关于 Maple Algebra 的这一路

枫聆 (maplegra)

2022 年 7 月 30 日

目录

Language

language equation and Arden's Rule

Theorem 1.1. The set $A^* \cdot B$ is the smallest language that is a solution for X in the linear equation

$$X = A \cdot X + B$$

where X, A, B are sets of string and $+$ stands for union of languages. Moreover, If the set A does not contain the empty word, then the solution is unique.

Annotation 1.2. Arden's rule can be used to help convert some finite automatons to regular expressions.

Equivalence of Program

Graph Ismorphism

Accepted Language Equivalentce

Annotation 2.1. [4] Chapter 1.

Bisimulation and Observation Equivalence

Definition 2.2. A labelled transition system (LTS) is a tuple $(S, \Lambda, \rightarrow)$ where S is set of states, Λ is set of labels, and $\rightarrow$ is relation of labelled transitions (i.e., a subset of $S \times \Lambda \times S$). A $(p, \alpha, q) \in \rightarrow$ is written as $p \xrightarrow{\alpha} q$.

Annotation 2.3. TODO: categorical semantics: F-coalgebra

Definition 2.4. [1]Let $T = (S, \Lambda, \rightarrow)$ be a labelled transition system. The set of traces $Tr(s)$, for $s \in S$ is the minimal set satisfying

- $\varepsilon \in Tr(s)$.
- $\alpha\ \sigma \in Tr(s)$ if $\{\, s' \in S \mid s \ \xrightarrow{\alpha}\ s' \text{ and } \sigma \in Tr(s') \,\}$.

Definition 2.5. Two states p, q are trace equivalent iff $Tr(p) = Tr(q)$.

Definition 2.6. (Simultation) Given two labelled transition system $(S_1, \Lambda, \rightarrow_1)$ and $(S_2, \Lambda, \rightarrow_2)$, relation $R \subseteq S_1 \times S_2$ is a simulation iff, for all $(p, q) \in R$ and $\alpha \in \lambda$ satisfies

$$\text{for any } p \xrightarrow{\alpha}_1 p', \text{ then there exists } q' \text{ such that } q \xrightarrow{\alpha}_2 q' \text{ and } (p', q') \in R$$

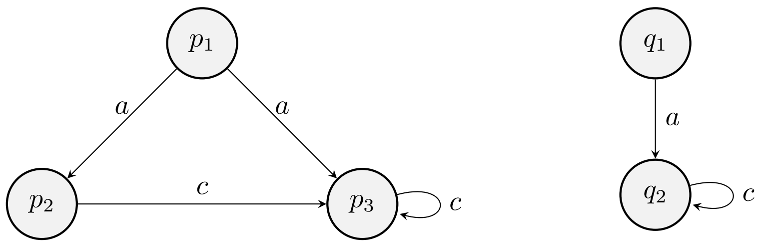
$$\begin{tikzcd}
p \arrow[r, "\alpha"] \arrow[d, dashed, "R"'] & p' \arrow[d, dashed, "R"] \\
q \arrow[r, "\alpha"'] & q'
\end{tikzcd}$$

Definition 2.7. We say q simulates p if there exists a simulation R includes (p, q) (i.e., $(p, q) \in R$), written $p < q$.

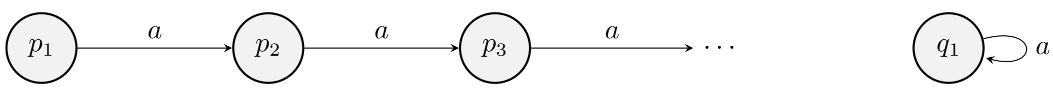
Definition 2.8. (Bisimultation) Given two labelled transition system $(S_1, \Lambda, \to_1)$ and $(S_2, \Lambda, \to_2)$, relation $R \subseteq S_1 \times S_2$ is a bisimulation iff both R and its converse $\overline{R}$ are simulations, for all $(p, q) \in R$ and $\alpha \in \Lambda$ satisfies

$$\text{for any } p \xrightarrow{\alpha}_1 p', \text{ then there exists } q' \text{ such that } q \xrightarrow{\alpha}_2 q' \text{ and } (p', q') \in R$$
$$\text{for any } q \xrightarrow{\alpha}_2 q', \text{ then there exists } p' \text{ such that } p \xrightarrow{\alpha}_1 p' \text{ and } (p', q') \in R$$

