TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Roman Holvason 135212IAPB

SPARK'I AVALDISTE TEISENDAMINE SIMULINK DIAGRAMMIDEKS

Bakalaureusetöö

Juhendaja: Tõnu Näks
MSc

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Roman Holvason

21.05.2019

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks oli luua konverter, mis teisendab SPARK programmeerimiskeeles kirjutatud avaldised Simulink diagrammideks. Eesmärgi saavutamiseks koostati Xtext vahenditega avaldiste grammatika ja kirjutati programm, mis parsib avaldist selle grammatika reeglite järgi ning teisendab leitud elemendid vastavateks Simulink plokkideks.

Programmi loomiseks tuli kõigepealt tutvuda töö eesmärkide saavutamiseks vajalike tehnoloogiatega ja programmeerimiskeeltega. Samuti oli vaja määrata toetatud avaldiste alamhulk, keele töötlemise strateegia, mis viisil tulemust salvestada ja kuidas teisendada vastavateks Simulink plokkideks. Esimesena kirjutati grammatika, mis vastutas parsimis protsessi ja keele süntaksi eest. Teiseks loodi meetod parsitud elementide salvestamiseks, töötlemiseks ja Simulink plokkide teisendamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 20 leheküljel, 5 peatükki, 10 joonist, 0 tabelit.

Abstract

CONVERTING SPARK EXPRESSIONS TO SIMULINK DIAGRAMS

The main goal of the current bachelor's thesis is to create a converter that converts all chosen expressions, which are written in SPARK programming language to Simulink diagrams. To accomplish this goal was created program, which parses expressions by grammar rules and converts all founded elements by corresponding Simulink blocks.

All work was written in XText software framework, which is using for implementing programming languages and domain-specific languages (DSL). DSL is able to implement and describe language syntax using grammar rules. Grammar rules also define and return expression elements and their types during parsing. After parsing all elements are stored in abstract syntax tree (AST). This structure holds each element in different nodes and also it helps to trace connections between them. Program is going through the syntax tree and convert all elements to corresponding Simulink blocks, which are implemented in jar file. All connections between expression elements are converted to signals, which goes out from every block outport and goes into other block inports. After converting all blocks, signals, values and names are saved in the .xmi file format. The last step is to use QGEN code generator, which transforms all data in .xmi file to Simulink diagram.

The thesis is in Estonian and contains 20 pages of text, 5 chapters, 10 figures, 0 tables.

Lühendite ja mõistete sõnastik

SysML	Süsteemide Modelleerimise keel (Systems Modeling Language)
DSL	Domeen-spetsiifiline keel (Domain Specific Language)
AST	Abstraktne süntaksipuu (Abstract Syntax Tree)
kvantifitseerima	teatud omadusi mõõtma või arvuliste näitajate kaudu väljendama
ANTLR	parser, mis kasutab parsimiseks LL(*) algoritmi (ANother Tool For Language Recognition)
LL(*)	Vasak-paremale, Vasakpoolne tuletamine (**L**eft-to-right, **L**eftmost derivation)
EMF	Eclipse Modelleerimis raamistik (Eclipse Modeling Framework)
UML	Ühtne modelleerimiskeel (Unified Modeling Language)
XML	‘Laiendav märgistuskeel’ (Extensible Markup Language)

Sisukord

1 Sissejuhatus 8

2 Kasutatud tehnoloogiad 9

2.1 SPARK 9

2.2 Simulink 9

2.2.1 Simulink Block Diagram 9

2.3 Xtext/Xtend 12

2.3.1 Xtext 12

2.3.2 Xtend 12

3 SPARK - Simulink teisendus 13

3.1 Toetatud avaldised 13

3.1.1 Operaatorid ja literaalid 13

3.1.2 Atribuut ja tingimuslik avaldis 14

3.2 Avaldiste esitamine Simulinkis 15

3.3 Testmudelid 18

4 Avaldise töötlemine 20

4.1 Grammatika 20

4.2 Parser 22

4.2.1 Abstract Syntax Tree 23

4.3 Teisendus plokkideks 24

4.3.1 Import Simulinki 25

4.4 Alternatiivne võimalus 25

5 Kokkuvõte 27

5.1 Edasine töö 27

Kasutatud kirjandus 28

Lisa 1 - XText Grammatika 30

Jooniste loetelu

Joonis 1. Ploki simulatsioon [5]. 10
Joonis 2. Alamsüsteem koos kahe jälgijaga [2].. 11
Joonis 3. Jälgija ToggleOnOff sisu [2].. 11
Joonis 4. Simulink diagramm näide nr 1. 18
Joonis 5. Simulink diagramm näide nr 2. 18
Joonis 6. Simulink diagramm näide nr 3. 19
Joonis 7. Simulink diagramm näide nr 4. 19
Joonis 8. Simulink diagramm näide nr 5. 20
Joonis 9. Abstract syntax tree [24]. 23
Joonis 10. Parseri protsessi ülevaade [22]. 24

1 Sissejuhatus

Sünkroonne vaatleja on süsteemimudeli lisand mis jälgib selle muutujate seisundit ja tekitab sündmuse, kui jälgijas defineeritud tingimus on rahuldatud [1]. Selle töö kontekstis käsitleme vaatlejaid mis on lisatud Simulink mudelisse kontrolleri omaduste kirjeldamiseks. Sellise vaatleja roll on jälgida allsüsteemi sisendeid ja väljundeid simulatsiooni ajal ja peatada simulatsioon (kasutades Assert plokki) kui vaatlejas kirjeldatud tingimus ei ole täidetud. Kui sama kontroller realiseerida funktsioonina lepinguid toetavas programmeerimiskeeles siis saab vaatleja kodeerida selle funktsiooni eel- või järeltingimusena. Käesolev töö realiseerib ühe sammu töövoos, kus süsteeminõuded spetsifitseeritakse SysML mudelis kitsendustena kasutades SPARK keelt ja teisendatakse siis Simulink mudelis olevateks jälgijateks [2].

Antud töö eesmärk on arendada konverter mis teisendab SysML'is esitatud piirangute definitsioone Simulink diagrammideks. Eesmärgi saavutamiseks kasutatakse Xtext raamistikku, Eclipse vahendeid, Xtend programmeerimiskeelt ning QGeni[1] koodigeneraatorit parserist saadud abstraktse süntaksipuust (AST) Simulink diagrammi saamiseks.

Töö võib jagada kaheks alamülesandeks. Esiteks oli vaja kirjeldada toetatud avaldisi ja nende elementide keele süntaksit. Seejärel defineerida grammatika millest saab genereerida keele parseri ning andmestruktuurid keeleelementide manipuleerimiseks ning salvestamiseks.

Teiseks loodi meetod, mis teisendab avaldises iga leitud elemendi vastavaks Simulink plokiks ja ühendab need vastavalt AST struktuurile signaalidega. Tulemus salvestatakse .xmi failiformaati. Lõpuks kasutatakse QGEN paketti kuuluvat mudeli importerit, mis teisendab faili andmed Simulink diagrammiks.

Loodud programmi kood on GitLabi repositooriumis[2].

[1] https://www.adacore.com/qgen

[2] https://gitlab.cs.ttu.ee/Roman.Holvason/iapb.git

2 Kasutatud tehnoloogiad

Selles peatükis on kirjeldatud kasutatud programmeerimiskeeled ja raamistikud. Lõputöö ülesande lahendamiseks on vajalike tehnoloogiate valik ei olnud töö skoobis. Kasutatavad tehnoloogiad olid juba varasemate etappide käigus identifitseeritud [2] ja määrati ülesandepüstitusega.

2.1 SPARK

SPARK on spetsifitseerimis- ja programmeerimiskeel mis on mõeldud tarkvara arendamiseks valdkondades kus on hädavajalik prognoositav ja tõestatult usaldusväärne käitumine [3]. Spetsifikatsiooni keel võimaldab kirjeldada süsteemi palju kõrgemal tasemel kui programmeerimiskeel. SPARK keel hõlmab suurt osa Ada programmeerimiskeelest ning kasutab lepinguid, et kirjeldada komponentide spetsifikatsioon. Programmi käitumist saab kirjeldada alamprogrammide eel- ja järeltingimuste ning väidete (*assertion*) abil.

