Corso di introduzione al linguaggio Scheme

Marco Maggesi <maggesi@math.unifi.it>
Questo documento è disponibile in linea: http://www.math.unifi.it/~maggesi/scheme/

Queste note, oltre a raccogliere il materiale necessario per il corso, costituiscono anche una selezione, secondo il mio gusto, di altro materiale disponibile.

1 Motivazioni ad imparare e usare lo Scheme

1. **E' un linguaggio facile, adatto ai neofiti.** Infatti lo Scheme viene sempre più usato nelle università e nei college di tutto il mondo come base per insegnare a programmare (cfr. Teach-Scheme fixme: curriculum, arsdigita, sicm). Questo significa anche che in futuro il linguaggio Scheme sarà parte di un background culturale diffuso.
2. **E' un linguaggio che può essere facilmente immerso (*embedded*) ed esteso.** Quindi è un linguaggio ideale per programmatori e per chi sviluppa nuovi sistemi. La Free Software Foundation ha scelto i linguaggi C e Scheme come base per il sistema operativo GNU (fixme: aggiungere i link al guile e al manifesto). Il numero degli applicativi e dei sistemi basati sullo Scheme cresce di mese in mese (es. Gimp (fixme: gimp brl).
3. **E' un linguaggio potente e flessible con un semantica pulita.** Per questo è un linguaggio che molti ricercatori usano ormai da più di vent'anni per implementare e sperimentare le loro idee.
4. **Dà diretto supporto a molti stili di programmazione**: Tutti i paradigmi di programmazione da i più diffusi a quelli più esotici trovano un modo adeguato di essere messi in atto. Alcuni esempi: programmazione imperativa, funzionale, a passaggio della continuazione (continuation passing style - CPS), logica, orientata agli oggetti, non deterministica. Un programmatore preparato usare il paradigma giusto al momento giusto in ogni occasione. Un neofita può esercitarsi e mettere in pratica molti concetti con un solo linguaggio con grande risparmio di tempo e di stress da *cyber anxiety*.
5. **Molte delle obiezioni mosse contro lo Scheme sono oggi superate.** Alle sue origini, quasi trent'anni fa, lo Scheme veniva considerato un linguaggio buono per la teoria (come moltri altri dialetti Lisp), ma irrimediabilmente inefficiente nella pratica. Nel frattempo la teoria dei compilatori e degli interpreti ha fatto dei notevoli progressi. Oggi esistono compilatori per lo Scheme molto efficienti (fixme: Stalin, Chicken). Esistono anche raffinati strumenti di analisi statica (fixme: MrSpider) e interpreti dello Scheme che danno un valido supporto nella fase di scrittura e debugging dello Scheme.

2 Materiale per il corso

1. Il testo guida per il corso sarà Teach Yourself Scheme in Fixnum Days [7] di Dorai Sitaram. Per comodità è disponibile una copia locale della versione Html.
2. Le esercitazioni saranno fatte inizialmente con il sistema DrScheme. L'ambiente DrScheme è pensato proprio per la didattica della programmazione ed è disponibile per varie piattaforme tra cui Linux e Windows. Chi segue il corso dovrebbe mettersi in codizione di poter usare questo sistema per gli esercizi. L'aula studenti del centro di calcolo del Dipartimento di Matematica "U. Dini" verrà attrezzata con questo software. Successivamente saranno usati anche altri ambienti tra cui MIT-Scheme e SCM.[2]

[2] DrScheme è disponibile da http://www.cs.rice.edu/CS/PLT/packages/drscheme/.
MIT-Scheme è disponibile da http://www.swiss.ai.mit.edu/projects/scheme/.
SCM è disponibile da http://www-swiss.ai.mit.edu/~jaffer/SCM.html.