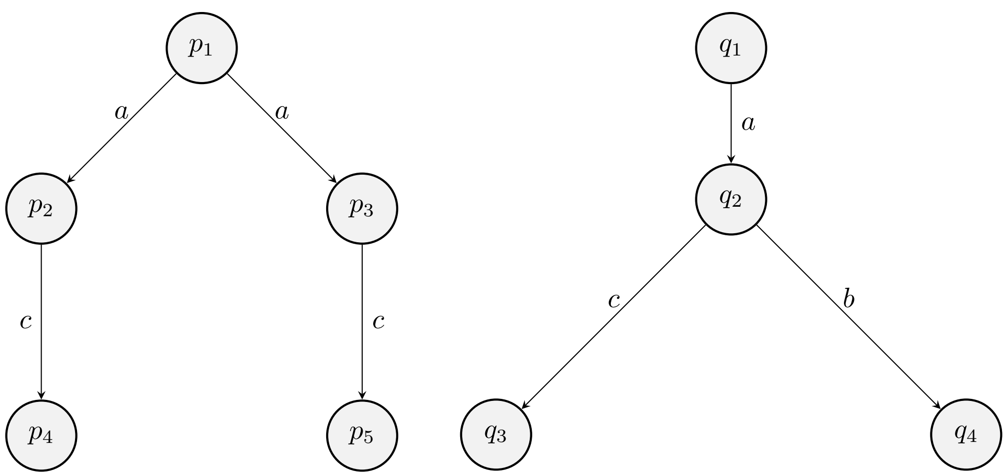
Example 2.9. 一些 bisimulation 的例子



关于上面两个 transition system 的 bisimultaion 为 $R = \{(p_1, q_1), (p_2, q_2), (p_3, q_2)\}$. 还有一个比较有点特别的例子



如果关于上图这样 bisimulation R 存在，那么 $(p_i, q_1) \in R$ for every i. 再看一个不是 bisimulation 的例子



这里不满足 $(p_3, q_2) \notin R$.

Definition 2.10. (Bisimilarity) Given two states p and q in S, p is bisimilar to q, written $p \sim q$, if and only if there is a bisimulation R such that $(p, q) \in R$.

Definition 2.11. The bisimilarity relation $\sim$ is the union of all bisimulations.

Lemma 2.12. The bisimulation has some properties:

- The identity relation *id* is a bisimulation(with two same LTS).
- The empty relation $\perp$ is a bisimulation.
- (closed under union) The $\bigcup_{i\in I} R_i$ of a family of bisimulations $(R_i)_{i\in I}$ is a bisimulation.

Lemma 2.13. [2] The bisimilarity relation $\sim$ is equivalence relation (i.e., reflexivity, symmetry, transitivity).

证明. 其中 reflexivity, symmetry 是比较显然的. Transitivity 稍微麻烦一点, 我们用 relation composition 定义新的 relation$R_3 = R_1; R_2$, 此时有 $(p,q) \in R_3$, 因此只要证明 R_3 is bisimulation 足够了. 取任意一个 $(p_1, q_1) \in R_3$, 那么按照 R_3 的定义, 存在 $(p_1, r_1) \in R_1$ 和 (r_1, q_1). 由 $p_1 \sim r_1$ 那么对于任意的 $p_1 \xrightarrow{\alpha} p_1'$, 存在 $r_1 \xrightarrow{\alpha} r_1'$ 满足 $(p_1', r_1') \in R_1$. 再由 $r_1 \sim q_1$, 存在 $q_1 \xrightarrow{\alpha} q_1'$ 满足 $(r_1', q_1') \in R_2$. 于是按照 R_3 的定义也有 $(p_1', q_1') \in R_3$. 再由 R_2 is bisimulation, 从 $(r_1, q_1) \in R_2$ 按照上述的思路往回证明即可, 最终 R_3 is bisimulation. □

Definition 2.14. [3] An LTS is called deterministic if for every state p and action α, there is at most one state q such that $p \xrightarrow{\alpha} q$.