Selle lõputöö raames ei olnud vaja selles keeles programmeerida. Teisendati SPARK keele süntaksit kasutatavaid avaldisi..

2.2 Simulink

Simulink on graafiline programmeerimiskeskkond dünaamiliste süsteemide modelleerimiseks, simuleerimiseks ja analüüsimiseks. Süsteemi mudel esitatakse plokidiagrammina (*simulink block diagram*) kus iga plokk kirjeldab mõnd arvutust. Tööriist sisaldab teeke standardplokkidega erinevate rakendusvaldkondade jaoks (juhtimissüsteemid, signaalitöötlus, närvivõrgud jne) [4].

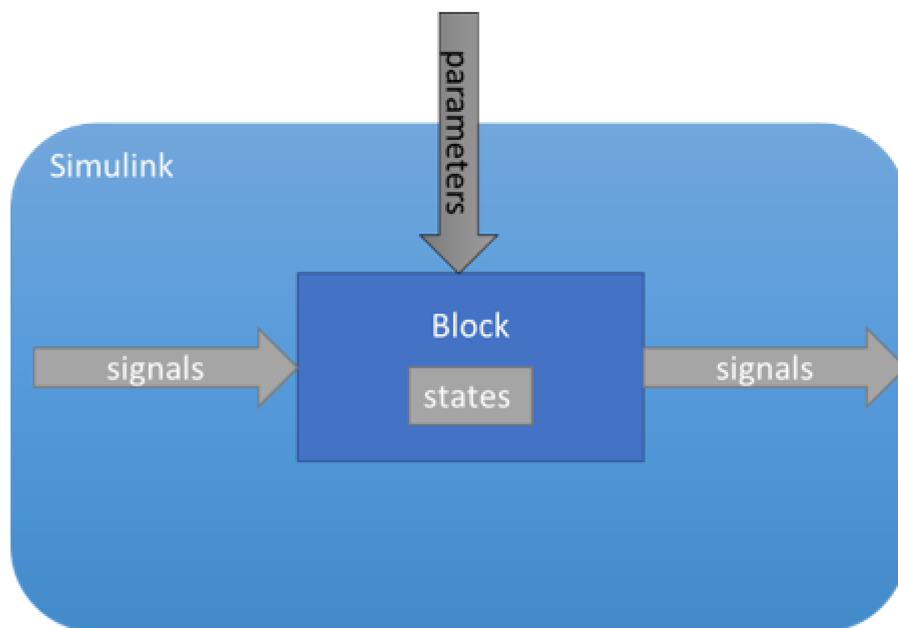
2.2.1 Simulink Block Diagram

Plokidiagramm koosneb plokkidest, mis kirjeldavad arvutusi ning seostest (signaalidest) nende plokkide vahel, mis kirjeldavad arvutustulemuste ülekandmist ühest plokist teise.

Rakenduse vaatepunktist võib plokk kujutada füüsilist komponenti, alamsüsteemi või süsteemi üht funktsiooni.

Arvutustes kasutatavad andmed jagunevad kolme gruppi:

- Signaalid (Signals) - ploki sisendid ja väljundid, arvutatakse simulatsiooni käigus
- Olekud (States) - sisemised väärtused, mis kujutavad ploki dünaamikat, arvutatakse simulatsiooni käigus
- Parameetrid (Parameters) - ploki käitumist mõjutavad väärtused, sisestatakse kasutaja poolt ja on simulatsiooni käigus konstantsed



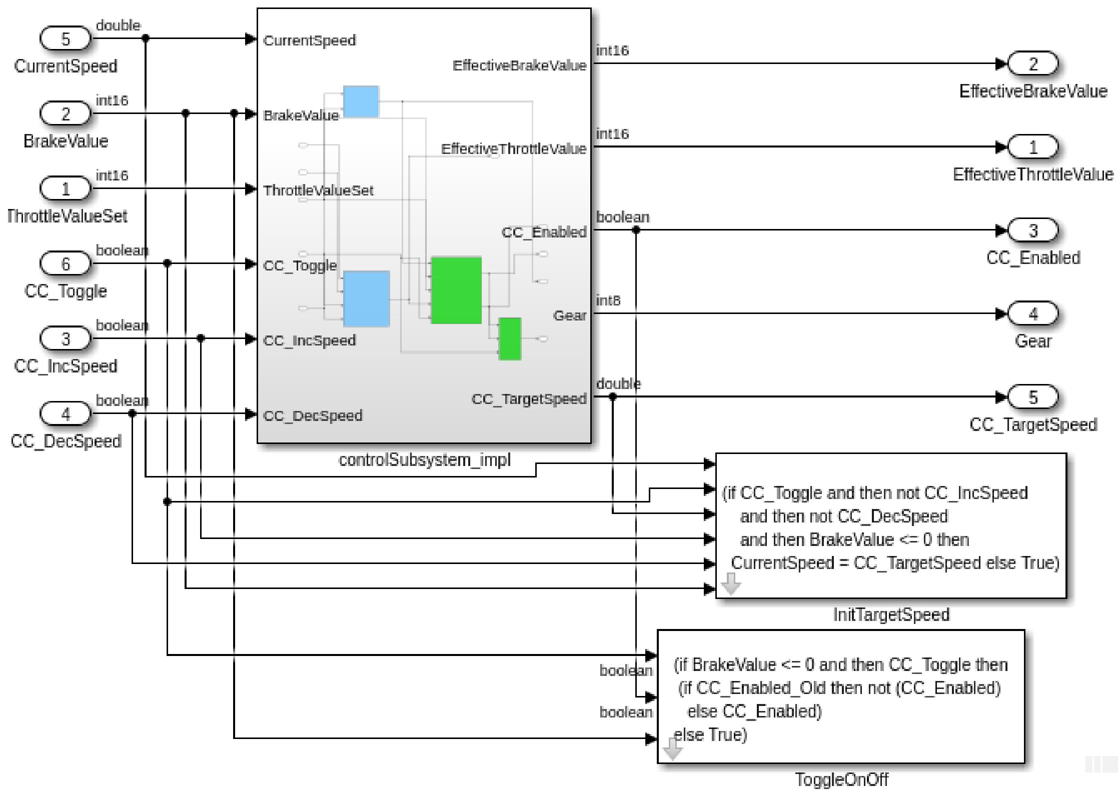
Joonis 1. Ploki simulatsioon *[5]*.

Süsteemi käitumise simuleerimiseks käivitatakse diagrammil kirjeldatud arvutust perioodiliselt. Igal käivitusel arvutab Simulink uued väärtused signaalide ja olekute jaoks. Ringseoste vältimiseks on oluline, et kõigepealt teostatakse kõik arvutused eelmisel sammul salvestatud olekute väärtustega ja alles peale arvutuste lõpetamist uuendatakse olekumuutujaid.

