

  $$\text{Env} := \{f \mid \text{Var} \xrightarrow{fin} \text{Val} \cup Exp\}$$

  dunque gli ambienti possono associare delle variabili anche a delle espressioni, in modo da poter ritardare la valutazione di quest'ultime fin quando non diventa strettamente necessario conoscerne il valore;

- $\forall E \in \text{Env}, x \in \text{Var} \quad x \in \text{dom}(E) \wedge M := E(x) \implies \dfrac{E \vdash M \rightsquigarrow v}{E \vdash x \rightsquigarrow v}$, per rendere M calcolabile, attraverso l'ambiente corrente, nel momento in cui viene assegnata ad una variabile;

- $\forall x \in \text{Var}, M, N \in Exp \quad [let]\ \dfrac{E\{(x,M)\} \vdash N \rightsquigarrow v}{E \vdash let\ x = M\ \text{in}\ N \rightsquigarrow v}$, in modo da ritardare la valutazione di M.

Si noti che tale valutazione utilizza lo scoping dinamico, poiché viene utilizzato l'ambiente corrente per valutare le variabili.

Esempio 2.3.1.2 (Valutazioni lazy dinamiche su Exp)**.** Si consideri l'espressione definita all'interno dell'Esempio 2.3.1.1; essa, valutata attraverso valutazione lazy dinamica, produce il seguente albero di derivazione:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{(x,3),(y,x)\}\{(x,7)\} \vdash 7 \rightsquigarrow 7}{\{(x,3),(y,x)\}\{(x,7)\} \vdash x \rightsquigarrow 7} \quad \dfrac{\{(x,3),(y,x)\}\{(x,7)\} \vdash 7 \rightsquigarrow 7}{\{(x,3),(y,x)\}\{(x,7)\} \vdash x \rightsquigarrow 7}}{\{(x,3),(y,x)\}\{(x,7)\} \vdash y \rightsquigarrow 7} \quad \dfrac{\{(x,3),(y,x)\}\{(x,7)\} \vdash 7 \rightsquigarrow 7}{\{(x,3),(y,x)\}\{(x,7)\} \vdash x \rightsquigarrow 7}}{\{(x,3),(y,x)\}\{(x,7)\} \vdash y + x \rightsquigarrow 14}}{\{(x,3),(y,x)\} \vdash let\ x = 7\ \text{in}\ y + x \rightsquigarrow 14}}{\{(x,3)\} \vdash let\ y = x\ \text{in}\ (let\ x = 7\ \text{in}\ y + x) \rightsquigarrow 14}}{\varnothing \vdash let\ x = 3\ \text{in}\ (let\ y = x\ \text{in}\ (let\ x = 7\ \text{in}\ y + x)) \rightsquigarrow 14}$$

Osservazione 2.3.1.3: *Exp* lazy statica

Sia *Exp* la grammatica definita nell'Esempio 2.2.1.1; per poter valutare le sue espressioni in maniera lazy statica, è necessario ridefinire alcune regole di inferenza definite all'interno dell'Esempio 2.2.3.1.

- l'insieme degli ambienti di *Exp* viene ridefinito come segue:

$$\text{Env} := \{f \mid \text{Var} \xrightarrow{fin} Exp \times \text{Env}\}$$

  dunque gli ambienti possono associare delle variabili anche a delle tuple contenenti espressioni ed ambienti; in particolare, per una tupla $(x, (M, E))$, si ha che x vale M valutata nell'ambiente E; questa correzione nella definizione di Env permette di tenere traccia degli ambienti in cui sono state fatte le assegnazioni;

- $\forall E \in \text{Env}, x \in \text{Var} \quad x \in \text{dom}(E) \land (M, E') := E(x) \implies \dfrac{E' \vdash M \rightsquigarrow v}{E \vdash x \rightsquigarrow v}$, affinché sia possibile valutare le variabili esattamente nell'ambiente in cui gli è stata assegnata l'espressione M, e non nell'ambiente corrente;

- $\forall x \in \text{Var}, M, N \in Exp \quad [let] \dfrac{E\{(x, (M, E))\} \vdash N \rightsquigarrow v}{E \vdash let\ x = M\ \text{in}\ N \rightsquigarrow v}$, in modo da salvare anche l'ambiente in cui alla variabile x è stata assegnata l'espressione M.

Si noti che tale valutazione utilizza lo scoping statico, poiché vengono salvati anche gli ambienti in cui sono state fatte le assegnazioni.

