HCL Embedded real-time system with applied machine learning

Embedded systems

Aalborg University

Group SW501E18

Title:

Software 5: "HCL Embedded real-time system with applied machine learning"

AALBORG UNIVERSITY
STUDENT REPORT

Group SW501E18

4. September - 2018

Semester:
5. semester
Project theme:
Embedded Systems
Project duration:
04/09/2018 - 20/12/2018
ECTS:
15
Supervisor:
Thomas Bøgholm

Authors:

Casper Weiss Bang

Daniel Moesgaard Andersen

Frederik Spang Thomsen

Nichlas Ørts Lisby

Thomas Højriis Knudsen

Thomas Lundsgaard Hansen

Abstract:
In the Danish educational system, the Arduino platform is often used to introduce students to computer science. The Arduino's hardware is relatively simple and easy to grasp. However, the Arduino language can often be difficult for novices. While the Arduino language is a simplified subset of `C++`, there are still parts of the language that are difficult to comprehend for novices.
This report elaborates and explains the development of the HCL language, a programming language designed to ease the introduction to programming for students and novices, whilst also implementing high-order functionality on the Arduino. HCL does this by emphasizing a resemblance to the English language. A compiler for HCL was developed, as part of the project. The compiler is written in Kotlin, by hand, and compiles to `C++` code.

Number of pages: 26.

Preface

This report is written during the fifth semester of the bachelor's degree in Software at Aalborg University. It is written for the study board of computer science at the School of Information and Communication Technology. The report is written with the guidance of one supervisor.

The topic of the semester is 'Embedded Systems'.

The system to be developed was chosen by the project group.

Terms and abbreviations used in the report:

HCL	: HCL Compiled Language
NTH	: Nice to Have
NXT	: Lego Mindstorms NXT 2.0
EV3	: Lego Mindstorms EV3

Contents

Introduction 1

some text

initial problem

Analysis 2

2.1 Embedded platform

In the following section will examine the embedded platforms, that will be used in the project. For the 5th semester of the Software education, a Lego Mindstorms kit is supplied to each project group. Therefore the analysis will primarily be based around the Lego Mindstorms platform, with a few additional platforms also considered.

To determine the best platform, the requirements for the project will need to be considered.

1. At the very least, the platform must be able to perform a targeting calculation, based on a machine learning model.
2. Be able to interface with motors, most likely Lego.
3. Potentially be able to train a machine learning model, which is used for the prediction on which the targeting is based.

As image recognition is an expensive computation, the current idea is to offload this to an external server or computer, which will compute the current object position and send that to the embedded device.

2.1.1 Lego Mindstorms

Lego Mindstorms is a programmable computer kit, made by Lego. The platform is based around a central control computer, called the Intelligent Brick. It also includes a variety of sensors, motors and connection cables. Each generation further improves on the concept, adding more computational power and better sensors.

The Lego Mindstorm platform, comes with a graphical drag and drop interface, but a variety of compilers for many different languages exist..

The choices for the 5th semester are the second generation, the Lego Mindstorms NXT 2.0 kit and the third generation, the Lego Mindstorms EV3 kit.

Lego Mindstorms NXT 2.0

The Lego Mindstorms NXT 2.0 (NXT) is the second generation Lego Mindstorms kit. The NXT Intelligent Brick is a micro computer based on the 32-bit ARM7 microprocessor, with 256 kbytes of flash memory and 64 kbytes of RAM[1]. It includes a variety of sensors, such as touch, sound, light and ultrasonic (vision) sensors. On top of this, the consumer kit also includes servo motors and lamps.

Lego Mindstorms EV3

The Lego Mindstorms EV3 (EV3) is the third generation of the Lego Mindstorms line. The EV3 mainly improves on the specifications of the second generation, adding a more powerful 300MHz ARM9 processor, running Linux, 16 MB of flash memory, which is extendable with a microSDHC card and 64MB of RAM[2]. On top of this, it also adds remote control and WiFi capabilities

Language Specification 3

Implementation 4

Bibliography

[1] Lego, “Nxt user guide,” https://www.generationrobots.com/media/Lego-Mindstorms-NXT-Education-Kit.pdf, 2018, [Online; Accessed: 2018-16-09].

