OSPF configure sytax && reduction algorithm

2023 年 12 月 28 日

1 OSPF Conf Sytax

我们定义 OSPF configure 的语法如下

$$
\begin{aligned}
CONF &::= CG_R; CONF | CG_I; CONF | \epsilon \\
CG_R &::= c_{RCtx}; CL_R | C_{RCtx} \\
CG_I &::= c_{ICtx}; CL_I | C_{ICtx} \\
CL_R &::= c_R; CG_R | \epsilon \\
CL_I &::= c_I; CG_I | \epsilon \\
C_{RCtx} &::= \textbf{ROSPF} \\
C_{ICtx} &::= \textbf{INTFN} \\
C_R &::= \textbf{RID} | ... \\
C_I &::= \textbf{IPOSPF} | ...
\end{aligned}
$$

另外定义一些项

$$
\begin{aligned}
C &::= C_{ctx} | C_{norm} \\
C_{ctx} &::= C_{RCtx} | C_{ICtx} \\
C_{norm} &::= C_R | C_I
\end{aligned}
$$

2 Reduction Algorithm

2.1 指令项的定义

(参数上下文) $\sigma_{args} \in Arg \longrightarrow Values$

σ_{args} 是指令参数到具体值的函数

(实例化指令) $c ::= C[\sigma_{args}]$

c 是一条实例化指令，它通过将指令模板 C 中的参数替换为具体的值得到。

(带上下文的实例化指令) $\mathbf{c} ::= \tau : c^l$

(上下文指令) $\tau ::= \mathbf{c}_{ctx} | \epsilon$

τ 是上下文指令类型的带上下文的实例

(指令位置) l

l 是该指令在 Conf 中的行号

2.2 上下文指令生成规则

规则用自然语言描述如下：

1. $\mathbf{c}_{ctx}$: ctxOp 是它自己

2. $\mathbf{c}_R$：如果前一条指令是 $\mathbf{c}_{RCtx}$，那么 ctxOp 是它，否则如果前条指令属于 C_R，ctxOp 是它的 ctxOp，否则 ctxOp 是 ϵ

3. $\mathbf{c}_I$: 如果前一条指令是 $\mathbf{c}_{ICtx}$, 那么 ctxOp 是它，否则如果前条指令属于 C_I, ctxOp 是它的 ctxOp，否则 ctxOp 是 ϵ

用形式语义书写，我们一共可以得到 9 条 judgment 规则 (1 1 条，2、3 各 4 条)