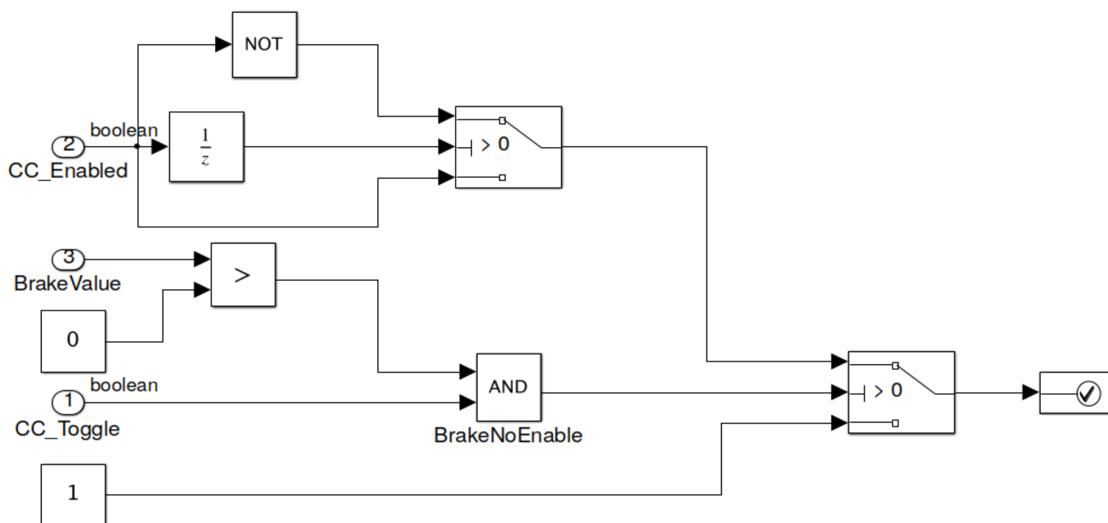
Simulinki diagrammi osi on võimalik grupeerida alamsüsteemideks mis võivad olla virtuaalsed (alamsüsteem on kasutusel ainult diagrammi loetavuse parandamiseks) või atomaarsed (alamsüsteemi sisse jäävad arvutused teostatakse koos ja ülejäänud mudeli osad “näevad” arvutuse tulemust peale kõigi alamsüsteemi väljundite väljaarvutamist. Tuues paralleeli C keelega võiks öelda, et virtuaalne alamsüsteem on sarnane makroga. Programmi kirjutamise ajal peidetakse funktsionaalsus makro nime taha. Eelprotsessor asendab nime enne kompileerimist programmilõiguga. Atomaarne alamsüsteem vastab

funktsioonile -- funktsiooni sisse jääv funktsionaalsus on ka kompilaatori jaoks alamprogrammiks eraldatud.

Antud töös eeldatakse, et SysML'ist saadud programmi komponendid ning kitsendused esitatakse mõlemad alamsüsteemidena. Kitsendusest genereeritud alamsüsteem on jälgija, mis kontrollib funktsiooni esitava alamsüsteemi sisendite ja väljundite korrektsust. Antud töö tulemusena tehtav teisendaja koostab jälgija sees oleva Simulinki diagrammi SPARK'i avaldise põhjal.



Joonis 2. Alamsüsteem koos kahe jälgijaga [2].



Joonis 3. Jälgija ToggleOnOff sisu [2].

2.3 Xtext/Xtend

2.3.1 Xtext

Xtext on raamistik valdkonnaspetsiifiliste keelte (*Domain-specific Language, DSL*) arendamiseks. DSL on programmeerimiskeel või siis spetsifitseerimiskeel, mida kasutatakse süsteemi või komponendi nõuete, lahenduse või muude tunnusomaduste väljendamiseks [26]. Xtext raamistik võimaldab keeli kiiresti luua ja hõlmab kõiki keelestruktuuri aspekte, alates parserist, koodigeneraatorist või teisendajast kuni täieliku Eclipse IDE integratsioonini [6]. Võrreldes teiste parserigeneraatoritega (nagu näiteks Bison, Lex, ANTLR), genereerib Xtext lisaks parserile ka klassimudeli abstraktse süntaksipuu (AST) objektide jaoks. Sellest omakorda saab Eclipse Modelling Framework vahendite abil genereerida süntaksitundliku keeleredaktori ja erinevaid teisendajaid.

Keelt kirjeldab grammatika mis näitab millistest elementidest keel koosneb ja millised on reeglid selles keeles avaldiste ja lausete moodustamiseks [7]. Pärast grammatika määratlemist, genereerib Xtext automaatselt mitmeid DSL töötlemise komponente ja IDE tööriistu.

Neljandas peatükis on põhjalikumalt kirjeldatud parsimise protsessi ja grammatika rolli.

2.3.2 Xtend

Xtend on üldotstarbeline programmeerimiskeel mis on mugav Xtexti parserist saadud andmestruktuuride teisendamiseks, kuid ei ole otseselt Xtext keelest sõltuv. Xtend on Java keele dialekt mida saab kasutada Javaga koos või selle alternatiivina. Java asemel võib vabalt kasutada Xtend programmeerimis keelt, kuna seda on lihtsam kasutada ja võimaldab kirjutada loetavama koodi [6]. Süntaktiliselt ja semantiliselt Xtend on Java programmeeriskeele moodi aga on täiustatud mitmeid aspekte: laiendamismeetodid (*extension methods*), lambda avaldised (*lambda expressions*), operaatorite üledefineerimine (*operator overloading*), võimas switch-avaldised (*powerful switch expressions*), mall-avaldised (*template expressions*), polümorfne ümbersuunamine (*multiple dispatch*) ja muud.

3 SPARK - Simulink teisendus

Selles peatükis defineerime kõigepealt toetatavate SPARK avaldiste alamhulga ja esitame siis Simulink plokkide ja SPARK avaldiste vastavused. Viimasena on selles peatükis toodud SPARK avaldise ja vastavate Simulink diagrammide näidised.

3.1 Toetatud avaldised

Avaldis on valem, mis määrab arvutuse või otsingu väärtuse. Avaldisel on tüüp, mis on määratud avaldise poolt arvutatava väärtuse tüübiga. Tavaliselt see on üks primitiivtüüpidest: numbriline (*numerical*), string või loogiline (*logical*). Avaldis ise on ühe või mitme konstandi, muutuja, operaatori ja funktsiooni kombinatsioon.

Selle töö raames huvitavad meid tõeväärtusavaldised, mis kontrollivad kas programmi sisendid või väljundid vastavad avaldisega kirjeldatud kitsendustele. S.t. vaatleme loogikaavaldisi, mis võivad küll sisaldada aritmeetilise operatsioone, kuid nad peavad tagastama alati tõeväärtuse (True | False).

Vaatame lähemalt, mis operaatorid, literaalid ja väljendid olid kasutusel SPARK avaldises selle töö raames.

3.1.1 Operaatorid ja literaalid

Loogiline Operaator (Logical Operators).

Näide: *if (Wind and Rain) or Snow then ...*

Avaldis, mis koosneb kahest omavahel seotud relatsioonist on tavaliselt ühendatud ***´and´*** või ***´or´*** loogilise operaatoriga. SPARK keeles kasutatakse ka ***´and then´*** ja ***´or else´*** lühis-operaatoreid (*short-circuit operator*) [8]. Erinevus seisneb avaldise lahendamise strateegias. “and” ja “or” operaatorite puhul lahendatakse alati kogu avaldis. “and then” ning “or else” operaatorite puhul väärtustatakse avaldis parem pool ainult juhul, kui vasak pool ei määra üheselt lõpptulemust. Kogu avaldis kus on kasutatud loogiline operaator võib kirjeldada niimoodi: *expression ::= relation {**and | or | and then | or else** relation}.*

Vaatame lähemalt millest võib koosneda relatsioon. Relatsiooni saab kirjeldada valemiga: *relation ::=* ***simple_expression [relational_operator simple_expression]*** [9].

Lihtne avaldis (simple expression).

Näide: *abs(10 / 3) + In1 * 10*

Lihtne avaldis võib koosneda ülejäänutest madalamatest operaatoritest või literaalidest. Korrutamis operaatorid: ´*´ (korrutamine), ´/´ (jagamine), ***´mod´*** (moodul) ja ***´rem´*** (jääk) [10].

Binaarsed operaatorid: ´+´ (liitmine), ´**-**´ (lahutamine) määratakse iga konkreetse numbrilise tüübile. On olemas ka ´**&**´ (liitumine) operaator, mis kasutatakse string ja char tüüpide liitmiseks [11].

Unaarsed operaatorid: ´+´ (positiivne) ja ´**-**´ (negatiivne) määratakse iga konkreetse numbrilise tüübile. Unaarse operaatorite külge kuluvad veel ***´abs´*** (absoluutne väärtus), mida samuti määratakse numbriliste tüüpidele ja ***´not´*** (eitus) mida määratakse boolean tüüpidele [12].

Literaal on fikseeritud väärtuse esitus lähtekoodis. On olemas numeric, string, char ja null literaalid [13].