[2] ——, “Ev3 user guide,” https://le-www-live-s.legocdn.com/ev3/userguide/1.4.0/ev3_userguide_enus.pdf, 2018, [Online; Accessed: 2018-16-09].

Appendix A

A.1 Aalborg Tekniske Gymnasium Student and Teacher Interview (In danish)

Projektstyring: Analog og digital teknik. Programmerbar teknologi. Næsten ingen programmering i faget. Allerede har haft programmering: Visual basic og C++ - B-niveau: Variabler, typer, kontrolstrukturer.

Projekter: - Cykellygte, - Spilprojekt på arduino, - Motor (Robot), - Censorprojektet, - Eksamensprojekt.

Spørgsmål: - Lave en funktion (opbygning) - Lang tid på fejlsøgning.

Snakker meget for visuel programmering. Giver forståelse for de initielle aspekter.

- Semikolon: Måske - (Men ja) - Typer: Brug begge dele. - Static vs dynamic typed: Så længe det virker. (Fare for at gøre noget uhensigtsmæssigt)

Faget: Ikke så meget forstå - mere bare lave. - Mindre fysik.

Er kurset i høj kurs: Har haft programmering -> vælger dette fag.

Digital design og udvikling (nyt fag, apps, spil, robotdesign)

Nogle skifter hurtigt. Andre har ingen erfaring, men vokser med opgaven.

Generelle hjælpelinjer: - Nem tilkobling og interaktion med hw.

GPIO: LED - DC Motor. Nogle sensorer. Bluetooth.

Duden er hw fan.

==================================

Til elever:

Lidt C# -> Programmering (C#) Meget C# Godt lide computere, men programmering for "boxed" - Ikke skrive forkert. "Hyggeligt når det fungere" Slet ingen programmering. Opnået en del interesse igennem faget. Linjefag. Alle har haft C#. Linjefag. C# Programmering. En rimelige skarp i javascript.

Programmering: - Rigtige funtkionskald, - Semikolon er fint - Det er okay (MEN

PISSE TRÆLS NÅR MAN GLEMMER) - Arrays - 0-indexed forvirrende - arrays forvirrende. - Var ikke brugt. -> auto brugt. - Kender ikke funktionskald (Mange forskellige) - Problemer med typer - (Type inference ville være nemmere) - Men kan godt lide eksplicit type. - Semikolon ikke træls - Skulle være. - Aner intet om typer - Ikke problemer med typer. - C# Let at gå til -> IDE'en. - C vs C# i ide -> De mener at det ville være lige svært i en almindelig text editor. - Loops var svære. Forskel på loops. - Kunne ikke se pointe med metoder. - Nemmere uden typer. Lidt svært at lære typer. - Svært ikke at lære typer til start? - Synes det var fint ikke at forholde sig til typer i start (brugte lua) - Meget udenadslære, ikke tænke sig til hvilke kommandoer der gør hvad. Det kunne være bedre fra IDE'en. - Loops er forvirrende. "For"-navnet er ikke intuitivt. - En med JS synes godt med var. - Lang tid på at lære typerne. - Kunne samle tal typer til num.

Hvad kunne være bedre?: - Dokumentationen kunne være bedre. - Man er vant til overhead -> Så det okay, men lidt tvilende. Men de mener også det er fordi det er det de har lært, og det nok kunne være simplere for begyndere. - Error beskeder lort. - Smart med ikke include. - Kan ikke huske hvordan man laver funktion, men de siger det altid nemt at se hvordan man gjorde sidst. - Bedre dokumentation - Overflod af metoder. - Eksempler til sproget. - Lidt problemer med header og includes. - Scoping er lidt svært. - Nem tilgang til muligt funktions kald. - Godt med afgrænsning fra - En forslår indentation som fra python.