2.3 Reduction 自动机的定义

我们为每条 input 的指令建立 reduction 自动机

(状态集合)Q ::= **init**|**submitted**|**active**|**removed**

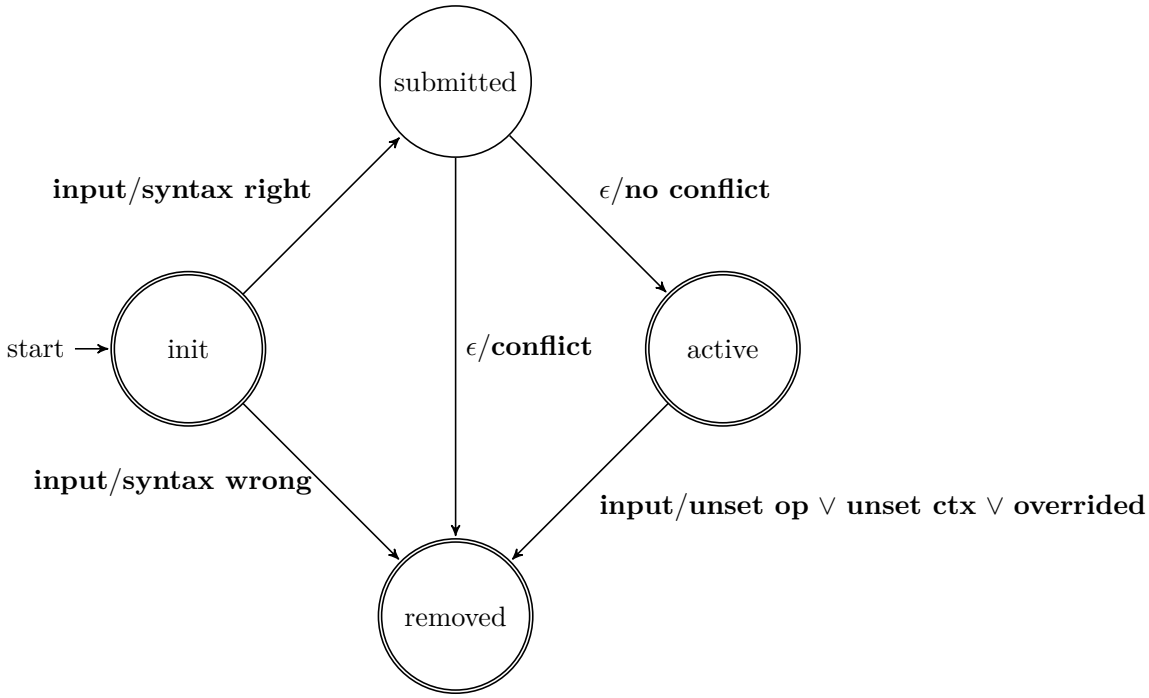
(输入)Σ ::= **input**|ϵ

(转移条件)P ::= **syntax right**|**syntax wrong**
|**conflict**|**no conflict**
|**unset op**|**unset ctxop**
|**override**

(初始状态)q_0 ::= **init**

(接受状态集合)F ::= {**init, active, removed**}

(状态转移函数)δ



当我们 input 一个指令的时候，该指令可能会改变之前指令的状态 (使得之前指令的转移条件满足)，该指令的状态转移依赖于之前的指令（转移条件中依赖之前指令的状态)。

2.4 转移条件的具体定义

TODO

2.5 Reduction 自动机的性质

从 reduction 自动机的转移函数中，我们可以推出以下 3 个非常重要的性质。

设新指令为 $\mathbf{c}_{new}$, 之前指令为$\mathbf{c}_{old}$, $\mathbf{c}_{new}$ 转移依赖的之前指令为 $\{\mathbf{c}_{dep}\}$, $\mathbf{c}_{new}$ 导致之前指令的指令状态发生改变 $\{\mathbf{c}_{change}\}$，这条指令加入后全部状态发生改变的指令 $\{\mathbf{c}_{change}\}_{total}$

1. 新指令不会改变其状态转移依赖的指令, 即 $\{\mathbf{c}_{dep}\} \cap \{\mathbf{c}_{change}\} = \emptyset$

2. 之前被改变的指令不会造成新的之前的指令被改变,即 $\{\mathbf{c}_{change}\}_{total} = \{\mathbf{c}_{change}\} \cup \mathbf{c}_{new}$

3. 一条指令只会在它为最新指令时改变状态或者被当前最新指令改变状态，这两种情况每种最多造成一次状态改变。

证明略，可以对根据自动机的结构对转移条件做归纳证明

2.6 Reduction 算法

Algorithm 1 Reduction algorithm

Require: OSPF configure **conf**

Ensure: Normal form of configure $\mathbf{conf}_{norm}$

Initialize $\sigma \leftarrow \{\mathbf{c} \rightsquigarrow \mathbf{INIT} | \mathbf{c} \in \text{conf}\}$ ▷ σ stores state of each command
Initialize $\mathbf{conf}_{norm}$ to empty
for each command c^l in conf **do**
    $\sigma \leftarrow \text{move}(conf[:l-1], c^l, \sigma)$ ▷ Update σ with new states
end for
for each command c^l in conf **do**
    if $\sigma[c^l] = \textbf{active}$ **then**
        add c^l to $\mathbf{conf}_{norm}$
    end if
end for