Verifying the FreeRTOS
Real-Time Operating System

ICTAC Summer School

Jim Woodcock

University of York

23 August 2011
Outline

Introduction

Glossary of Notation

Example Z Specification

Refinement
Outline

Introduction

Glossary of Notation

Example Z Specification

Refinement
Verifying FreeRTOS

1. Background to FreeRTOS – aims of verification exercise – data refinement in Z (Jim)

2. What FreeRTOS is expected to do and how it works (Deepak)
   - concrete context on ARM processor
   - how a user must use FreeRTOS
   - simulator demo of FreeRTOS
   - scheduling policy in FreeRTOS
   - other functionality (interrupts, context switching)
   - implementation architecture of FreeRTOS

3. Strategy for verifying FreeRTOS (Deepak)
   - separate scheduling and non-scheduling related specifications
   - check non-scheduling related specs for particular processor by hand, or other machine-assisted techniques

4. Scheduler
   - detailed top-level spec: valid sequences of API calls (Deepak)
   - Z model of scheduler (Jim)

5. Theory of dynamic frames for handling pointers (Jim)
Background to FreeRTOS

ICTAC Summer School

Jim Woodcock

University of York

23 August 2011
UK Grand Challenges in Computing

- UKCRC: UK Computing Research Committee (Hoare/Milner)
  GC0  Grand Challenges Committee
  GC1  In Vivo – in Silico
  GC2  Ubiquitous Computing — Science and Design
  GC3  Memories for Life
  GC5  Architecture of Brain and Mind
  GC6  Dependable Systems Evolution
  GC7  Journeys in Non-Classical Computation

- International Verified Software Initiative

- Hoare, Misra, Shankar, O’Hearn

- VSTTE conferences & workshops

- many journal papers
The Verified Software Initiative

- industrial software usually has extensive documentation
- ... but software behaviour is often a complete surprise!
- programing is hard
- reduce problem to rules that can be blindly followed by computer
- components interact and interfere
- undesirable properties emerge
- systems fail to satisfy their users’ needs
- hard to define requirements, foresee interactions, add new functions
- documentation: usually lots of text, pictures, diagrams
- often imprecise and ambiguous
- important information often hidden by irrelevant detail
- design mistakes often discovered too late, making them expensive or even impossible to correct
- tribute to skill of software engineers that systems work at all
Another Way?

- practitioners and researchers believe it’s now practical to use formal methods to produce software, even non-critical software...
- ...and that this will turn out to be the cheapest way to do it
- formal methods tools are already transforming industrial practice in software engineering
- who said this?

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification, we’re building tools that can do actual proof about the software and how it works in order to guarantee reliability.

- which company gives a five-year warranty for the correctness of its software?
Science & Engineering

- a mature scientific discipline should set its own agenda and pursue ideals of purity, generality, and accuracy far beyond current needs
- science explains why things work in full generality by means of calculation and experiment
- engineering exploits scientific principles in the study of the specification, design, construction, and production of working artifacts, and improvements to both process and design
- the verification challenge is to achieve a significant body of verified programs that have precise external specifications, complete internal specifications, machine-checked proofs of correctness with respect to a sound theory of programming
VSI Deliverables

1. A comprehensive theory of programming
   ▶ all features needed to build practical and reliable programs

2. A coherent toolset
   ▶ automating the theory and scaling up to large codes

3. A collection of verified programs
   ▶ replacing existing unverified ones
   ▶ continuing to evolve as verified code
   ▶ a repository


Tony Hoare:

*You can’t say any more it can’t be done!*

*Here, we’ve done it!*
FreeRTOS: A Challenge for Formal Methods

- Woodcock & Barry 2008
- be the first to verify FreeRTOS
- one possible approach: a community effort
  - refinement of Free RTOS, correct by construction
- challenges (something for everybody!)
  - Theory
    - total correctness in pointer logic
    - pointer logic and refinement
  - Tools
    - integrate pointer logic into B and Event-B: Separation-B
    - link Pointer-B and Smallfoot
    - automatically generate pointer programs from specifications
- Experiments
  - verify the refinement of a wide range of pointer programs
  - verify the refinement of Free RTOS
FreeRTOS Formal Specification and Verification

- first attempts:
  - Déharbe: B specification
  - Lin, Freitas, Woodcock: Z specification
  - Pronk: Promela/SPIN
  - Ferreira: VDM
  - Qin: abduction-based shape analysis
  - Woodcock: dynamic frames
  - Jacob: Alloy modelling
  - IISc work: finite-state transducers, specifications

- all specify current code base

- fresh attempts:
  - Abrial: informal requirements
  - Abrial: Event-B
  - Cheng, Woodcock: Z specification
  - Mistry: FreeRTOS-multicore
  - Lamsam: VeriFast analysis
What does FreeRTOS really do?

- lightweight, embeddable, multi-tasking, real-time OS
- key assumption: target system has a single processing unit
- library of types and functions to build microkernels
- C + assembly, ported to most embedded systems architectures
- very small kernel produced for target microcontrollers (4–9kB)
- provides services for embedded programming
  - tasks, communication and synchronisation, memory management, real-time events, I/O-device control
- code divided in two:
  1. boot phase: set up tasks and communication channels
  2. application execution phase: schedule and execute tasks
- prioritised scheduling: running process has highest priority
- preemptive scheduling: running process may be interrupted
- cooperative scheduling: processes decide interruption points
What can we try to prove?