Arduino platformen: - Den er fin, IDE dårlig ikke hjælper. - Minestorm forturkket af en der er "Dårlig" til programmering. - Love it. Det er simpelt. - Arduino er "Dum" gør hvad den bliver bedt om -> godt. - Svært med for mange komponenter, men generelt god læringsplatform. - Godt med hands-on experience. Bedre med noget fysisk, end bare en terminal fra C#. - Svært med hvad forskellige gpio og ports gør. - Ideen med at man får noget til at "Lyse" er fed! - Fejlbeskeder er dårlige. - Editoren burde hjælpe mere.

Læse videre: - Software - Nanoteknologi - Maskinmester - Programmering (Presset mod det.) - Datalog - Bedre sprog kunne have øget interesse. - Maskinmester - En ville lave hjemmesider, men synes programmering ikke var underholdende, fordi læringskurve var for stejl. - Vil gerne hurtigere nå målet. - Sidste gruppe snakker rigtig meget for at et nemmere sprog kunne have øget interessen, og at de er blevet skræmt lidt væk. - En software ingeniør.

- Vores sprog: - Ikke så synligt for dem. Men efter forklaring gav mening.

- Første 3 gik meget hurtigt og klar igennem, 4 var lidt mere besværlig. Hjalp med parenteser.

- Gruppe løste alle 4 opgaver. (Denne gruppe havde IKKE programmeret før)

- Denne gruppe acede det. Stadig en smule problemer med associering.

- Sidste gruppe var rimelig god, men stadig lidt tvivl om associering.