Relatsiooniline Operaator (Relational Operators).

Näide: *In1 /= 0 and In1 >= 10*

Relatsioonilisi operaatoreid võib olla kahte tüüpi. Võrdlus operaatorid: ´=´ (võrdub) ja ´/=´ (ei võrdu) ja järjestus operaatorid: ´>´ (rohkem kui), ´<´ (väiksem kui), ´>=´ (rohkem või võrdub) ja ´<=´ (väiksem või võrdub) [14].

3.1.2 Atribuut ja tingimuslik avaldis

SPARK keeles on võimalik panna muutuja juurde tunnus (atribuut), mis võimaldab programmil lugeda muutuja metaandmeid (tüüp, määramispiirkond jms). Atribuut eraldatakse muutuja nimes sümboliga ' (apostroof).

Atribuut **Old** on kasutatud, et tähistada objekti (muutuja) väärtust enne alamprogrammi käivitamist. Käivituse ajal tehakse alamprogrammi sisenemisel muutuja X koopia ja loetakse seda alles siis kui leitakse X'Old väärtust järel-tingimustes [15]. See on vajalik

juhul, kui programm muudab mõnd oma parameetrit ja lepingus on vaja näidata väljundi seost sisendiga. Näiteks loenduri puhul oleks protseduuri spetsifikatsioon selline:

```
procedure Count (Val : in out Integer)
  with Post => (Val = Val'Old + 1);
```

SPARK lubab veel tingimuslikke avaldisi ja kvantifitseeritud avaldisi. Selles töös on käsitletud if-avaldis:

if_expression ::= **if** condition **then** *dependent*_expression
{**elsif** condition **then** *dependent*_expression}
[**else** *dependent*_expression]

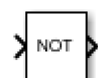
Sõltuva avaldise (*dependent_expression*) väärtus võetakse arvesse siis kui üks või mitu vastavat tingimust (*condition*) on täidetud. Tingimuslik avaldis on ka case_expression mida antud töös ei käsitleta. Grammatikas on kirjeldatud If lause struktuur ja parsimis metoodika.

Kvantifitseeritud avaldis on näiteks for_expression ja mida samuti selles töös ei käsitleta

3.2 Avaldiste esitamine Simulinkis

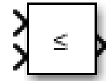
Iga SPARKi operaatori, konstandi või muutuja jaoks on võimalik leida Simulinkis plokk, mis vastab sama tingimustele ja omadustele. Plokkide kokku ühendamisel vastavalt AST struktuurile on võimalik konstrueerida diagramm, mis vastab avaldisele. Allpool on toodud Simulink plokkide vastavused ja nende omadused.

Kõik loogilised operaatorid on Simulink´is esitatud *Logical Operator* plokis. Vaikimisi sellele plokile on määratud operaator AND aga seadetes on võimalik valida teisi. Simulink toetab rohkem loogilisi operaatoreid kui on toodud eelmises peatükis. Ploki väärtus võib olla [AND, OR, NAND, NOR, XOR, NXOR ja NOT]. Panen tähele, et unaarne operaator *´not´* (eitus) on esitatud Simulink´is sama plokina. Operaator *´not´* võtab vastu ainult ühe sisendi, kuigi teistele operaatoritele on võimalik panna kaks või rohkem [16].



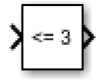
Simulink plokkide teegis *Logical Operator* plokk asub *Simulink/Logic and Bit Operations* kaustas.

Relatsioonilised operaatorid on esitatud Simulink´is *Relational Operator* plokis. Võrdlus ja järjestus operaatorid on kõik ühes plokis. Plokk võtab vastu kaks sisendit mida hakkab võrdlema valitud operaatoriga. Simulink ja SPARK keeles võrdlus operaatoritel on erinev süntaks. SPARK keele operaatorite ´=´ ja ´/=´ asemel Simulink kasutab ´==´ ja ´~=´ vastavalt [17].

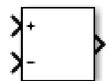


Simulink plokkide teegis *Relational Operator* plokk asub *Simulink/Logic and Bit Operations* kaustas.

Eraldi konstantidega võrdlemiseks Simulink´is on *Compare To Constant* plokk. Kasutada võib kõiki relatsioonilisi operaatoreid ja konstanti peab eelnevalt määratleda. Nulliga võrdlemiseks on *Compare To Zero* plokk. Plokid asuvad *Simulink/Logic and Bit Operations* kaustas.

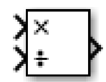


Binaarsed operaatorid liitmine ja lahutamine on võimalik esitada Simulink diagrammis erinevate plokkidega. *Add, Subtract, Sum of Elements* ja *Sum* on identsed plokid. Ploki toimingut ja sisendite arvu määratakse (+)pluss või (-)miinus parameetritega. Näiteks ´+ - +´ tähendab kolme sisendit. Plokk lahutab teise (keskmise) sisendi esimesest (ülemisest) sisendist ja lisab seejärel kolmanda (alumise) sisendi [18].



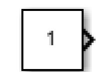
Simulink plokkide teegis *Add, Subtract, Sum of Elements* ja *Sum* plokid asuvad *Simulink/Math Operations* kaustas.

Korrutamise ja jagamise tehete jaoks Simulink´is on eraldi plokid. *Product* plokk on korrutamise jaoks ja *Divide* plokk jagamise. Diagrammi simulatsiooni käigus selgus, et mõlemad plokid saavad tegeleda mõlema tehetega.



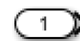
Simulink plokkide teegis *Product* ja *Divide* plokid asuvad *Simulink/Math Operations* kaustas.

Kõik muutujad (parameetrid) ja konstandid, mis on esitatud avaldises on vaja ka defineerida plokkidena. *Constant* plokk genereerib reaalse või kompleks väärtuse signaali. Kasutatakse seda ploki pideva signaali sisestamiseks [19].



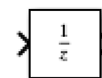
Boolean tüüp on esitatud diagrammis samuti Constant plokina, kus 1 on True väärtus ja 0 on False.

Muutuja on esitatud diagrammis *Inport* plokina. Inport plokk ühendab süsteemi väljastpoolt tulnud signaali selle sama süsteemiga. Muutuja on tavaliselt esitatud avaldises koos loogiliste, relatsiooniliste või binaarsete operaatoritega. See tähendab, et muutuja ongi üks sisend parameetritest mida operaator plokk võtab vastu.

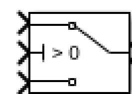


Simulink plokkide teegis *Constant* ja *Inport* plokid asuvad *Simulink/Sources* kaustas.

'Old atribuudile vastab jälgija sisendisse ühendatud *Unit Delay* plokk. See plokk hoiab ja viivitab sisse tulnud signaali määratud ajaperioodi järgi. Simulink plokkide teegis asub *Unit Delay* plokk asub *Simulink/Discrete* kaustas.

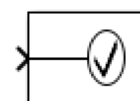


Eelmises peatükis on toodud näidisena tingimuslik avaldis ja üks nendest on If lause. If lause jaoks, kus on kasutusel ainult if-else struktuur on Simulink´is eraldi plokk. *Switch* plokk läbib esimest sisendit või kolmandat sisendsignaali ainult teise sisend väärtuse alusel [20]. Sama loogika on if-else lauses, kus ühe tingimuse väärtuse järgi valitakse vastava väljendi.



Simulink plokkide teegis *Switch* plokk asub *Simulink/Signal Routing* kaustas.

Iga diagrammis kirjeldatud jada peab lõppema *Assert* plokiga mis peatab simulatsiooni, kui tema sisend ei ole tõene (True või nullist erinev arv). Simulink plokkide teegis asub *Assert* plokk *Simulink/Model Verification* kaustas.



