Dynamické struktury a chování programu

Aleš Smrčka

smrcka@fit.vut.cz

https://www.fit.vut.cz/person/smrcka/

Brno, 2024-12-05

- **Variantní typ** (variantní záznam, tagged union) je datový typ, který uchovává hodnotu předem neurčeného datového typu.
- Datový typ se vybírá z pevně stanovené množiny datových typů **v době přiřazení**.
- Podpora je nativní v různých programovacích jazycích např. Python, JavaScript, Perl, shell.
- Použití typicky pro kolekce (tj. dynamické datové struktury s proměnným počtem položek), např. pole/seznam/strom položek, kde předem neznáme, jakého typu položka bude.

Variantní typ v C

```
enum var_type { VT_NONE, VT_CHAR, VT_INT, VT_DOUBLE, VT_POINTER };
typedef struct var {
  enum var_type tag;  // název buď tag (od toho Tagged type), type nebo kind

  char c;
  int i;
  double d;
  void *p;

} Var;
```

skutečnou hodnotu nese pouze jedna z položek

Deklarace typu

```
void set_int(Var *v, int value)
{
  v->tag = VT_INT;
  v->i = value;
}
```

```
void set_pointer(Var *v, void *value)
{
  v->tag = VT_POINTER;
  v->p = value;
}
```

Definice operací

```
void foo() {
  Var array[10];
  set_int(&v[0], 42);
  set_pointer(&v[1], “Hello”);
}
```

Použití

Variantní typ v C, efektivněji pomocí union

```
enum var_type { VT_NONE, VT_CHAR, VT_INT, VT_DOUBLE, VT_POINTER };
typedef struct var {
  enum var_type tag;  // název buď tag (od toho Tagged type), type nebo kind
  union {             // následující položky budou sdílet místo v paměti (tj. budou se překrývat)
    char c;
    int i;
    double d;
    void *p;
  };
} Var;                // sizeof(Var) = sizeof(tag) + max(sizeof(c),sizeof(i),sizeof(d),sizeof(p))
```

položky se v paměti překrývají, šetří se místo

Deklarace typu

```
void set_int(Var *v, int value)
{
  v->tag = VT_INT;
  v->i = value;
}
```

```
void set_pointer(Var *v, void *value)
{
  v->tag = VT_POINTER;
  v->p = value;
}
```

Definice operací

```
void foo() {
  Var array[10];
  set_int(&v[0], 42);
  set_pointer(&v[1], “Hello”);
}
```

Použití

- Variantní typ
- Escape sekvence
- Stavový automat

Escape sekvence

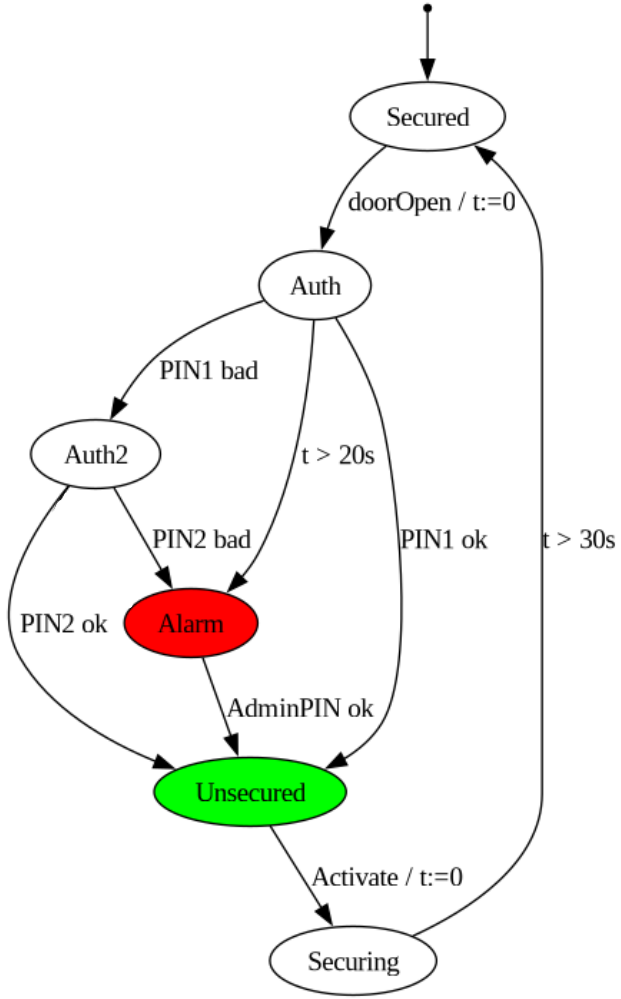
- Escape sekvence jsou znaky, které mají v řetězcích jiný význam než doslovnou reprezentaci znaků.
- Příklady:
  - “Hello, world**\n**” (C)
  - “The file is ****”output.txt****”.” (C)
  - “The word **“”**output**””**” (CSV)
  - <template>Dear **<**customerName**>**, ...</template>
  - znak ESC (ASCII 27_{10}, $1b_{16}$, 33_{8}): **\033**[31m (nastavení červené barvy v terminálu; \0XX značí bajt s ordinální hodnotou XX v osmičkové soustavě)
- Speciálním případem je tzv. quoting, tj. ohraničení řetězce se speciálními symboly uvozovkami (double quotes) nebo apostrofy (single quotes). Typicky se tímto zbavujeme významu oddělovačů, např. mezer, čárek nebo středníků (na příkazové řádce nebo ve strukturovaných textových souborech CSV, YAML).