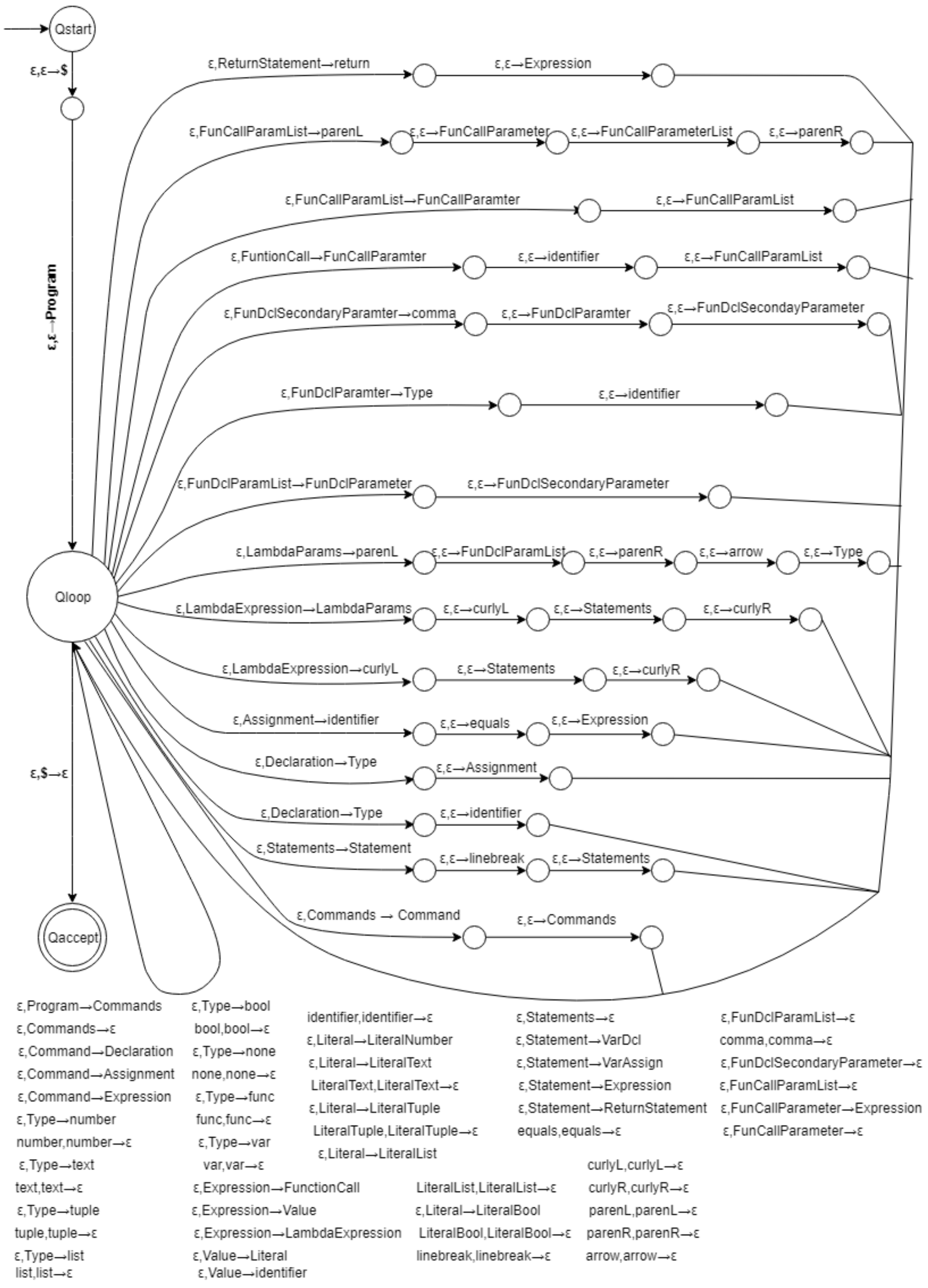
A.2 Extended Backus-Naur Form of HCL grammar

```
           <Program> → <Cmds> $
              <Cmds> → {<Cmd>}
               <Cmd> → <VarDcl> linebreak
                     | <Assign> linebreak
                     | <Expr> linebreak
                     | <ReturnCmd> linebreak
               <Dcl> → <ImplicitType> identifier [equals <DclValue>]
      <ImplicitType> → <Type>
                     | func
                     | var
              <Type> → number
                     | text
                     | tuple sqBracketL [<TypeList>] sqBracketR
                     | list sqBracketL [<Type>] sqBracketR
                     | bool
                     | func sqBracketL [<TypeListNoneAndGenerics>] sqBracketR
                     | none
          <TypeList> → <Type> [comma <TypeList>]
  <TypeListGenerics> → <TypeGenerics> [separator <TypeListGenerics>]
<TypeNoneAndGenerics> →<TypeGenerics>
                     | none
```

```
<TypeListNoneAndGenerics> → <TypeNoneAndGenerics> [separator <TypeListNoneAndGenerics>]
            <TypeGenerics> →<Type>
                           | identifier
                    <Expr> → <FunctionCall>
                           | <Value>
                           | parenL <Expr> parenR
                   <Value> → <Literal>
                           | identifier
                 <Literal> → literalNumber
                           | literalText
                           | literalBool
                           | <LiteralTuple>
                           | <LiteralList>
                  <Values> → <Value> [comma <Values>]
            <LiteralTuple> → parenL <Values> parenR
             <LiteralList> → sqBracketL <Values> sqBracketR
                <DclValue> → <Expr>
                           | <LambdaExpr>
                  <Assign> → identifier equals <DclValue>
              <LambdaExpr> → parenL [<FunDclParams>] parenR colon <TypeNoneAndGenerics> linebreak <LambdaBody>
              <LambdaBody> → curlyL <Cmds> curlyR
```

```
<FunDclParams> → <FunDclParam> [comma <FunDclParams>]
<FunDclParam> → <TypeListGenerics> identifier
<FunctionCall> → identifier
               | <Arg> identifier [<Args>]
<Args> → {<Arg>}+
       | parenL {<Arg>}+ parenR
<Arg> →[colon]<Value>
      | <LambdaExpr>
      | <LambdaBody>
<ReturnCmd> → return <Expr>
```

A.3 NPDA for HCL



A.4 HCL Built-In Functions

Arithmetic Operations

HCL includes five standard arithmetic functions for use on numerical values. The functions are seen in table A.1.