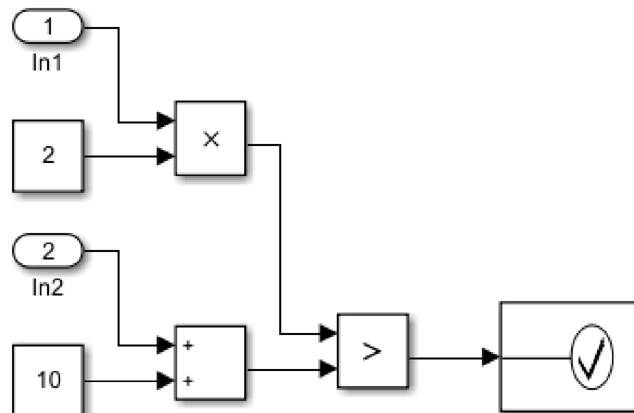
Peatükis *´3.1 Toetatud avaldised´* on toodud palju rohkem operaatoreid kui selles peatükis Simulink plokke. Näiteks ´**mod**´ (moodul), ´**rem**´ (jääk), ´**abs**´ (absoluutne väärtus), ´**&**´ (string liitmine), ´******´ (astendamine) ja **case_avaldis.** Kõikide nende operaatorite jaoks Simulink´is on olemas ka eraldi plokid. Selle töö raames neid operaatoreid ei olnud vaja kasutata SPARK avaldistes. Seega Grammatika failis need operaatorid ei ole kirjeldatud ega võetud kasutusele.

3.3 Testmudelid

Selles peatükis on esitatud avaldiste näidised ja nende Simulink diagrammide vastavused. Näidistes on esitatud kõik Simulink plokid mis on toodud eelmises peatükis.

Näidis nr 1.

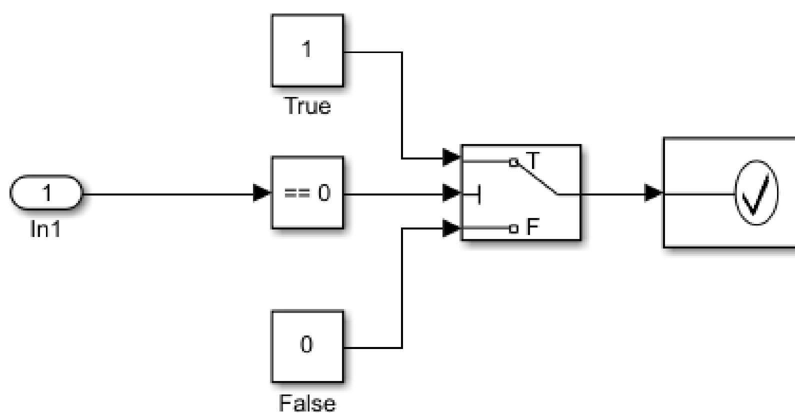
SPARK avaldis: ((In1 * 2) > (In2 + 10))



Joonis 4. Simulink diagramm näide nr 1.

Näidis nr 2.

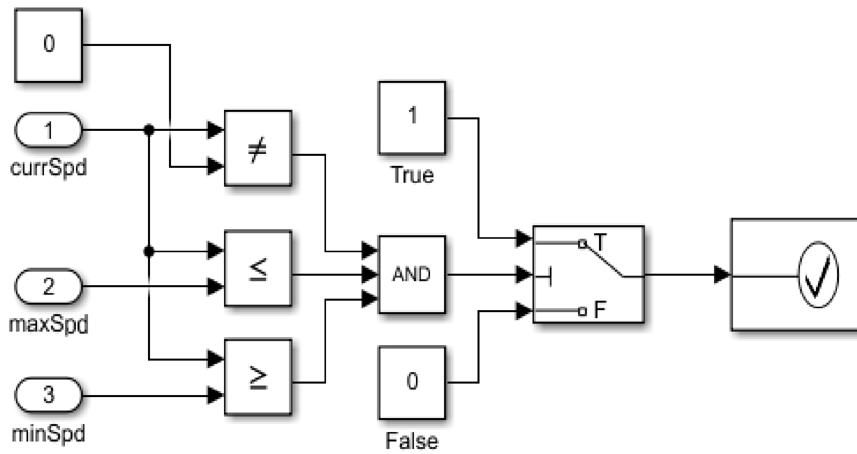
SPARK avaldis: if In1 = 0 then TRUE else FALSE



Joonis 5. Simulink diagramm näide nr 2.

Näidis nr 3.

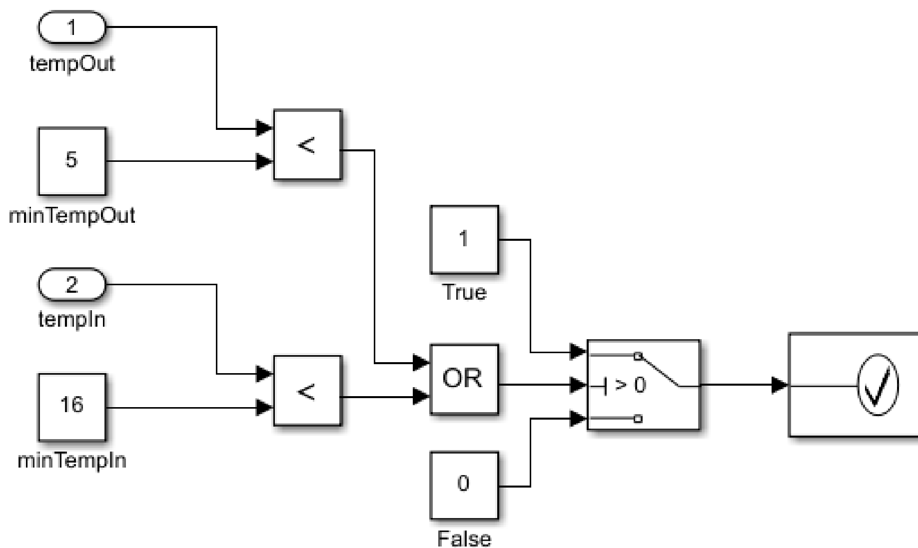
SPARK avaldis: if currSpd /= 0 and currSpd <= maxSpd and currSpd >= minSpd
then TRUE
else FALSE



Joonis 6. Simulink diagramm näide nr 3.

Näidis nr 4.

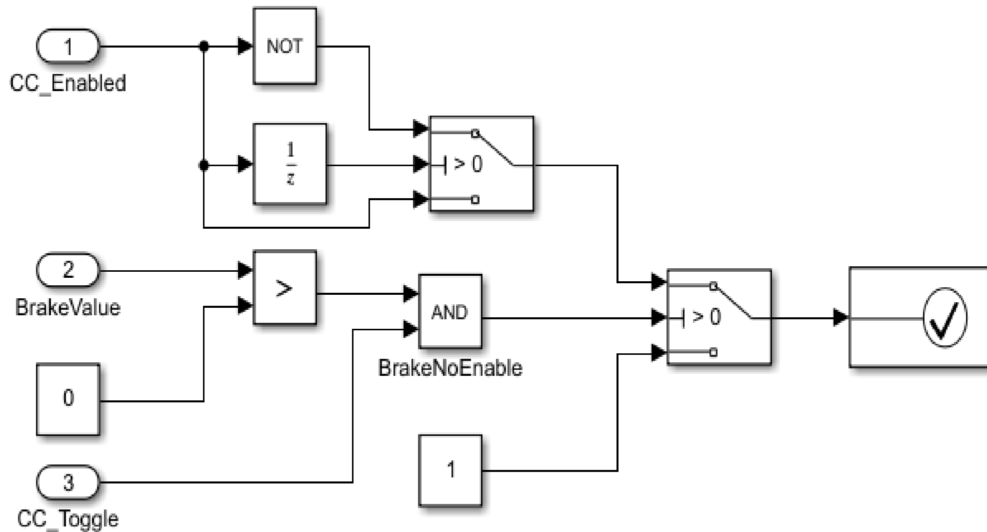
SPARK avaldis: if tempOut < 5 or tempIn < 16
then TRUE
else FALSE



Joonis 7. Simulink diagramm näide nr 4.

Näidis nr 5.

SPARK avaldis: (if CC_Toggle and BrakeValue > 0 then
(if CC_Enabled'Old then
not CC_Enabled
else
CC_Enabled)
else TRUE)



Joonis 8. Simulink diagramm näide nr 5.

