We hebben gekozen om in C# een (console) applicatie te maken. Voor het tonen van de schema’s is gebruik gemaakt van GrapViz.

Onze applicatie bestaat uit meerdere klassen die hieronder uitgelegd staan.

|  |  |
| --- | --- |
| Klasse | Beschrijving |
| Program | Startpunt van onze applicatie. |
| RegExp | Hiermee wordt in de code een reguliere expressie gerepresenteerd. Hierin staat ook de bijbehorende code |
| ExampleContainer | Dit is de klasse waarin verschillende voorbeelden te vinden zijn. Deze hebben wij op sommige momenten toegepast en gebruikt. |
| Automata | Dit is de klasse die het mogelijk maakt om een NDFA of DFA te maken. Hierin wordt er met behulp van de Transition klasse verschillende transities bijgehouden |
| GraphVizEngine | Wordt gebruikt bij het maken van een afbeelding met behulp van GraphViz. |
| DotGraphEngine | Deze engine maakt het mogelijk om via GraphViz NDFA en DFA’s te tonen via een afbeelding (PNG) |
| Transition | Een onderdeel van de automaat. |
| TestAutomata | Dit is een klasse die gebruikt kan worden om de Automata klasse te testen. |
|  |  |
| Table | Deze klasse wordt gebruikt om de conversie te maken van NDFA naar DFA. Deze bevat meerdere TableColumn, op basis van de hoeveelheid aan symbolen. Deze komt in twee verschillende vormen (de HelperTable en de StateTable). Bij beide vormen zijn dezelfde klasse gebruikt, maar dan wel met een iets andere indeling |
| TableColumn | De TableColumn klasse is ervoor gemaakt om per symbol en toestand de mogelijke transitie toestanden bij te houden. |
| PartitionTable | Dit is voor het minimaliseren van een DFA. Deze klasse houdt een enkele partitie bij en zorgt ervoor dat door middel van de PartitionRows eigenlijk hetzelfde kan gebruiken als bij de Table klasse. Deze zijn achter net ietsje anders en vereisen daarom ook een eigen klasse. |
| PartitionRow | De PartitionRow klasse is ervoor om te zorgen dat er netjes alle benodigde data opgehaald en tijdelijk opgeslagen kan worden. Deze combinatie van een hoofdklasse (PartitionTable) en sub klasse (PartionRow) zorgt ervoor dat er op het einde weer gemakkelijk en overzichtelijk een DFA gemaakt kan worden. |
| NDFAToDFAEngine | Hierin staan methoden voor het omzetten van NDFA naar DFA en het optimaliseren van DFA’s met behulp van het Brzozowski algoritme. |
| NDFAtoDFAEngineExample | Voor de ondersteuning van de NDFAToDFA conversie hadden wij een voorbeeld genomen. Hieruit is ook onze eigen variant op gebaseerd. |
|  |  |
|  |  |
|  |  |
| ThompsonEngine | Dit is de klasse die het mogelijk maakt om van een reguliere expressie naar een (N)DFA te converteren. Om deze te realiseren hebben wij vooral het voorbeeld van de thompsonconstructie genomen. Hierop hebben wij onze eigen variant gemaakt. |
| ThompsonConstructionExample | Het voorbeeld van de ThompsonConstructie klasse |
|  |  |
|  |  |
| RegExpOperatorTypes | Deze klasse is gemaakt om enum waardes netjes gescheiden te houden van de andere klasse. |
| RegExpMutator | Deze klasse maakt het mogelijk om verschillende aanpassingen op een reguliere expressie uit te voeren. |
| RegExpLogicOperator | Deze klasse is verantwoordelijk voor de logica achter een reguliere expressie |
|  |  |
| StateSubType | Om te zorgen dat het minimaliseren op een juiste manier verloopt is het van belang om bij te houden welke soort toestand een transitie is. Daarom hebben wij een SubType (die bestaat uit End & Normal) en we hebben een SuperType (die bestaat uit Normal, Start & End). Hierdoor kunnen wij op het moment van omzetten tijdens een partitie gemakkelijk controleren welke soort toestand het precies was. |
| StateLetterModel | Deze klasse is ervoor om te zorgen dat er een gemakkelijke weg ontstaat voor het opzoeken van de letter op basis van een toestand. |
| StateSuperType | Dit is het SuperType van een toestand. (in StateSubType) is uitgelegd waarom dit zo is gedaan. |
| StateEquivalencyModel | Voor het minimaliseren is het van belang om te kijken of een toestand equivalent is met een andere toestand. Daarvoor is het StateEquivalencyModel voor gemaakt. Deze houdt verschillende datatypes bij , zoals het symbol en de hoeveelheid, welk soort sub en super type een toestand is. |
|  |  |
| HopCroftEngine | Deze klasse is verantwoordelijk voor het minimaliseren van de DFA. Het minimaliseren gebeurt door het gebruik van de HopCroftAlgoritme |
| HopCroftAlgorExample | We hebben voor onze eigen variant een voorbeeld gebruikt om een richting te kunnen krijgen welke verstandig was qua implementatie en aanpak. Daarvoor is deze klasse gebruikt |
|  |  |
|  |  |