Table A.1: Arithmetic Functions

Operation	Symbol	Input Parameters	Return Type
Addition	+	num, num	num
Subtraction	-	num, num	num
Division	/	num, num	num
Multiplication	*	num, num	num
Modulo	mod	num, num	num

Boolean Operations

Functions for boolean operations are listed in table A.2. These functions compare numeral, textual or boolean values and return a boolean value.

Table A.2: Boolean Functions

Operation	Symbol	Input Parameters	Return Type
Less Than	lessThan	num, num	bool
Less Than Equals	lessThanEqual	num, num	bool
Greater Than	greaterThan	num, num	bool
Greater Than Equals	greaterThanEqual	num, num	bool
Numerical Equals	equals	num, num	bool
Numerical Not Equals	notEquals	num, num	bool
And	and	bool, bool	bool
Or	or	bool, bool	bool
Boolean Equals	equals	bool, bool	bool
Boolean Not Equals	notEquals	bool, bool	bool
Not	not	bool	bool
Textual Equals	equals	txt, txt	bool
Textual Not Equals	notEquals	txt, txt	bool

Text Manipulation
Functions used to manipulate text-objects or convert values to text, seen in table A.3.

Table A.3: Textual Functions

Operation	Symbol	Input Parameters	Return Type
Concat Text	+	txt, txt	txt
Numeric To Text	toText	num	txt
Boolean To Text	toText	bool	txt
Textual To Text	toText	txt	txt
List To Text	toText	list[T]	txt
At index	at	txt, num	txt
Text Concatenation	+	txt, txt	txt
Sub Text	splitAt	txt, num, num	txt
Text Length	lenght	txt	num

Control Structures
Functions used as control structures in HCL, seen in table A.4

Table A.4: Control Structures

Operation	Symbol	Input Parameters	Return Type
Then Statement	then	bool, func[none]	bool
While Loop	while	func[none], bool	bool
Foreach Loop	forEach	list[T], func[none]	none

List Operations
Functions used to manipulate or monitor lists, seen in table A.5.

Table A.5: List Operations

Operation	Symbol	Input Parameters	Return Type
Get Lenght	lenght	list[T]	num
At Index	at	list[T], num	T
Concatenation	+	list[T], List[T]	list[T]
Get Sublist	splitAt	list[T], num, num	list[T]
Map	map	list[T], func[T, T]	list[T]
Filter List	where	list[T], func[T, bool]	list[T]
To List	to	num, num	list[num]
Equals	equals	list[T], list[T]	bool
Not Equals	notEquals	list[T], list[T]	bool

Pin-Functions

Functions used to read and write from arduino's analogue and digital pins, seen in table A.6.

Table A.6: Pin Operations

Operation	**Symbol**	**Input Parameters**	**Return Type**
Set Digital Pin	setDigitalPin	num, bool	none
Read Digital Pin	readDigitalPin	num	num
Write Analog Pin	setAnalogPin	num, num	none
Read Analog Pin	readAnalogPin	num	num
Delay	delay	num	none

Print Operations

Functions used to print to output, seen in table A.7.

Table A.7: Print Functions

Operation	**Symbol**	**Input Parameters**	**Return Type**
Print	print	num/bool/txt	none
Print List	print	list[T]	none

A.5 Standard Library Functions

HCL includes multiple functions as part of its standard library. All standard library functions are implemented exclusively using the HCL syntax and builtin-functions. The following sections

Snippet A.1, returns true if any element in a list upholds the predicate from compareFunc. Returns false if no element in list upholds compare predicate.

Snippet A.1: `any` function.

```
var any (list[T] myList, func[T,bool] compareFunc):bool{
  myList where :compareFunc length greaterThan 0
}
```

Snippet A.2 returns true if all elements in a list upholds the predicate from compareFunc. Returns false if any element does not uphold compare predicate.

Snippet A.2: `all` function.

```
var all (list[T] myList, func[T,bool] compareFunc):bool{
  myList where :compareFunc length equals myList length
}
```

Snippet A.3 returns true if an element is present in a list. Returns false if not present.