4 Avaldise töötlemine

4.1 Grammatika

Keele süntaksi kirjeldamiseks koostatakse grammatika, millest on võimalik automaatselt genereerida parser vastava keele jaoks. Sel põhjusel nimetatakse selliseid vahendeid parser-generaatoriks ja selles kontekstis spetsifikatsiooni nimetatakse *grammatikaks*. Üldiselt võib öelda, et *grammatika* on reeglite kogum, mis kirjeldab keele süntaksit [6]. Reeglid teisendavad keele teksti märgendite (*token*) puuks kus märgendid tähistavad keeles defineeritud sümboleid (sõnu).

Java-maailmas on tõenäoliselt kõige tuntum ANTLR[3] keel. See võimaldab kirjeldada keele grammatika ja genereerib siis sellest parseri ja lexeri. Antud töös on kasutuse Xtext, mis samuti võimaldab koostada keelekirjeldus ja selle alusel genereerida keele töötlemise vahendid
Kogu kirjutatud selle töö jaoks grammatika asub lisas **Lisa 1**.

```
grammar ttu.diploma.expression.Expression with
org.eclipse.xtext.common.Terminals

generate expression "http://www.diploma.ttu/expression/Expression"

Model: elements += Expression*;

Expression: Or;

Or returns Expression:
	And (=> ({Or.left = current} op = OrOp) right = And)*;
```

Esimene rida deklareerib keele ja grammatika nimetuse, mis vastab .xtext faili nimele. Faili nimi on Expression.xtext ja asub ttu.diploma.expression paketis. Teine rida impordib paketi Terminals, mis sisaldab standardsete terminalsümbolite reegleid. Näiteks stringi, numbreid ja kommentaare ei ole vaja oma grammatikas defineerida. Terminal grammatika on Xtext teegi osa.
Kolmas rida viitab, et Xtext võib tuua välja Ecore mudeleid grammatikast. Ecore mudel koosneb põhimõtteliselt EPackage´ist, mis sisaldab EClasses, EDataType ja EEnume. Kirjutatud reaga genereeritakse EPackage nimetusega *expression* ja nsURL "*http://www.diploma.ttu/expression/Expression*". Xtext lisab seejärel EClasses, EAttributes ja EReferences erinevate parseri reeglitele, mis on kirjutatud grammatikas [7].

Pärast kolme esimest deklaratsiooni rida määratletakse grammatika tegelikud reeglid. Iga grammatika esimene reegel määratleb kust parser alustab ja määratleb AST juurelementi tüüpi [6]. Selle töö jaoks on deklareeritud, et DSL on Expression (avaldiste) elementide kogum. Seda kogumi hoitakse Model objektina. Ülejäänud grammatika reeglid defineerivad mis alamelementidest võib koosneda Expression. Esimene alamelement millele viitab *Expression* on loogiline operaator Or. Or reegel määrab struktuuri ja

[3] https://www.antlr.org/

rekursiivselt kutsub välja järgmist alamelementi. Järgmine alamelement millele viidatakse on loogiline operaator And.

4.2 Parser

Parsimine on protsess, milles arvutikeeles või andmestruktuurides esinevaid sõnesid analüüsitakse vastavalt keele formaalse grammatika reeglitele [21]. Eelmises peatükis oli mainitud, et reeglit kirjeldatakse märgendite järjestuse abil. Iga märgend on atomaarne element programmeerimiskeele jaoks. See võib olla märksõna, identifikaator, literaal, operaator (näiteks aritmeetiline- või võrdlusoperaator) või eraldaja (näiteks ümarsulud või semikoolon).

Protsess, mis konverteerib tähemärkide jada märgendite jadaks nimetatakse *lexical analysis*. Samuti on vaja veenduda, et on moodustatud õige avaldis selles keeles. See tähendab, et on järgitud keele süntaktilist struktuuri. Seda faasi nimetatakse *syntactic analysis* või parsimine [6]. Parsimise reeglid on määratletud grammatikaga, mis määravad rekursiivselt komponente, millest võib koosneda avaldis ja järjekorda, milles nad võivad ilmuda [22].

Xtext kasutab ANTLR parserit, mis rakendab LL(*) algoritmi. LL (**L**eft-to-right, **L**eftmost derivation) algoritm tähendab, et parser analüüsib sisendit vasakult paremale ja toestab vasakpoolsemat tuletamist. Siiski selline parser ei saa tegeleda vasak rekursiooniga. Grammatika jaoks see tähendab, et on vaja eemaldada vasak rekursiooni ja kasutada vasak faktooringut. Näidisena on toodud kaks reeglit tehtud töö grammatikast.

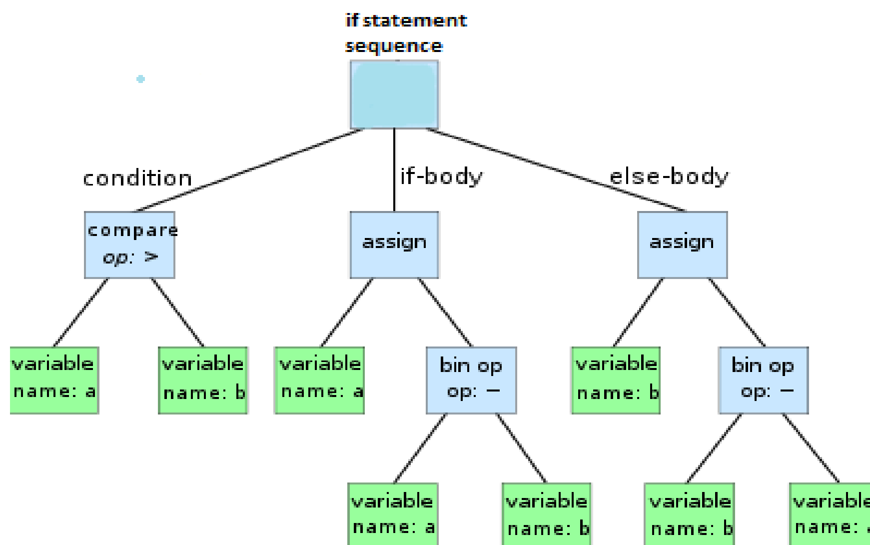
```
Equality returns Expression:
      Comparison (=> ({Equality.left = current} op = ('=' | '/='))
      right = Comparison)?;

Comparison returns Expression:
      PlusOrMinus (=>({Comparison.left = current} op = ('>=' | '>' |
      '<' | '<=')) right = PlusOrMinus)?;
```

Equality reegel kutsub välja mitte iseennast vaid järgmist Comparison reeglit. Antud juhul rekursiooni eest vastutab juba reegli parem pool ja seda ANTLR parser toetab. Operaatori tähtsus on määratletud delegeerimis järjekorraga. Mida hiljem seda reeglit välja kutsutakse seda kõrgema prioriteediga see on [23].

4.2.1 Abstract Syntax Tree

Avaldise parsimine on ainult esimene etapp selle töötlemiseks. Kui avaldis on süntaktiliselt õigesti kontrollitud siis rakendusprogramm peab midagi tegema avaldise elementidega [6]. Xtext´is iga reegel tagastab teatud väärtust. Reegli tagastamise tüüpi saab täpselt määratleda kasutades märksõna *´return´*. Kui reegli nimetus on sama mis tagastamis tüübi nimetus siis võib mitte kirjutada. Eelmises peatükis toodud näidisel on näha, et iga reegel tagastab sama Expression tüüpi, kuna need on rekursiivsed.

Parsimise käigus tagastatud objektid on vaja salvestada mällu, et oleks võimalik teha semantilist analüüsi. Muidu oleks vaja sama avaldise kogu aeg uuesti parsida. Mugav kuju kus võib hoida parsimise käigus leitud elemente on puu struktuur. Abstraktne Süntaksipuu (AST) võimaldab kujutada avaldise abstraktse süntaksi struktuurina [6].