Esempio 2.3.1.3 (Valutazioni lazy statiche su *Exp*)**.** Si consideri l'espressione definita all'interno dell'Esempio 2.3.1.1; essa, valutata attraverso valutazione lazy statica, produce il seguente albero di derivazione:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varnothing \vdash 3 \rightsquigarrow 3}{E \vdash x \rightsquigarrow 3} \quad \dfrac{E' \vdash 7 \rightsquigarrow 7}{E'' \vdash x \rightsquigarrow 7}}{E'' \vdash y \rightsquigarrow 3}}{E'\{(x, (7, E'))\} \vdash y + x \rightsquigarrow 10}}{E\{(y, (x, E))\} \vdash let\ x = 7\ \text{in}\ y + x \rightsquigarrow 10}}{\{(x, (3, \varnothing))\} \vdash let\ y = x\ \text{in}\ (let\ x = 7\ \text{in}\ y + x) \rightsquigarrow 10}}{\varnothing \vdash let\ x = 3\ \text{in}\ (let\ y = x\ \text{in}\ (let\ x = 7\ \text{in}\ y + x)) \rightsquigarrow 10}$$

dove

$$E := \{(x, (3, \varnothing))\}$$
$$E' := E\{(y, (x, E))\} = \{(x, (3, \varnothing))\}\{(y, (x, \{(x, (3, \varnothing))\}))\}$$
$$E'' := E'\{(x, (7, E'))\} = \{(x, (3, \varnothing))\}\{(y, (x, \{(x, (3, \varnothing))\}))\}\{(x, (7, \{(x, (3, \varnothing))\}\{(y, (x, \{(x, (3, \varnothing))\}))\}))\}$$

2.4 Funzioni

2.4.1 Definizioni

Definizione 2.4.1.1: Clausola fn

Sia G una grammatica; allora, è possibile definire su G una funzione fn, come segue:

$$fn : \text{Var} \times G \to G$$

e verrà utilizzata attraverso la sintassi

$$fn \texttt{ *variable*} \Rightarrow \texttt{\_expression\_}$$

che restituisce una funzione avente come parametro `*variable*`, il cui valore sarà utilizzato per valutare `_expression_`.

Definizione 2.4.1.2: Applicazione

Sia G una grammatica; allora, dati due suoi termini M, N, è possibile definire su G l'applicazione di M ad N, attraverso la seguente sintassi:

$$MN$$

Tale sintassi è presa in prestito dal lambda calcolo.

Si noti che un'espressione MNL applica prima M ad N, e poi MN ad L, dunque la precedenza è da sinistra verso destra, ovvero $(MN)L$.

Esempio 2.4.1.1 (Grammatica Fun)**.** Sia Fun la seguente estensione della grammatica Exp, definita all'interno dell'Esempio 2.2.1.1:

$$M, N ::= k \mid x \mid M \texttt{ + } N \mid M \texttt{ * } N \mid let\ x = M \text{ in } N \mid fn\ x \Rightarrow M \mid MN$$

Inoltre, l'insieme dei valori assegnabili ai termini di Fun verrà esteso come segue

$$\text{Val} := \{\texttt{0}, \texttt{1}, \ldots\} \cup (\text{Var} \times Fun \times Fun)$$

dunque ad una variabile sarà possibile assegnargli il valore che una clausola *let* restituisce TODO DA CAMBIARE, ASSICURATI DI COME VA FIXATO, CREDO SIANO FN E NON LET

Esempio 2.4.1.2 (Espressioni su Fun)**.** Sia Fun la grammatica definita all'interno dell'Esempio 2.4.1.1, e sia

$$(fn\ x \Rightarrow x + 1)7$$

una sua espressione; essa, poiché applica la funzione $fn\ x \Rightarrow x + 1$ all'espressione 7, viene valutata a

$$x = 7 \implies x + 1 = 7 + 1 = 8$$

Esempio 2.4.1.3 (Espressioni su Fun)**.** Sia Fun la grammatica definita all'interno dell'Esempio 2.4.1.1, e sia

$$(fn\ x \Rightarrow x3)(fn\ x \Rightarrow x + 1)$$

una sua espressione; essa, una volta valutata, applica inizialmente la funzione $fn\ x \Rightarrow x$ all'espressione 3, e dunque il valore risultante è 3, e successivamente applica la funzione $fn\ x \Rightarrow x + 1$ all'espressione 3, e dunque il suo valore è pari a

$$x = 3 \implies x + 1 = 3 + 1 = 4$$

TODO DA RIVEDERE CHE NON SI CAPISCE NIENTE

Osservazione 2.4.1.1: Curryficazione

Si consideri la clausola fn della Definizione 2.4.1.1; è possibile definirne una notazione contratta, che prende il nome di *curryficazione*, ed è definita come segue:

$$fn\ x_1 x_2 \dots x_n \Rightarrow M \iff fn\ x_1 \Rightarrow (fn\ x_2 \Rightarrow \dots (fn\ x_n \Rightarrow M) \dots)$$

Il processo inverso prende il nome di *uncurryficazione*.

Esempio 2.4.1.4 (Curryficazioni)**.** Sia Fun la grammatica dell'Esempio 2.4.1.1, e sia

$$(fn\ xy \Rightarrow yx)7(fn\ x \Rightarrow x + 1)$$

una sua espressione; la sua uncurryficazione corrisponde alla seguente espressione

$$(fn\ x \Rightarrow fn\ y \Rightarrow yx)7(fn\ x \Rightarrow x + 1)$$

TODO COME SE VALUTA