3 Materiale disponibile via web

- **R5RS: L'ultima specifica del linguaggio** *"The Revised5 Report on the Algorithmic Language Scheme"* [**6**]. Formati disponibili Postscript (870KB), Gzipped Postscript (215KB), Html (è disponibile per comodità anche una copia locale).
- **Le pagine del MIT sullo Scheme** (`http://www.swiss.ai.mit.edu/projects/scheme/`). Il MIT è stato uno dei centri di ricerca che ha contribuito di più alla nascita e allo sviluppo dello Scheme. Questo sito è stato fino ad oggi il punto di riferimento su internet.
- **Schemers.org** (`http://www.schemers.org/`). Il sito che sta cercando di diventare (riuscendoci) il "portale" sullo Scheme. E' gestito dal PLT.
- **Versione online di SICP** (`http://sicp.arsdigita.org/text/sicp/`). Il sito del libro *"Structure and Interpretation of Computer Programs"* [1].
- **Versione online di HTDP** (`http://www.htdp.org/`). Il sito del libro *"How To Design Programs"* [4].
- **SRFI:** *Scheme Request For Implementation* (`http://srfi.schemers.org/`). L'obiettivo dell'SRFI (nato nel 1998) è quello di rendere standard alcune estensioni dello Scheme con un meccanismo che ricorda molto quello degli RFC di Internet. Il legame fra SRFI e RnRS è per certi aspetti contorto: diciamo che SRFI è nato per essere uno standard un po' meno standard (ma anche più libero e flessibile) degli standard RnRS.

4 Materiale per le lezioni

- Lezione n. 1. *"Da zero alla ricorsione in coda."* Dopo aver introdotto la sintassi e l'uso di alcune procedure fondamentali si passa ad esaminare il comportamento di una chiamata ricorsiva in coda con l'ausilio dello stepper. (Soluzioni degli esercizi della prima lezione.)
- Lezione n. 2. *"Esempi di funzioni per la manipolazione di liste."* Breve ripasso sulle funzioni primitive che operano sulle liste e introduzione di alcune funzioni di uso comune come '`append`' e '`reverse`' e delle loro possibili implementazioni. (Soluzioni degli esercizi della seconda lezione.)
- Lezione n. 3. *Uso degli stream.* Gli stream sono facilmente implementabili in Scheme e sono uno strumento potente nella programmazione funzionale. Una libreria di funzioni per implementare gli stream è localmente disponibile da `http://www.math.unifi.it/~maggesi/scheme/stream.scm`. E' disponibile anche un esempio più elaborato di uso degli stream: un semplice giocatore di Master-Mind.

5 I punti di forza dello Scheme

Dal punto di vista concettuale, lo Scheme si distingue per avere le caratteristiche elencate di seguito. A mio avviso, un corso di base dovrebbe illustrare approfonditamente almeno i primi quattro punti. Per quanto questi non siano concetti difficili, non è facile descriverli a parole senza avere degli esempi sotto mano. Il significato della seguente lista quindi si chiarirà durante il corso.

1. **Completa ricorsività in coda.** Questo significa che alcuni programmi scritti in stile ricorsivo generano processi iterativi guadagnando in efficienza. Tutto ciò sarà chiaro dopo i primi esempi di funzioni ricorsive che incontreremo nel corso con l'ausilio dello *stepper*.
2. **I campi delle variabili sono statici.** Questo significa che per ogni occorrenza di una variabile vincolata è possibile determinare staticamente (cioè analizzando il codice sorgente del programma senza bisogno di metterlo in esecuzione) l'espressione che la definisce. Tutto ciò sarà chiarito in maniera *visiva* con l'ausilio del *syntax checker* di DrScheme.
3. **Le procedure sono oggetti di prima classe.** Questo significa che le procedure sono valori che possono essere assegnati a variabili, passati come argomenti ad altre procedure o ritornati come valori da una procedura. (Qualcosa che ricorda quello che avviene in C con i puntatori a funzioni). Anche questo sarà chiaro dopo opportuni esempi.

4. **Gestione automatica della memoria.** Cioè: *"Mai più puntatori"* con le relative `malloc()` e `free()`, e con i soliti *memory leak* difficili da rintracciare.
5. **Le macro sono igieniche.**
6. **Le funzioni possono ritornare valori multipli**.
7. **Esistono le continuazioni come oggetti di prima classe**.