Lemma 2.15. In a deterministic LTS, two states are bisimilar if and only if they are trace equivalent,

$$s_1 \sim s_2 \iff Tr(s_1) = Tr(s_2)$$

证明. 先证 $\Rightarrow$, 设满足 $s_1 \sim s_2((s_1, s_2) \in R$ and R is bisimultaion), 设 $\sigma_{s_1} \in Tr(s_1)$, 其中 σ_{s_1} 为 sequence $(\alpha_i)_{i\in I}$ where I is a indexed famliy. 由于 $s_1 \sim s_2$, 那么对于 $s_1 \xrightarrow{\alpha_1} s_1'$, 存在 $s_2 \xrightarrow{\alpha_1} s_2'$, 于是 $(s_1', s_2') \in R$, 根据 σ 长度做 induction 可以证明 $\sigma_{s_1} \in Tr(q)$. 再反过来证明 $\sigma_{s_2} \in Tr(s_2)$ 也同样有 $\sigma_{s_2} \in Tr(s_1)$. 最终 $Tr(s_1) = Tr(s_2)$.

对于 $\Leftarrow$, 我们可以用 $Tr(s_1) = Tr(s_2)$ 构造一个 bisimulation, 定义 relation R 为

$$Tr(s_1) = Tr(s_2) \iff (s_1, s_2) \in R.$$

只要能证明 R bisimulation 即可. 首先我们来说明在 deterministic 限制下一个比较好性质: 若 $Tr(s_1) = Tr(s_2)$ 且当 $s_1 \xrightarrow{\alpha} s_1', s_2 \xrightarrow{\alpha} s_2'$, 那么 $Tr(s_1') = Tr(s_2')$. 这样对于任意地 $(s_1, s_2) \in R$, 它们 accept 相同 action 对应的 transition $(s_1', s_2') \in R$. 因此 $s_1 \sim s_2$. □

Definition 2.16. (Weak Bisimultation) Given two labelled transition system $(S_1, \Lambda, \rightarrow_1)$ and $(S_2, \Lambda, \rightarrow_2)$, relation $R \subseteq S_1 \times S_2$ is a bisimulation iff both R and its converse $\overline{R}$ are simulations, for all $(p,q) \in R$ and $\alpha \in \Lambda \cup \{\tau\}$ satisfies

$$\text{for any } p \xrightarrow{\alpha}_1 p', \text{ then there exists } q' \text{ such that } q \xrightarrow{\tau *\ \alpha\ \tau *}{}^*_2\, q' \text{ and } (p', q') \in R$$
$$\text{for any } q \xrightarrow{\alpha}_2 q', \text{ then there exists } p' \text{ such that } p \xrightarrow{\tau *\ \alpha\ \tau *}{}^*_1\, p' \text{ and } (p', q') \in R$$

where $\rightarrow^*$ is multi-transition.

Annotation 2.17. 对于 LTS 的一些想法:

- 如果你想用 transition system 来做 reasoning 可以考虑把它和 Kripke frame 联系起来，同时要构造一些 modality 来设计方便做 reasoning 的 calculus.
- (*bisimulation proof method*) 对于两个特别的 states 来说，我们应该如何找到这样 bisimulation 来满足 $(p, q) \in R$?
- 对于两个特别的 LTS 来说，我们怎样以 bisimulation 思考它们是否 equivalent? bisimulation 的最初定义应该叫做 strong bisimulation, 它建立的是一种 strong equivalence, 而 weak bisimulation 建立是一种 observation equivalence.

Annotation 2.18. TODO: CCS(calculus of communicating systems)[4] and mCRL2 [3].

Symbolic Execution

Some Reasoning

Example 3.1. Symbolic reachability analysis 这是来自 [5] 的一个小例子, 我们尝试用 bounded model checking 来做一些 reasoning, 也就是 unfolding loop.

```c
#define VALVE_KO(status) status == -1
#define TOLERANCE 2
extern int size;
extern int valvesStatus[];

int getStatusOfValve(int i){
    if(i < 0 || i >= size){
        printf ("ERROR");
        exit(EXIT_FAILURE);
    }
    int status = valvesStatus[i];
    return status;
}

int checkValves(int wait1, int wait2) {
    int count, i;
    while(wait1 > 0) wait1--;
    count = 0, i = 0;
    while(i < size){
        int status = getStatusOfValve(i);

        if(VALVE_KO(status)) {
            count++;
        }
        i++;
    }

    if(count > TOLERANCE)
        printf ("ALARM");
    }
    while(wait2 > 0) wait2--;
        return count;
```

