Sistemas de Gerenciamento de Versão

Thiago Rafael Becker

Dezembro de 2001

Resumo

blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

Introdução

Os sistemas de gerenciamento de versão(*source control management, SCM*) são ferramentas largamente utilizadas em desenvolvimento de software e operações para controlar as mudanças em código fonte e configurações do sistema.

Estrutura e operação dos SCM

Um SCM gerencia estados de objetos. O conteúdo destes objetos é uma sequancia de dados pertencentes ao alfabeto $\Sigma_\perp = \Sigma \cup \perp$. Todos os estados de um objeto são definidos pelo conjnto gerado pelo fecho de Kleene sobre o alfabeto $\Sigma_\perp$, $\Sigma_\perp^\star = \langle \Sigma_\perp, \circ, \varepsilon \rangle$. $\perp$ é um elemento especial que indica indefinição das funções parciais sobre o conjunto $\Sigma_\perp^\star$.

Além de dados, um SCM gerencia metadados, como nomes apontados para objetos. A modificação de metadados não modifica o conteúdo de um estado, apenas o objeto para o qual o metadado aponta.

O conjunto de estados, transições e metadados são conhecidas como *repositório*.

O processo normal de uso de um SCM genérico envolve algumas operações básicas:

- *commit*: a partir das modificações feitas pelo usuário e do estado apontado pelo metadado `HEAD`, cria um novo estado e move o metadado `HEAD` para o novo estado. Pode também mover outros metadados para o novo objeto. Sem modificações, esta operação se comporta como NOP.
- *branch* `nome estado`: cria um novo metadado `nome` apontando para `estado`, movendo o metadado `HEAD` para `estado`. Essencialmente, esta operação cria uma linha de desnvolvimento paralela a linha de desenvolvimento de origem. Novos *commit*nesta linha de desenvolvimento são aplicados independentes da linha original.
- *merge* `estado-base estado-branch`: para que esta operação tenha efeito, uma operação de *branch*deve ter ocorrido anteriormente a partir de `ancestral-comum`, e commits devem ter sido feitos neste branch. A operação de merge cira um novo estado a partir de `estado-base` aplicando as modificações de $diff$:`ancestral-comum`→`estado-branch`. Esta operação pode gerar um conflito (indefinição) que deve ser resolvido manualmente pelo usuário.
- *tag* `nome estado`: Esta operação cria um metadado `nome` apontando para `estado`. É muitas vezes semelhante à operação *branch*, e em alguns SCM ambas são a mesma operação. Uma *tag*especial é a *tag*`HEAD`, que aponta o estado cujo objeto está recebendo modificações.

Adição e remoção de caracteres

Dois funcionais são definidos sobre $\Sigma^\star_\perp$: *adição de caracter* e *remoção de caracter*. A adição de caracter

$$\oplus^c_k : \Sigma^\star_\perp \to \Sigma^\star_\perp, \oplus^c_k(\perp) = \perp \tag{1}$$

insere o caracter c na posição k, e desloca todos os caracteres a direita de k uma posição à direita. A remoção de caractere

$$\ominus^c_k : \Sigma^\star_\perp \to \Sigma^\star_\perp, \ominus^c_k(\perp) = \perp \tag{2}$$

remove o caracter c da posição k, e move todos os caracteres à sua direita uma poisção para a esquerda. Além disso, a identidade $id : \Sigma^\star_\perp \to \Sigma^\star_\perp$ é definida.

As funções geradas pelos funcionais $\oplus^c_k$ e $\ominus^c_k$ ($\{ger\oplus^c_k\}$, $\{ger\ominus^c_k\}$) formam um grupo de transformação $T_{\Sigma^\star_\perp} = \langle \{ger\oplus^c_k\} \cup \{ger\ominus^c_k\}, \circ, id\rangle$. Para provarmos as propriedades do grupo, definimos o mapeamento $\Sigma^\star_\perp \to \mathbb{N} \cup \{\perp\}$

$$\Gamma = \{\perp_{\Sigma_\perp} \mapsto \perp_{\mathbb{N}\cup\perp}, \varepsilon_{\Sigma_\perp} \mapsto 0_{\mathbb{N}\cup\perp}, c_\Sigma \mapsto n_\mathbb{N}\} \tag{3}$$

tal que o mapeamento é uma bijeção. O mapeamento das operações $\oplus^c_k$ é

$$\Gamma(\oplus^c_k(\sigma)) = \Gamma(\sigma)p_k^{\Gamma(c)}, p_k \in \mathbb{P} \tag{4}$$

e $\ominus^c_k$ é

$$\Gamma(\ominus^c_k(\sigma)) = \Gamma(\sigma)p_k^{-\Gamma(c)}, p_k \in \mathbb{P}, \tag{5}$$

e caso exista $n > 0$ tal que $\Gamma(\ominus^c_k(\sigma))p_k^{-n} \in \mathbb{N}$, então $\ominus^c_k(\sigma) = \perp$. Além disso, se $\Gamma(\ominus^c_k(\sigma)) \notin \mathbb{N}$, então $\ominus^c_k(\sigma) = \perp$.

O mapeamento de $\Gamma(\sigma)$, $\sigma \in \Sigma^\star_\perp$ é tal que

$$\Gamma(\sigma) = \prod_{i\geq 1} p_i^{\Gamma(\sigma_i)}, p_i \in \mathbb{P}, \sigma_i \in \Sigma^\star_\perp \tag{6}$$

Com isso é possível verificar que $\Gamma(\oplus^c_k)$ e $\Gamma(\ominus^c_k)$ formam um grupo de transformação sobre $\mathbb{N}\cup\{\perp\}$, provando a existencia de $T_{\Sigma^\star_\perp}$.

Representação como grafo

Existe um funtor F que permite representar a categoria livremente gerada de $T_{\Sigma^\star_\perp}$ como um grafo, tal que

$$F(\Sigma^\star_\perp) = \sigma_1, \sigma_1 \subset \Sigma^\star_\perp \tag{7}$$

$$F(\{ger\oplus^c_k\} \cup \{ger\ominus^c_k\} \cup \{id\}) = f : \sigma \to \sigma\prime, \sigma, \sigma\prime \in \Sigma^\star_\perp \tag{8}$$

em que σ_1 é um conjunto unário. F constrói a estrutura do repositório, e é inversível. Para completar o conceito de um repositório, precisamos definir as operações executadas nele, que é o tema da próxima seção.

O grafo que representa um dado repositório é um subgrafo *Repo* do grafo direto reflexivo

$$\langle F\Sigma^\star_\perp, F(\{ger\oplus^c_k\} \cup \{ger\ominus^c_k\} \cup \{id\})\rangle,$$

e é fácil verificar as propriedades compositiva e associativa das setas $F(\{ger\oplus^c_k\} \cup \{ger\ominus^c_k\} \cup \{id\})$. Desta forma, a categoria `Repo`

$$\langle F(\Sigma^\star_\perp), F(\{ger\oplus^c_k\} \cup \{ger\ominus^c_k\} \cup \{id\}), \delta_0, \delta_1, id, \circ\rangle$$

é a categoria livremente gerada pelo grafo *Repo*.

Operações sobre `Repo`

Na seção , definimos intuitivamente as operações *commit*, *branch*, *merge*e *tag*. Nesta seção, vamos definir estas operações como homofuntores sobre `Repo`.