|  |  |
| --- | --- |
| Vincent de Rooij & Waylon Lodder  2142881 & 2145303 | Formele methoden 2021 |

Inhoud

[1. Inleiding 2](#_Toc75120639)

[2. Klassenbeschrijving 3](#_Toc75120640)

[3. Functionaliteiten 5](#_Toc75120641)

1. Inleiding

We hebben ervoor gekozen om onze uitwerking van formele methoden 2021 uit te werken in C# om zo een (console) applicatie te maken die gebruik maakt van GrapViz om de diagrammen te tekenen.

Onze applicatie bestaat uit meerdere klassen waarvan de globale werking in hoofdstuk 2 uitgelegd wordt.

Vanuit onze applicatie is het mogelijk om verschillende conversies te maken. Zo is het mogelijk om van een reguliere expressie naar een NDFA te converteren met behulp van de ThompsonEngine klasse. Deze past de regels toe om te zorgen dat een reguliere expressie goed wordt omgezet naar een NDFA.

Daarnaast is het ook mogelijk om vanuit een NDFA naar DFA te kunnen gaan. Deze wordt afgehandeld door de NDFAToDFAEngine. Deze zet een NDFA om door middel van het opzoeken van de epsilon states. Waarbij het kijkt naar de mogelijke epsilon transities. De Epsilon overgangen worden netjes meegenomen.

Tevens is het mogelijk om op verschillende manieren te minimaliseren. Via de hopcroft methode en de reverse methodiek. Tot slot de extra functionaliteiten. Deze zijn de mogelijkheid om een DFA/NDFA om te draaien en er op deze manier een NOT van te maken. Een volledige, op console gebaseerde, test applicatie en de mogelijkheid om een reguliere expressie zelf op te bouwen op basis van een string input.

# 2. Klassenbeschrijving

Onze applicatie bestaat uit meerdere klassen die hieronder uitgelegd staan.

|  |  |
| --- | --- |
| Klasse | Beschrijving |
| Program | Startpunt van onze applicatie. Hierin zit een complete console applicatie in begrepen die het mogelijk maakt om alle functionaliteiten te kunnen testen. Verder wordt er vanuit de Program klasse ook de diagrammen geprint en door gestuurd naar de GraphVizEngine. Deze handelt vervolgens de request af. |
| RegExp | Hiermee wordt in de code een reguliere expressie gerepresenteerd. Hierin staat ook de bijbehorende code |
| TesterContainer | Deze klasse is een utility klasse die gebruikt wordt om gemakkelijk te kunnen testen met onderdelen als Minimalisatie, NOT, Reverse, Thompson, etc… Hierdoor blijft andere code voor het testen een stuk schoner en overzichtelijker |
| 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. |
| 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. |
| 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. |
| RegExpOperatorTypes | Deze klasse is gemaakt om enum waardes netjes gescheiden te houden van de andere klasse. |
| 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. |
| HopCroftEngine | Deze klasse is verantwoordelijk voor het minimaliseren van de DFA. Het minimaliseren gebeurt door het gebruik van de HopCroftAlgoritme |
| StringToRegExpBuilder | Deze klasse maakt het voor ons mogelijk om met string vorm een reguliere expressie op te bouwen en op deze manier het mogelijk te maken dat er gemakkelijk zelf een eigen expressie gemaakt kan worden. |
| DFABuilder | In deze klasse zijn de hardcoded samples gedefineerd die gebruikt kunnen worden om een DFA op te bouwen |
| NDFABuilder | In deze klasse zijn de hardcoded samples gedefineerd die gebruikt kunnen worden om een NDFA op te bouwen |
| RegExpBuilder | In deze klasse zijn de hardcoded samples gedefineerd die gebruikt kunnen worden om een Reguliere Expressie op te bouwen |

# 3. Functionaliteiten

De functionaliteiten van onze applicatie worden in dit hoofdstuk aangegeven op de manier van het invulformulier voor het assessment 2021.

|  |  |
| --- | --- |
| 1a. Hardcoded voorbeelden van (N)DFA en regulier expressies | Ja |
| 1b. Optie: parse1 voor reguliere expressies | Ja |
| 2a. Woorden genereren aan de hand van de ingevoerde reguliere expressie  Woorden genereren die niet in de taal zitten | Ja / alleen aanwezig “get accepted languages” geïmplementeerd |
| 2b. Voor DFA: testen of een woord wel/niet geaccepteerd wordt | **Nee** |
| 3a. Constructor operaties voor DFA om snel test DFA's te maken:   * begint met xyz * eindigt op xyz   bevat xyz | **Nee** |
| 3b. Operaties op DFA (liefst met tupel-constructie):   * en * of * niet | Ja – semi **“not”** is alleen aanwezig |
| 4. Thompson constructie | Ja |
| 5a. (zonder) en 5b. (met epsilon)  NDFA->DFA | Ja, met epsilon overgang |
| 6. Minimalisatie  (via reverse)  Bonus: ook algoritme uit het dictaat (via opsplitsen tabel) | Ja – beide algoritmes |
| 7. Gelijkheid op reguliere expressies / (NDFA) | **Nee** |
| 8. Test Applicatie:  (Makkelijk testen, plaatje met GraphViz, wegschrijven test resultaten…) | Ja, navigeerbaar console en GraphViz images |
| 9 (extra) vertaling tussen: NDFA  grammatica  Grammatica -> NDFA | **Nee** |