- Variantní typ
- Escape sekvence
- Stavový automat

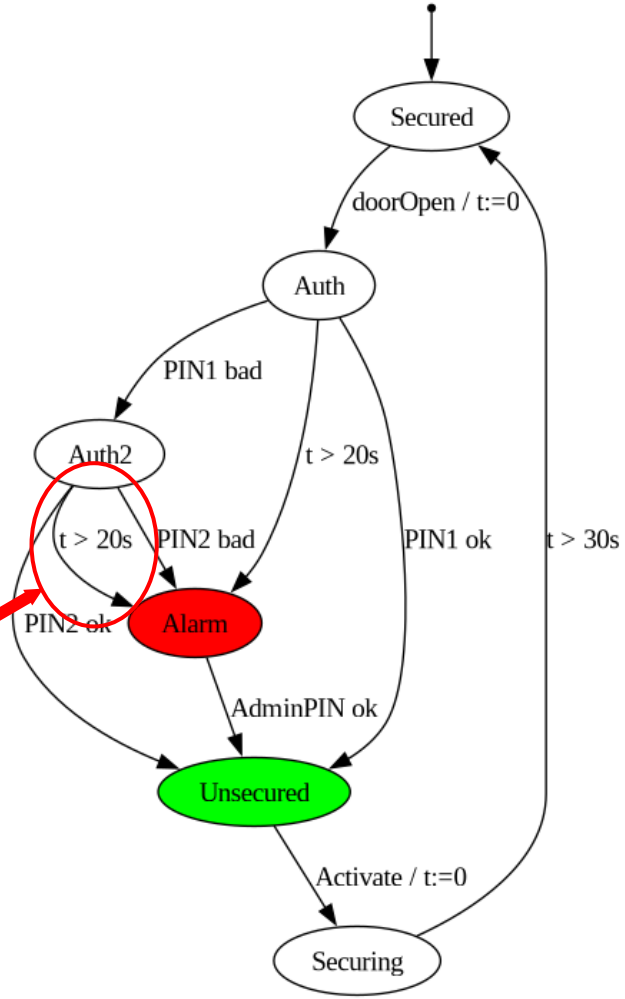
- **Stavový automat** (konečně stavový automat, **finite state machine**, **FSM**) je model výpočetního/řídícího systému, který se skládá z konečné množiny stavů a přechodů mezi nimi.
- Jedna ze základních formálních definic: $M = (Q, \Sigma, \delta, q0, F)$, kde
  - Q je konečná množina stavů,
  - Σ je konečná množina vstupních symbolů (abeceda),
  - $\delta : Q \times \Sigma \rightarrow Q$ je přechodová funkce určující přechod ze stavu do stavu přes vstupní symbol abecedy,
  - $q0 \in Q$ je počáteční stav,
  - $F \subseteq Q$ je koncový stav.

- Obecněji (neformálně) je stavový automat složen z:
  - množiny stavů,
  - množiny možných vstupů (symbolů),
  - počáteční stav,
  - přechodové relace,
  - množiny akcí.
- Automat má právě jeden stav aktivní. Říkáme: "**Automat se nachází ve stavu** XY".
- Automat postupně **přijímá vstupy** a **přechází mezi stavy** na základě daných přechodů.
  - Posloupnost symbolů na vstupu může být nekonečná.
  - Symbol může reprezentovat událost (např. window_closed, serviceFailed).
  - Symbol může být parametrický, tj. může nést dodatečnou hodnotu (např. userAuthenticated("john")).
  - Ve zvláštních případech symbol může být popsán predikátem (pravdivostním výrazem). Ten je potom nazýván stráž/guard (např. vehicle_speed > 90).