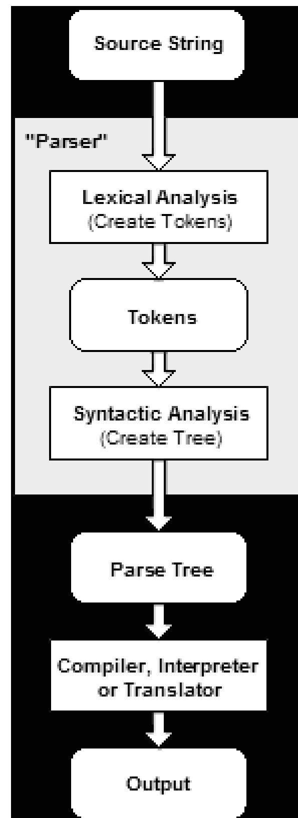


Joonis 9. Abstract syntax tree [24].

Selle Abstraktse Süntaksipuule vastab järgmine avaldis:

```
if a > b
   a := a − b
else
   b := b − a
```

Kogu avaldise töötlemist on võimalik kujutada järgmise pildiga.



Joonis 10. Parseri protsessi ülevaade [22].

Selle töö raames *Source String* on avaldis, mis on esitatud konverteerimiseks. Kirjutatud grammatika reeglite järgi toimub parsimine. Kõigepealt toimub sõnavaraline analüüs (*lexical analysis*), mis konverteerib tähemärkide jada Token jadaks. Seejärel toimub süntaktiline analüüs (*syntactic analysis*), mis kontrollib keele struktuuri õigsust. Sama protsessi käigus salvestatakse avaldise komponendid puusse. Pärast puu tekkimist on võimalik seda kompileerida või interpreteerida.

4.3 Teisendus plokkideks

Pärast parsimist tekkinud abstraktne süntaksipuu on valmis töötlemiseks. Iga puu läbi käimine algab puu tipust. Avaldise tüüp ja struktuur määravad seda sama tippu. Kui tegemist on if avaldisega siis tippu pannakse element, mis vastab sellele (vaata ***Joonis 9***). Igale operaatorile süntaksipuus vastab Simulinkis plokk. Näiteks selle elemendi tüübile vastab Simulinkis *switch* plokk, millest on räägitud peatükis *3.2 Avaldiste esitamine Simulinkis*. Plokkide järjekorra ja ühendused määrab operaatorite järjekord süntaksipuus. Koodi põhjal ploki loomine QGeni API abil näeb välja niimoodi:

```
val Block me = GAUtils.createBlock (uniqueName(type), type, parent);
```

Ploki loomiseks antakse talle unikaalne nimetus ja määratakse ploki tüüp. Pärast loomist määratakse ka sisendite (inports) ja väljundite (outports) arv ja luuakse signaalid sisendite ning puus madalamal tasemel olevate operaatorite jaoks loodu plokkide väljundite vahel. Ühte ja sama protseduuri tehakse iga elemendi kohta (ploki tüüp, parameetrid ja sisendite arv sõltuvad operaatori tüübist).

4.3.1 Import Simulinki

QGen kasutab sõltumatut vaheformaati, millega ta sisemiselt teisendab plokidiagramme. Ülalkirjeldatud teisendus loob uue diagrammi selles vaheformaadis. Diagrammi Simulinki teisendamiseks pole vaja teha muud kui käivitada qgeni utiliit

```
qgen_import_xmi('expr.xmi', false, true)
```

MATLAB käsurealt käsu käivitamisel hakkab toimuma andmete importimine.

```
[INFO] Importing model from expr.xmi
```

Importimine lõpeb reaga:

```
[INFO] Imported model expr from expr.xmi
```

Sama kausta kus oli .xmi fail tekib juba uus fail expr.slx. Selle faili avades ilmub ekraanile avaldisest genereeritud Simulink diagramm.

4.4 Alternatiivne võimalus

Autor katsetas ka alternatiivset lahendust avaldiste teisendamiseks ja simulinki diagrammide loomiseks. Sellist võimalust pakub Epsilon[4]. Epsilon on keelte ja vahendite perekond, mis kasutatakse koodi genereerimiseks, modelleerimiseks, mudeli valideerimiseks, võrdlemiseks, migratsiooniks ja refaktoreerimiseks. Epsilon on laiendatav raamistik, mis on võimeline töötlema EMF, UML, Simulink, XML ja teisi mudelitüüpe. Epsiloni keskmes on Epsilon Object Language (EOL) [25].

Pärast Epsiloni tarkvara paigaldamist koostatakse .eol fail projekti konfiguratsiooniga ja sünkroniseeritakse see DSL keele defineeriva projektiga (meie lahenduses Xtext grammatikas genereeritud parser). Tekkinud failis on võimalik kirjutada koodi EOL keeles ja samuti välja kutsuda meetodeid ja klasse, mis on kirjutatud põhiprojektis.

---

[4] https://www.eclipse.org/epsilon/

Epsilon failis kutsutakse välja parsimise meetodit, mis võtab vastu avaldise ja tagastab AST. See tähendab, et Epsilon ei kasuta oma grammatikat ega parserit, vaid kasutab eelnevalt tehtud selle töö jaoks Xtext grammatikat ja parsimise meetoodikat. Epsilon võimaldab kasutada Simulink plokkide teeki otseselt koodi sees. Näiteks *Switch* plokki defineerimine näeb välja niimoodi:

```
var Switch = new `simulink/Signal Routing/Switch`;
```

Teisendus plokkideks käib sama tehnoloogia alusel nagu põhiprojektis. Käiakse abstraktse süntaksipuu läbi ja määratakse igale operaatorile vastava Simulinki plokk. Plokile on samuti võimalik määrata parameetrid, väärtused ja nimetused. Epsiloni kõige suurem erinevus on Simulink diagrammi vastava faili tekkimine. Pärast .eol faili käivitamist tekib kohe .slx fail. See tähendab, et kaob ära QGen kasutamise vajadus.

Proovides kahte erinevat võimalust võin öelda, et mõlemal on palju positiivseid ja ka samuti negatiivseid külgi. Epsilon on päris kiire ja mugav tarkvara. AST läbi käimine, teisendamine, plokkide ühendamine oli tehtud ühes failis ja tulemuseks oli kohe Simulink diagramm. Samas eelnes sellele keeruline konfigureerimine ja Epsiloni Objekti keele süntaks, mille võimalused on ebamugavad võrreldes Java ja Xtendiga. Tänapäeval on väga vähe informatsiooni ja dokumentatsiooni EOL keele kohta, mis teeb selles keeles kirjutamist veel keerulisem.

Xtend võrreldes EOL keelega on palju võimsam ja arusaadavam. Seda keelt on võimalik kasutada erinevates raamistikudes ja on lihtsam rakendada. Importimine Simulinki on ka erinev. Epsilon teisendab avaldise otse .slx failiformaati. QGen API kasutab selleks .xmi vaheformaati. Ühest küljest see nõuab rohkem aega ja ressurssi, teisest küljest .xmi fail on vahetulemus, mida saab kasutada ka teiste rakenduste jaoks. XMI formaat on kergloetav ja on võimalik selle faili abil varakult tuvastada viga.

GitLabi repositooriumi[5] on lisatud EpsilonConfiguration.docx fail, kus on põhjalikult kirjeldatud Epsilon raamistiku paigaldamine ja projektiga konfigureerimine.

[5] https://gitlab.cs.ttu.ee/Roman.Holvason/iapb.git

5 Kokkuvõte

Töö lõppeesmärgiks oli arendada konverter, mis suudaks teisendada kõik valitud avaldised ja tulemus oleks käivitatav Simulinkis. Selle jaoks tehtud programm võtab vastu SPARK programmeerimiskeeles kirjutatud avaldise ja faili nimetuse, kuhu salvestatakse teisenduse tulemus. Keele süntaksi määratlemiseks ja parseri genereerimiseks oli kirjutatud Xtext grammatika. Parsimise käigus avaldise elemendid salvestatakse abstraktse süntaksipuu sisse järgneva rakendamise lihtsustamiseks. Peamine meetod käib läbi puud ja määrab iga tipule vastava Simulink plokki. Plokid ühendatakse signaalidega vastavalt AST struktuurile. Kõik plokid koos nimetustega, signaalid ja väärtused salvestatakse .xmi fail formaati, mis tekib sama kausta kus asub projekt. Simulink diagrammi teisendamiseks on vaja MATLAB käsurealt käivitada tekkinud faili kasutades QGen importimist. Selle tulemuseks tekib fail .slx formaadiga, mis ongi lõplik Simulink diagrammi fail.