[5] 提到了一个 symbolic reachability analysis, 它和我们常见的 symbolic execution 是不一样的, 它可以看做给定一个 postcondition 沿着 control flow 往后推. 例如我们想进入 L29 所在的 branch, 那么 one-step induction 如下:

```
// P_28 = count > 2
{L28: count > 2}
// Q_28 = true
```

可以看到 precondition 是 weakest 的, 后面推导依然保持这个性质. 继续往后推导我们需要尝试得 resolve 掉 L19-L26 的 while, 这里可能就有 infinitely many paths, 例如执行 $0, 1, 2, \cdots$ 次这个 loop. 顺着这个思路来选择路径往后做 symbolic execution, 路径直到 function entry 为结束.

```
//Path_1: L15-> L16 -> L17 -> L18 -> L19 -> L28

// P_18 = count > 2 ∧ i ≥ size ∧ 0 > 2 ∧ 0 > size ≡ false
{L18: count = 0, i = 0; }
// Q_18 = count > 2 ∧ i ≥ size
// P_19 = count > 2 ∧ i ≥ size
{L19: i >= size}
// Q_19 = count > 2
// P_28 = count > 2
{L28: count > 2}
// Q_28 = true
```

在 P_{18} 这里得到了一个 contradiction, 这就意味着上面选择的 path 是 infeasible 的，那么到这里我们就不能往后再继续推理了. 现在我们给用 $Ln_a, Ln_b, \cdots$ 的形式来表示对同一 statement Ln 的多次执行.

```
//Path_2: ··· -> L19_b -> L20_b -> L22_b -> L23_b -> L25_b -> L19_a -> L28

...
// P_19b = count > 1 ∧ i = size − 1 ∧ i ≥ 0 ∧ valvesStatus[i] = −1 ∧ i < size
{L19b: i < size}
// Q_20b = count > 1 ∧ i = size − 1 ∧ i ≥ 0 ∧ valvesStatus[i] = −1
// P_20b = (count > 1 ∧ i ≥ size − 1 ∧ i ≥ 0 ∧ i < size ∧ valvesStatus[i] = −1) ≡
// (count > 1 ∧ i = size − 1 ∧ i ≥ 0 ∧ valvesStatus[i] = −1)
{L20b: int status = getStatusOfValve(i);}
// Q_20b = count > 1 ∧ i ≥ size − 1 ∧ status = −1
// P_22b = count > 1 ∧ i ≥ size − 1 ∧ status = −1
{L22b: status == -1}
// Q_22b = count > 1 ∧ i ≥ size − 1
```

```
// P_{23b} = (count + 1 > 2 ∧ i ≥ size − 1) ≡ (count > 1 ∧ i ≥ size − 1)
{L23b: count++}
// Q_{23b} = count > 2 ∧ i ≥ size − 1
// P_{25b} = (count > 2 ∧ i + 1 ≥ size) ≡ (count > 2 ∧ i ≥ size − 1)
{L25b: i++; }
// Q_{25b} = count > 2 ∧ i ≥ size
// P_{19a} = count > 2 ∧ i ≥ size
{L19a: i >= size}
// Q_{19a} = count > 2
// P_{28} = count > 2
{L28: count > 2}
// Q_{28} = true
```

上面就是执行了最后一次循环并且在这次循环中进入了 L23 所在的 branch，主要需要注意一下 P_{20b} 这里设计到了 inter-analysis.

参考文献

[1] Introduction to labelled transition systems.
http://wiki.di.uminho.pt/twiki/pub/Education/MFES1617/AC/AC1617-2-LTS.pdf

[2] An Introduction to Bisimulation and Coinduction.
https://homes.cs.washington.edu/~djg/msr_russia2012/sangiorgi.pdf

[3] Labelled transition systems.
https://www.mcrl2.org/web/user_manual/articles/lts.html

[4] A Calculus of Communicating Systems. Robin Milner.

[5] Baluda, Mauro, Giovanni Denaro, and Mauro Pezzè. ”Bidirectional symbolic analysis for effective branch testing.” IEEE Transactions on Software Engineering 42.5 (2015): 403-426