- Velkou výhodou stavových automatů je jejich **grafová reprezentace**:
  - uzly reprezentují stavy,
  - orientované hrany reprezentují přechody,
  - hrany jsou označené (popsané) vstupním symbolem, příp. i akcí prováděnou při aplikaci přechodu mezi stavy.
- Grafová reprezentace může rychle odhalit chyby (chybějící nebo nechtěné přechody nebo akce).



- Velkou výhodou stavových automatů je jejich **grafová reprezentace**:
  - uzly reprezentují stavy,
  - orientované hrany reprezentují přechody,
  - hrany jsou označené (popsané) vstupním symbolem, příp. i akcí prováděnou při aplikaci přechodu mezi stavy.
- Grafová reprezentace může rychle odhalit chyby (chybějící nebo nechtěné přechody nebo akce).



- Dle jednoznačnosti vstupu: deterministické x nedeterministické
  - Deterministický automat = z každého stavu existuje maximálně jeden proveditelný přechod. Jednoduché pro implementaci.
  - Nedeterministický: může být jednodušší na tvorbu, ovšem neaplikovatelné pro implementaci. Je nutno provést tzv. determinizaci (viz učivo formálních jazyků a teoterické informatiky).
- Dle výstupu:
  - Konečný výstup:
    - pravdivostní = pokud je automat po přijetí vstupu v koncovém stavu,
    - hodnotou = ve kterém z konečných stavů se nachází nebo jaké je ohodnocení pomocných proměnných,
  - Průběžný výstup

- Validace – automaty kontrolují platnost vstupu
  - vstup: zřetězená data
  - výstup: pravdivostní (řetězec přijat = koncový stav, nepřijat)
  - Příklad: kontrola formátu emailové adresy
- Transformace – transformují jeden vstup na jiný
  - vstup: zřetězená data
  - výstup: zřetězená data jiného formátu
  - Příklad: kodér Morseovy abecedy
- Řízení – slouží pro ovládání konečně stavového stroje
  - vstup: události systému (aktivace sensorů/tlačítek, vypršení času, ...)
  - výstup: aktivace aktualizace pomocných proměnných, řídící signály pro pomocné jednotky (světla, aktuátory, ...)
  - Příklad: řídící jednotka výtahu, semaforu.

- Základem imlementace je jedna proměnná pro označení aktuálního stavu a funkce implementující přechodovou relaci (provedení přechodů) a hlavní smyčkou:

```
int state = INIT_STATE;
int input_symbol;

while (! end_of_input()) {
  input_symbol = read_input();
  state = transition(state, input_symbol);
}
```

- Stavy je vhodné pojmenovat (pomocí enum).
- Komplikované vstupy (např. slova) je vhodné transformovat na symboly vstupní abecedy (ve funkci read_input).

- Přechodová funkce je obvykle pomocí řídicí struktury switch-case. V případě velkých automatů pomocí přechodové tabulky (nad rámec IZP).

```
State transition(State state, Input input) {
    switch (state) {
        case q0:
            if (input == ZERO)
              return q0;
            else
              return q1;
        case q1:
            if (input == ZERO)
              return q2;
            else
              return q1;
        ...
    }
}
```

Před skončením funkce případně i s doprovodným výstupem (nebo provedením akce)

- Vytvořte automat, který rozpoznává binární řetězce končící na “01”.
- Vytvořte automat, který transformuje písmena angl. abecedy na Morseovu abecedu.
- Vytvořte automat, který rozpozná, jestli je na vstupu celé číslo (začíná volitelným znaménkem, pokračuje posloupností 0-9) nebo desetinné číslo (volitelné znaménko, celá část čísla, desetinná část čísla), kde je číslo ukončené bílým znakem nebo koncem řetězce. Automat musí detekovat, jestli vstup žádnému číslu neodpovídá.
- Mějme pole řetězců a formátovací řetězce. Vytvořte automat generující řetězec podle formátovacího řetězce, ve kterém nahradí poziční reference za řetězce v poli. Např.

  pole = [“Freddy Krueger”, “1428 Elm St, Los Angeles”, “1x steak glove”]
  řetězec = ”Dear $0, we will send you the order of $2 to address $1.”
- Vytvořte stavový automat, který ze zdrojového souboru extrahuje všechny komentáře. Pozn. pozor na: `char *str = “This is /* not a comment”;`