5.1 Edasine töö

Selle töö raames oli kasutusel päris väike hulk matemaatilisi operaatoreid. Avaldistes olid kasutusel ainult tavalised operaatorid nagu liitmine, lahutamine, korrutamine ja jagamine. Muidu SPARK programmeerimiskeel ja Simulink plokid võimaldavad kasutada ka keerulisemaid operaatoreid, millest on kirjutatud peatükis *3.1 Toetatud avaldised*. Siin oleks vaja iga uue operaatori jaoks kirjutada grammatikasse vastava reegli. Samuti tulevikus oleks võimalik kirjeldada case_expression, mis on ka tingimuslik avaldis nagu if-avaldis.

IF-avaldise jaoks on vaja ka teha täiendused. Kasutatud selles töös if-avaldised olid If-Then-Else struktuurina. SPARK keeles, nagu teistes keeltes on võimalik kasutada If-Elseif-Else struktuuri, mis vajab grammatika täiendamist.

Epsilon vajab samuti täiendava uurimist, et saada teada selle raamistiku potentsiaali. Praegu mõned avaldised teatud struktuuriga Epsilon ei suuda lõpuni teisendada, kuna autori poolt ei ole veel leitud teatud probleemile sobiv lahendus kasutades EOL keelt.

Töö on osa suuremast projektist, kus avaldisi teisendatakse SysML mudelist Simulinki mudelisse ja sealt edasi programmikoodi. Teisendaja integreerimine raamistikku ei olnud selle töö skoobis ja tehakse hiljem.

Kasutatud kirjandus

[1] Rushby J., The Versatile Synchronous Observer. In: Gheyi R., Naumann D. (eds) Formal Methods: Foundations and Applications. SBMF 2012. Lecture Notes in Computer Science, vol 7498. Springer, Berlin, Heidelberg.

[2] Naks, T., Aiello, M. A., Taft, S.T., „Using SPARK to Ensure System to Software Integrity: A Case Study," *submitted to DeCPS 2019 - Workshop on Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering, Ada-Europe*, Warsaw, Poland., 11-14 June 2019.

[3] „SPARK programming language," [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/SPARK_(programming_language). [Kasutatud 18 04 2019].

[4] „Simulink," [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Simulink. [Kasutatud 20 04 2019].

[5] „se.mathworks.com, Simulink Block," [Võrgumaterjal]. Available: https://se.mathworks.com/help/simulink/gs/simulation_data.png. [Kasutatud 20 04 2019].

[6] Lorenzo Bettini, Implementing Domain-Specific Languages with Xtext and Xtend - Second Edition, Packt Publishing, 2016.

[7] „XText Grammar," [Võrgumaterjal]. Available: https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html. [Kasutatud 21 04 2019].

[8] „Logical Operator," [Võrgumaterjal]. Available: http://docs.adacore.com/live/wave/arm12/html/arm12/arm12-4-5-1.html. [Kasutatud 24 04 2019].

[9] „Relation," [Võrgumaterjal]. Available: http://docs.adacore.com/live/wave/arm12/html/arm12/arm12-4-4.html#S0123. [Kasutatud 24 04 2019].

[10] „Multiplying Operators," [Võrgumaterjal]. Available: http://docs.adacore.com/live/wave/arm12/html/arm12/arm12-4-5-5.html. [Kasutatud 24 04 2019].

[11] „Binary Adding Operators," [Võrgumaterjal]. Available: http://docs.adacore.com/live/wave/arm12/html/arm12/arm12-4-5-3.html. [Kasutatud 24 04 2019].

[12] „Unary Adding Operator," [Võrgumaterjal]. Available: http://docs.adacore.com/live/wave/arm12/html/arm12/arm12-4-5-4.html. [Kasutatud 24 04 2019].

[13] „Literals," [Võrgumaterjal]. Available: http://docs.adacore.com/live/wave/arm12/html/arm12/arm12-4-2.html.

[Kasutatud 24 04 2019].

[14] „Relational Operators,“ [Võrgumaterjal]. Available: http://docs.adacore.com/live/wave/arm12/html/arm12/arm12-4-5-2.html. [Kasutatud 25 04 2019].

[15] „Attribute `Old,“ [Võrgumaterjal]. Available: https://docs.adacore.com/spark2014-docs/html/ug/en/source/specification_features.html. [Kasutatud 25 04 2019].

[16] „Logical Operator Block,“ [Võrgumaterjal]. Available: https://se.mathworks.com/help/simulink/slref/logicaloperator.html. [Kasutatud 28 04 2019].

[17] „Relational Operator Block,“ [Võrgumaterjal]. Available: https://se.mathworks.com/help/simulink/slref/relationaloperator.html. [Kasutatud 28 04 2019].

[18] „Add & Subtract Blocks,“ [Võrgumaterjal]. Available: https://se.mathworks.com/help/simulink/slref/add.html. [Kasutatud 29 04 2019].

[19] „Constant Block,“ [Võrgumaterjal]. Available: https://se.mathworks.com/help/simulink/slref/constant.html. [Kasutatud 29 04 2019].

[20] „Switch Block,“ [Võrgumaterjal]. Available: https://se.mathworks.com/help/simulink/slref/switch.html. [Kasutatud 29 04 2019].

[21] „Parsimine,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Parsimine. [Kasutatud 06 05 2019].

[22] „Parsing,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Parsing. [Kasutatud 13 05 2019].

[23] „Parsing Expressions With Xtext,“ [Võrgumaterjal]. Available: https://typefox.io/parsing-expressions-with-xtext. [Kasutatud 07 05 2019].

[24] „Abstract syntax tree,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Abstract_syntax_tree. [Kasutatud 06 05 2019].

[25] „Epsilon,“ [Võrgumaterjal]. Available: https://www.eclipse.org/epsilon/doc/. [Kasutatud 19 05 2019].

Lisa 1 - XText Grammatika

```
grammar ttu.diploma.expression.Expression with org.eclipse.xtext.common.Terminals

generate expression "http://www.diploma.ttu/expression/Expression"

Model:
	elements += Expression*;

Expression:
	Or;

Or returns Expression:
	And (=> ({Or.left = current} op = OrOp) right = And)*;

OrOp:
	'or'
|	'or' 'else';

And returns Expression:
	Equality (=> ({And.left = current} op = AndOp) right = Equality)*;

AndOp:
	'and'
|	'and' 'then';

Equality returns Expression:
	Comparison (=> ({Equality.left = current} op = ('=' | '/='))
	right = Comparison)?;

Comparison returns Expression:
	PlusOrMinus (=>({Comparison.left = current} op = ('>=' | '>' | '<' | '<='))
	right = PlusOrMinus)?;

PlusOrMinus returns Expression:
	MulOrDiv (=>({Plus.left=current} op = '+' | {Minus.left = current} op = '-')
	right = MulOrDiv)*;

MulOrDiv returns Expression:
	Prefix (=>({Multiplication.lef t= current} op = '*' | {Division.left = current} op
	= '/') right = Prefix)*;

Prefix returns Expression:
	{Not} op = 'not' expression = Prefix | Old;

Old returns Expression:
	Atomic (=> ({Old.expression = current} '`Old'))*;

Atomic returns Expression:
```

```
	Literal |
	{IfThenElse} 'if' condition = Expression 'then' thenStatement = Expression
	(=> 'else' elseStatement = Expression)? |
	{IfThenElse} '(if' condition = Expression 'then' thenStatement = Expression
	(=> 'else' elseStatement  = Expression ')' )? |
	'(' Expression ')';

Literal:
	{IntRef} value = INT |
	{BoolRef} value = ('true' | 'false' | 'TRUE' | 'FALSE') |
	{VariableRef} value = ID;
```