Snippet A.3: `in` function.

```
var in = (T element, list[T] myList):bool{
  myList any { value is element }
}
```

Snippet A.4 returns true if an element is not present within a list. Returns false if element is present in list.

Snippet A.4: `notIn` function.

```
var notIn = (T element, list[T] myList):bool{
  element in myList not
}
```

Snippet A.5 is used to create a ternary statement the `else` function. Returns a tuple with types *bool* and *T*

Snippet A.5: `then` function.

```
var then = (bool condition, func[T] body): tuple[bool,T] {
  return (condition,body)
}
```

Snippet A.6 shows `else` function used together with `then` function. Returns function-body dependent on result from `then` function.

Snippet A.6: `else` function.

```
var else = (tuple[bool,T] thenResult, func[T] body): T {
  T output
  thenResult at 0 then { output = thenResult at 1 }
  thenResult at 0 not then { output = body }
  return output
}
```

Snippet A.7 shows `thenElse` function that uses the functionality of both `then` and `else` functions to simulate the classical "if"-statement. Returns function-body dependent on condition-evaluation.

Snippet A.7: `thenElse` function.

```
var thenElse = (bool condition, func[T] trueBody, func[T] falseBody): T {
  T output
  condition then {output = trueBody} else { output = falseBody}
  return output
}
```

A.6 Semantics Derivation Tree Example

This is an example of how to create a derivation tree for a program written in HCL. It is based on the following code snippet.

```
num x
x = 5
```

Using the abstract syntax described in section **??**, the code can be written as the following big step semantic:

$$env_V, env_F \vdash \langle num\ x;\ \ x = 5, env_V, env_F, sto\rangle \rightarrow_{stm} (sto', env'_V, env'_F)$$

Using the Compositional Big Step Semantic rule, the following derivation is produced:

$$\frac{env_V, env_F \vdash \langle num\ x, env_V, env_F, sto\rangle \rightarrow_{stm} (sto'', env''_V, env''_F) \quad env''_V, env''_F \vdash \langle x = 5, env''_V, env''_F, sto''\rangle \rightarrow_{stm} (sto', env'_V, env'_F)}{env_V, env_F \vdash \langle num\ x;\ \ x = 5, env_V, env_F, sto\rangle \rightarrow_{stm} (sto', env'_V, env'_F)}$$

Table A.8 shows how the program state changes after each statement in the code is run.

Table A.8: Table of program states after each statement.

	Abstract program state	Concrete program state
Initial state	(sto, env_V, env_F)	(sto, env_V, env_F)
After first statement	$(sto'', env''_V, env''_F)$	$(sto[l \mapsto 0], env_V[x \mapsto l], env_F)$
After second statement	(sto', env'_V, env'_F)	$(sto[l \mapsto 5], env_V[x \mapsto l], env_F)$

A.7 Type Rule Derivation Tree Example

This is an example of how to create a type derivation tree for a program written in HCL. It is based on the following code snippet.

```
num x = 5
bool b = x greaterThan 0
b print
```

The program calls two functions, *greaterThan* and *print*. They are declared with the following types:

```
func[num, num, bool] greaterThan # Two number parameters, returns bool
func[bool, none] print # overloaded to take a bool parameter, returns nothing
```

Following the type rules specified in section **??**, this derivation tree will be produced:

$$\dfrac{\dfrac{E \vdash 5 : num}{E \vdash num\ x = 5 : \texttt{ok}} \quad \dfrac{\dfrac{\dfrac{E(x) : num \quad E \vdash 0 : num}{E \vdash x\ greaterThan\ 0 : bool}}{E \vdash bool\ b = x\ greaterThan\ 0 : \texttt{ok}} \quad \dfrac{E \vdash b : bool}{E \vdash b\ print : (none, \texttt{ok})}}{E \vdash bool\ b = x\ greaterThan\ 0;\ b\ print : \texttt{ok}}}{E \vdash num\ x = 5;\ bool\ b = x\ greaterThan\ 0;\ b\ print : \texttt{ok}}$$