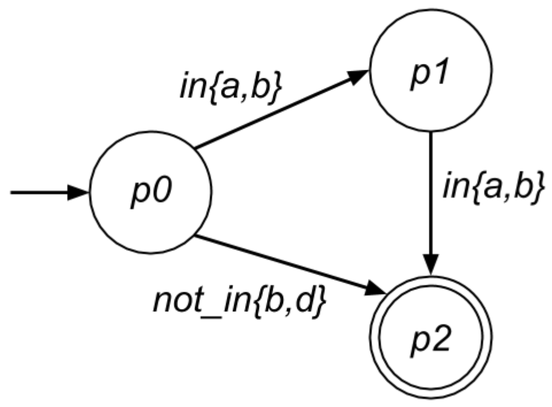
Excel@FIT 2017

Michaela Bieliková
xbieli06@stud.fit.vutbr.cz

8 Knižnica pre symbolické automaty určená na rýchle prototypovanie

Symbolické automaty

Symbolické automaty sú rozšírením konečných automatov. Namiesto symbolov používajú v prechodoch predikáty, ktoré môžu predstavovať množiny symbolov. Zápis automatu je takto stručnejší a jednoduchší.



Väčšina operácií definovaná nad konečnými automatmi sa dá po úprave použiť aj na symbolické automaty. Táto úprava zvyčajne zahŕňa nahradenie rovnosti symbolov za konjunkciu alebo disjunkciu predikátov.

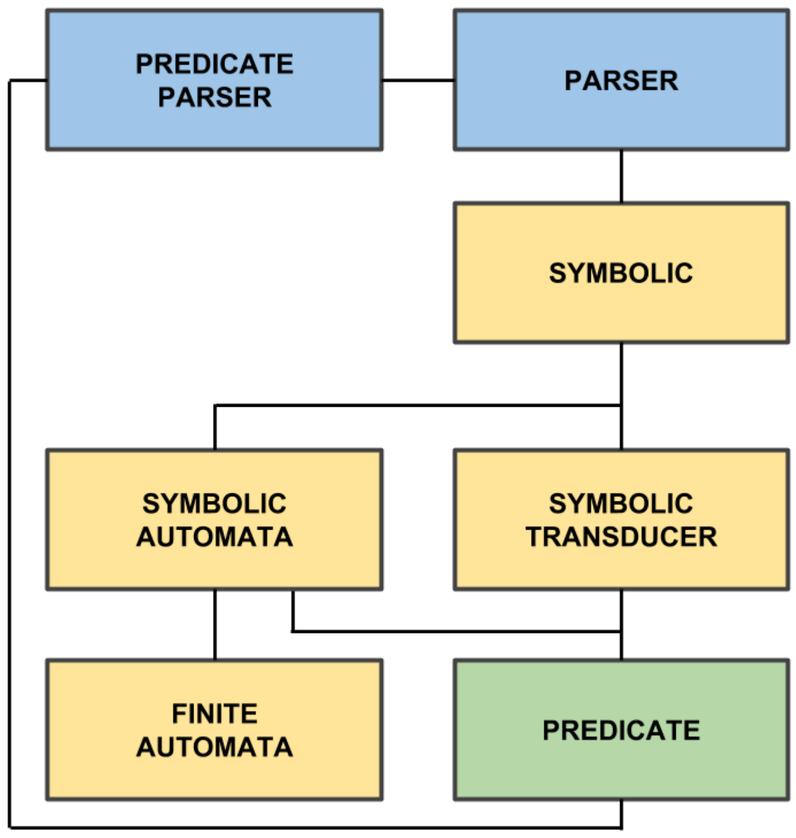
Motivácia

Symbolické automaty vedú k výraznému zmenšeniu počtu prechodov ak je automat definovaný na veľkej abecede. Príklady sa dajú nájsť v modelovaní systémov alebo v spracovaní prirodzeného jazyka, kde abecedou môžu byť aj celé slová.

V súčasnosti existujú rôzne knižnice zaoberajúce sa symbolickými automatmi. Komplexné a optimalizované knižnice ako VATA (C++), AutomataDotNet (C#), symbolicautomata (Java) sú veľmi efektívne a poskytujú rôzne pokročilé automatové operácie. Nevýhodou je, že implementácia v nich je zložitá a používajú dátové štruktúry zamerané na optimalizáciu, takže nie sú použiteľné na rýchle prototypovanie bez nutnosti naštudovať vnútornú štruktúru knižnice.

Cieľom tejto práce bolo vytoriť knižnicu, vhodnú na rýchle prototypovanie algoritmov, ktorá by zároveň implementovala pokročilé metódy manipulácie s automatmi. Knižnica sa má zamerať na jednoduchosť a intuitívnosť používania na úkor rýchlosti a optimalizovanosti.

Vytvorená knižnica



Výsledkom práce je knižnica Symboliclib napísaná v Pythone. Knižnica je určená na rýchle prototypovanie a testovanie nových algoritmov, ale dá sa využiť aj na optimalizovanie existujúcich algoritmov alebo študijné účely.

Podporuje klasické konečné aj symbolické automaty a implementuje základné aj pokročilé operácie nad obomi typmi automatov. Dôraz bol kladený na algoritmy použiteľné vo formálnej verifikácii (inklúzia jazykov).

Cieľ knižnice bol umožniť jednoduché prototypovanie algoritmov pre automaty. Použité dátové štruktúry nie sú optimalizované na rýchlu manipuláciu s automatmi, ale na intuitívnu manipuláciu. Kvôli tomu je Symbolic pomalšia ako existujúce knižnice na spracovávanie automatov. Túto nevýhodu vyvažuje svojou jednoduchou použiteľnosťou.

Príklad použitia

Naľavo vidíme implementáciu algoritmu Complement v knižnici VATA, napravo v knižnici Symboliclib. Práca s dátovou štruktúrou automatu v Symboliclib je už na prvý pohľad jednoduchšia ako vo VATA.

```
template <class SymbolType, class Dict>
 VATA::ExplicitFiniteAutCore VATA::Complement(
            const VATA::ExplicitFiniteAutCore &aut,
            const Dict &/*alphabet*/) {
    typedef VATA::ExplicitFiniteAutCore ExplicitFA;

    ExplicitFA res;
    res.transitions_ = aut.transitions_;
    res.startStates_ = aut.startStates_;
    auto transitions_ = aut.transitions_;

    // All nonfinal states are marked as final states
    for (auto stateToCluster : *transitions_) {
        if (!aut.IsStateFinal(stateToCluster.first))
            res.SetStateFinal(stateToCluster.first);
        for (auto symbolToSet : *stateToCluster.second)
            for (auto stateInSet : symbolToSet.second)
                if (!aut.IsStateFinal(stateInSet))
                    res.SetStateFinal(stateInSet);
    }
    return res;
}
```

```
def complement(self):
        """
        Converts automaton into complement
        :return: complement automaton
        """
        det = self.determinize()

        complement = deepcopy(det)
        # changes final states for non-final
        complement.final = det.states - det.final

        return complement
```