Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Choreia: A Static Analyzer to generate Choreography Automata from Go source code

Relatore:
Chiar.mo Prof.
Ivan Lanese

Presentata da:
Enea Guidi

Sessione III
Anno Accademico 2020/2021

Agli amici che ho conosciuto e che
mi hanno accompagnato in questo viaggio

Sommario

Le Choreographies sono un paradigma emergente per la descrizione dei sistemi concorrenti. Lo scopo principale è quello di fornire al programmatore uno strumento il quale permetta di capire in maniera immediata la "coreografia" dei participanti nel sistema e come questi interagiscano tra loro. Partendo dai singoli partecipanti e loro Choreographies *locali* è possibile ricomporre in maniera bottom-up l'intera Choreography *globale* del sistema. Un ulteriore vantaggio delle Choreographies è che, se rispettano alcune proprietà ben definite, permettono di fare assunzioni sull'assenza di tipici problemi di concorrenza quali Deadlocks e Race Conditions. Esistono vari modelli formali di Choreographies, questa tesi tratta nello specifico i *Choreography Automata*, basati su *Finite State Automata* (FSA). In questa tesi viene presentato Choreia: un tool di analisi statica che, partendo da un codice sorgente Go, ricava il Choreography Automata del sisitema concorrente in maniera bottom-up.

Indice

Capitolo 1

Introduzione

Negli ultimi anni si è visto un interesse sempre più pronunciato verso lo sviluppo di sistemi concorrenti e distribuiti, questo si riflette nella progettazione a microservizi o in linguaggi di programmazione più recenti come Go, Rust o C++ 20 i quali implementano tutti, in forme disparate, un approccio alla concorrenza facilitato e univoco evitando la complessità e frammentazione che può derivare dall'utilizzo di librerie esterne e non standardizzate.
Prendendo per esempio Go, talvolta chiamato anche golang, un linguaggio di programmazione creato nel 2009 da Robert Griesemer, Rob Pike e Ken Thompson, che negli anni successivi prende una forte trazione, fino a essere supportato da Google, sopratutto nel settore del web-development e system programming. Alcuni dei fattori da successo sono una compilazione ed esecuzione veloce, una grammatica semplice e snella e un approccio *built-in* alla concorrenza con tipi e primitive fornite direttamente dalla standard library. Quest'ultima infatti fornisce un solo tipo di dato *chan* che può essere *buffered* o *unbuffered*, la comunicazione che avviene su questi canali è rispettivamente asincrona e sincrona.
Parallelamente a questo rinnovato interesse verso la programmazione concorrente e distribuita nasce l'esigenza di formalizzare dei modelli teorici che possano supportare la progettazione, gestione e manutenzione di sistemi concorrenti. Uno di questi modelli sono le *Choreographies*, le quali facilitano la rappresentazione e descrizione di sistemi concorrenti in cui i singoli *attori* comunicano tra di loro. Un aspetto distintivo delle Choreographies è la coesistenza di due *view*:

- Vista **globale**: descrive la coordinazione necessaria tra i componenti di un sistema
- Vista **locale**: descrive il comportamento di un singolo componente *in isolamento*

L'obiettivo di questa tesi è quello di progettare e implementare un tool che permetta, preso un sorgente Go e per mezzo di analisi statica, di estrarre un Choreography Automata, ovvero un automa a stati finiti che rappresenta la Choreography in modo grafico, più chiaro e immediato rispetto alla lettura del codice sorgente.

Capitolo 2

Nozioni preliminari e notazione

2.1 FSA non deterministici e deterministici

Prima di introdurre le Choreographies e i Choreographies Automata è necessario fare un breve richiamo di alcune nozioni fondamentali quali la nozione di Automa a Stati Finiti (FSA) e alcune trasformazioni possibili sullo stessso. Gli automi a stati finiti sono la descrizione di un sistema dinamico che si evolve nel tempo, esiste un parallelo tra gli automi e i calcolatori moderni, per esempio il flusso d'esecuzione di un programma può essere rappresentato attraverso un automa. Alcune applicazioni pratiche di questi automi possono essere regular expression (RegEx o RegExp), lexer e parser solo per citarne alcuni ma possono anche essere impiegati, come vedremo in questa tesi, anche nel campo della concorrenza.