6 I difetti dello Scheme

Per certi aspetti, lo Scheme è un linguaggio che non ha imperfezioni. Semmai, lo Scheme ha qualche carenza. Soprattutto la mancanza di standard in alcuni aspetti per così dire periferici del linguaggio. Anche perché la comunità dei programmatori Scheme ha sempre rifiutato di adottare degli standard che non fossero completamente soddisfacenti (cfr. [Jargon] *The Right Thing* © MIT) In particolare:

- **La mancanza di librerie standard.** Esempi: di librerie standard di interfaccia con sistema operativo, filesystem e rete, dispositivi grafici e multimediali, oppure di interfaccia con database o, ancora, per elaborazione di testi, html/xml, matematica statistica e applicazioni scientifiche. Ovviamente sono state scritte molte librerie, spesso di buona qualità e facilmente portabili da una implementazione Scheme ad un'altra. Ma non sono comunque standard. Questo problema sembra in via di soluzione definitiva con il processo di standardizzazione SRFI.
- **La mancanza di un sistema di moduli standard** (soprattutto di un sistema per la *"partizione del namespace"* e per consentire la riusabilità del codice). Non è una grossa carenza in realtà, molti linguaggi di grande successo (un esempio per tutti: il C) non hanno nessun sistema di moduli. Vari sistemi di moduli sono stati proposti e implementati, anche sulla base dei modelli proposti in altri linguaggi di programmazione, ciascuno con i propri pregi e difetti. Il fatto è che il problema da risolvere è a livello teorico. Si vedono continui progressi nella ricerca in questo campo ma una soluzione definitiva non sembra ancora vicina.
- **La mancanza di un sistema di oggetti standard.** Anche la carenza di un sistema ad oggetti standard nello scheme è semplicemente il riflesso del fatto che a tutt'oggi non esiste una teoria unanimamente accettata a fondamento della programmazione orientata agli oggetti (anche perché moduli e oggetti sono in relazione tra di loro). Di fatto lo Scheme permette di implementare senza eccessiva difficoltà qualsiasi tipo di programmazione orientata agli oggetti[1]. Tuttavia un sistema ad oggetti che fosse standard semplificherebbe certamente il lavoro quotidiano.

Inoltre lo Scheme è ancora troppo poco utilizzato dall'industria, probabilmente proprio a causa della carenza di una completa (e rassicurante) standardizzazione. Tuttavia, versioni ridotte, specializzate e standardizzate dello Scheme sono state adottate con successo (fixme: esempi Java, dsssl, dylan).

7 Bibliografia

È possibile scaricare questa bibliografia in formato BibTeX.

[1] Hal Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press and McGraw-Hill, second edition 1996 edition, 1985.

[2] W. Clinger and I. Rees. Revised 4 report on the algorithmic language scheme. Technical report, Eugene (Oregon), 1992.

[3] R. Kent Dybvig. *The Scheme Programming Language*. Prentice Hall, second edition edition, 1996.

[4] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs*. MIT press, 2001. available in full on-line at *http://www.htdp.org/*.

[5] Daniel P. Friedman and Matthias Felleisen. *The Little Schemer*. MIT Press, fourth edition edition, 1995.

[1] Qualcuno dice che *"Gli oggetti sono la chiusura dei poveri"* intendendo dire che avere i campi statici e procedure di prima classe significa avere molto di più della programmazione ad oggetti

[6] Richard Kelsey, William Clinger, and Jonathan Rees (Editors). Revised^5 report on the algorithmic language Scheme. *ACM SIGPLAN Notices*, 33(9):26–76, September 1998.

[7] Dorai Sitaram. Teach yourself scheme in fixnum days. `http://www.cs.rice.edu/~dorai/t-y-scheme/`.

[8] George Springer and Daniel P. Friedman. *Scheme and the Art of Programming.* MIT Press/McGraw Hill, 1990.