Definition 2.1 (Finite State Automata) *Un automa a stati finiti (FSA) è una tupla* $A = \langle \mathcal{S}, s_0, \mathcal{L}, \delta \rangle$ *dove:*

- $\mathcal{S}$ *è un insieme finito di stati*
- $s_0 \in \mathcal{S}$ *è lo stato iniziale dell'automa*
- $\mathcal{L}$ *è l'alfabeto finito, talvolta detto anche insieme di label* ($\epsilon \notin \mathcal{L}$)
- $\delta \subseteq \mathcal{S} \times (L \cup \{\epsilon\}) \times \mathcal{S}$ *è la funzione di transizione* (ϵ *denota la stringa vuota)*

Tipicamente, oltre a quanto definito sopra, è solito trovare anche un insieme di stati di terminazione (o accettazione), detto $\mathcal{F}$. Visto che in questo caso detto insieme non è stato definito assumiamo che ogni $s \in \mathcal{S}$ sia uno stato di accettazione. Inoltre va notato che questa definizione coincide con quella di automa a stati finiti *non deterministico*, solitamente indicato in letteratura con la sigla NFA (*Non Deterministic Finite Automata*) e distinto dalla nozione di DFA (*Deterministic Finite Automata*).

Definition 2.2 (Deterministic Finite Automata) *Un automa a stati finiti deterministico è una tupla* $D = \langle \mathcal{S}, s_0, \mathcal{L}, \delta \rangle$ *con* $\delta \subseteq \mathcal{S} \times L \times \mathcal{S}$

Le varianti deterministiche si distinguono dalle loro controparti non deterministiche dal fatto che non ammettono l'utilizzo di ϵ transizioni e l'utilizzo più transizioni *uscenti* dallo stesso stato con la medesima etichetta. Sebbene queste due varianti siano tra loro equivalenti, l'utilizzo di una variante rispetto all'altra può essere determinato da fattori come necessità di una maggiore elasticità (gli NFA sono meno stringenti rispetto ai DFA) o, al contrario, di una migliore chiarezza (i DFA sono più immediati rispetto alla loro controparte).

In ogni caso è sempre possibile, dato un NFA qualunque, un DFA ad esso equivalente. L'algoritmo che permette di fare questa trasformazione fa uso estensivo di ϵ *closure* e della funione *mossa* che andiamo a definire di seguito:

Definition 2.3 (ϵ closure) *Fissato un NFA* $N = \langle \mathcal{S}, s_0, \mathcal{L}, \delta \rangle$ *ed uno stato* $s \in \mathcal{S}$ *si dice* ϵ *closure di s, indicata con* ϵ*-clos(s) il più piccolo* $\mathcal{R} \subseteq \mathcal{S}$ *tale che:*

- $s \in \epsilon$*-clos(s)*
- *se* $x \in \epsilon$*-clos(s) allora* $\delta(x, \epsilon) \subseteq \epsilon$*-clos(s)*

Remark 2.3.1 *Se* $\mathcal{X}$ *è un insieme di stati definiamo* ϵ*-clos(*$\mathcal{X}$*) come* $\bigcup_{x \in \mathcal{X}} \epsilon$*-clos(x).*

Definition 2.4 (Mossa) *Dato un insieme di stati* $\mathcal{X} \subseteq \mathcal{S}$ *e un simbolo* $\alpha \in \mathcal{L}$ *definiamo la funzione* mossa*:* $\mathcal{P}(\mathcal{S}) \times \mathcal{L} \longrightarrow \mathcal{P}(\mathcal{S})$ *tale che:* $\text{mossa}(\mathcal{X}, \alpha) = \bigcup_{x \in \mathcal{X}} (\delta(x, \alpha))$

L'algoritmo che permette di ricavare un DFA ad un dato NFA è il seguente:

Algorithm 1 Subset Construction Algorithm

```
x ← ϵ-clos(s_0)                          ▷ Lo stato iniziale del DFA
T ← {x}                                  ▷ Un insieme di ϵ-clos
while ∃ t ∈ T non marcato do
    marca(t)
    for each α ∈ L do
        r ← ϵ-clos(mossa(t, α))
        if r ∉ T then
            T ← T ∪ {r}
        end if
        δ(t, α) ← r          ▷ Denota che la δ del DFA con input t ed α da output r
    end for
end while
```

Capitolo 3

Adattazioni e modifiche

TODO

Capitolo 4

Tecnologie utilizzate

TODO

Capitolo 5

Descrizione del tool

TODO

Capitolo 6

Conclusioni e lavori futuri

TODO

Capitolo 7

Template

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis fringilla augue, cursus feugiat neque. Cras varius tortor ut cursus molestie. Quisque odio felis, sodales a tincidunt a, auctor vestibulum velit. Pellentesque non aliquam elit, vitae dictum augue. Suspendisse vulputate nec ligula et gravida. Etiam dui ligula, malesuada quis massa eget, feugiat rhoncus eros. Aenean at felis eu arcu sagittis molestie. Suspendisse dui elit, consectetur et porttitor quis, aliquam vel enim. Sed id justo sem. Nunc pretium, est non laoreet faucibus, sapien dolor laoreet diam, efficitur consequat justo urna sed ex. Nam blandit quam enim, ac varius purus vestibulum lacinia. Integer dignissim auctor ligula ac pharetra. Cras in urna ac enim consequat euismod in a nisi. Pellentesque imperdiet tellus magna, id sollicitudin risus pulvinar eu. Proin est libero, dictum non suscipit malesuada, mollis sit amet eros.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis fringilla augue, cursus feugiat neque. Cras varius tortor ut cursus molestie. Quisque odio felis, sodales a tincidunt a, auctor vestibulum velit. Pellentesque non aliquam elit, vitae dictum augue. Suspendisse vulputate nec ligula et gravida. Etiam dui ligula, malesuada quis massa eget, feugiat rhoncus eros. Aenean at felis eu arcu sagittis molestie. Suspendisse dui elit, consectetur et porttitor quis, aliquam vel enim. Sed id justo sem. Nunc pretium, est non laoreet faucibus, sapien dolor laoreet diam, efficitur consequat justo urna sed ex. Nam blandit quam enim, ac varius purus vestibulum lacinia. Integer dignissim auctor ligula ac pharetra. Cras in urna ac enim consequat euismod in a nisi. Pellentesque imperdiet tellus magna, id sollicitudin risus pulvinar eu. Proin est libero, dictum non suscipit malesuada, mollis sit amet eros.

7.1 Section

Vivamus eu mattis metus, vitae venenatis ante. In dictum augue pulvinar tellus rutrum molestie. Pellentesque augue tellus, bibendum in scelerisque sed, porttitor vel ipsum. Nunc at molestie quam, et lobortis eros. Ut congue, metus vel sodales varius, nibh dui vulputate dolor, nec cursus lectus nisi in ante. Etiam lacinia mi et finibus cursus. Nam blandit mi eget pharetra congue. Etiam ac semper lacus. Suspendisse sed sagittis arcu. Sed sit amet fringilla urna. Praesent urna turpis, imperdiet at congue nec, molestie quis lectus. Ut condimentum neque ut nisi scelerisque aliquet.

7.1.1 Subsection

Integer viverra varius mauris, at cursus ante maximus ac. Maecenas commodo ullamcorper ex, eget congue lorem cursus eget. Donec consequat eros purus, ac luctus felis congue ut. Proin ut urna malesuada, condimentum libero sed, pretium ligula. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut vel erat faucibus, egestas ipsum vitae, mattis mauris. Etiam vestibulum, nunc non ornare scelerisque, urna erat molestie lorem, et dapibus nisl purus nec ipsum. Phasellus ac tortor vitae lorem fringilla tristique vitae vel felis. In quis purus urna. Vestibulum id congue enim. Integer a convallis justo, a vehicula felis. Maecenas et tincidunt mi. Morbi et enim dignissim, consectetur diam vitae, iaculis diam. In hac habitasse platea dictumst.