- see Pronk’s feasibility study
- functional correctness:
  - reconstruct specification and verify compliance
- structural properties:
  - null-pointer dereferencing
  - memory-allocation problems (e.g., memory leaks)
  - unspecified language issues (e.g., expression evaluation order)
  - arithmetic problems (e.g., overflow, underflow)
- timing properties:
  - tasks scheduled on time \((t \pm \delta t)\)
  - interrupt latency time (time from interrupt to handling)
- liveness properties:
  - deadlock freedom for scheduling mechanism
  - mutual exclusion guarantees
  - scheduling fairness
Task functionality

- Suspending a task with `vTaskSuspend()`
- Resuming a task with `vTaskResume()`
- Blocking API function called
- Event
- Blocking API function called
Task states

- **running**: currently executing task
- **ready**:
  - candidate for scheduling
  - running task has higher or equal priority
  - scheduler must choose next ready task with maximum priority
- **blocked**:
  - task waiting on an event
  - temporal delay or external user interaction
- **suspended**:
  - neither running, ready, nor blocked
  - cannot be executed until it becomes ready again
- **idle task**:
  - systematically created in every application
  - lowest priority
  - may have **idle-task hook**
    - e.g., to put device in low power mode
Co-routine functionality

- prioritised cooperative scheduling
- shared single stack to reduce global RAM usage

- **running**: currently executing co-routine
- **ready**: candidate for scheduling
- **blocked**: waiting on an event
- co-routines are prioritised
- all tasks have higher priority
Intertask communication

- queues:
  - FIFO buffers accessed by tasks
  - fixed-size items, fixed-capacity queue
  - default copy semantics, no protection for pointers
  - tasks can wait for an insertion/deletion or for space
  - task synchronisation technique

- semaphores:
  - synchronisation between tasks and interrupts
  - singleton-or-empty queues, content irrelevant
  - P-operation: task empties queue (may block)
  - V-operation: interrupt fills queue

- mutexes:
  - binary semaphore for shared resource
  - associated queue initially full
FreeRTOS real-time aspects

- Microcontroller internal timer generates periodic interrupts
- Allows FreeRTOS to count time
  - Delay mechanism and task preemption
- Default configuration:
  - Source clock 32,768 Hz
  - FreeRTOSConfig.h file specified desired tick rate, say 1kHz
  - TimerA CCR register: $32,768/1,024 = 32$ for 1,024Hz tick rate
- Division rounding may result in erroneous values
Memory Organisation

- kernel allocates memory dynamically for tasks and queues
- compile-time declaration: byte array for heap storage
- heap contains task execution stacks and queues
- remaining RAM holds all global and static variables
- total RAM space typically 10kB
- “Goldilocks configuration”: for static heap allocation
  - not too big, not too small
  - if the heap is too big, program may not compile
  - if it’s too small, task stacks can’t be allocated at run-time
- implies additional correctness criteria
The Z Notation

- model-oriented specification language
- state-based specification languages:
  - Alloy, ASM, B, Event-B, Circus, PerfectDeveloper, RAISE, VDM
  - equivalent in expressive and analytical power
- model-oriented method:
  - build an abstract model of a system to formalise requirements
  - take design decisions to produce a concrete representation
  - prove concrete model refines abstract model
  - repeat until implementable in programming language
  - every concrete behaviour is permitted
  - no test can distinguish concrete model from abstract one
  - contracts: correctness by construction
Specifications

Schema box \begin{schema}{Name}{Params}
\begin{center}
Name
\[\text{Name}\]
\end{center}
\end{schema}

Declarations \begin{schema}{Name}{Params}
\begin{center}
\text{Declarations}
\end{center}
\end{schema}

Predicates \begin{schema}{Name}{Params}
\begin{center}
\text{Predicates}
\end{center}
\end{schema}

Axiomatic description \begin{axdef}
\begin{center}
\text{Declarations}
\end{center}
\end{axdef}

Predicates \begin{axdef}
\begin{center}
\text{Predicates}
\end{center}
\end{axdef}

Generic definition \begin{gdef}{Name}{Params}
\begin{center}
\text{Declarations}
\end{center}
\end{gdef}

Predicates \begin{gdef}{Name}{Params}
\begin{center}
\text{Predicates}
\end{center}
\end{gdef}

\begin{zed}
\begin{center}
\text{Basic type definition}
\end{center}
\end{zed}

\text{Basic type definition}

\begin{gdef}{NAME}{DATE}
\begin{center}
\text{NAME}
\end{center}
\end{gdef}

Abbreviation definition

\text{DOC} \leftrightarrow \text{seq CHAR} \quad \text{DOC} \leftrightarrow \text{seq CHAR}

Constraint

n\_disks < 5

Schema definition

Point \equiv [x, y : Z] \quad \text{Point} \defns \{x, y : \text{num}\}

Free type definition

Ans ::= ok(\{Z\}) \mid \text{error} \quad \text{Ans} ::= \text{ok} \\text{\textbackslash data\text\num\\text\rdata}

\end{zed}

Logic and schema calculus

true, false \quad \text{true, false}

\neg P \quad \\text{\textbackslash lnot} P

P \land Q \quad P \\land \text{Q}

P \lor Q \quad P \lor \text{Q}

P \Rightarrow Q \quad P \Rightarrow \text{Q}

\forall x : T \mid P \cdot Q \quad \forall \text{forall} \ldots

\exists x : T \mid P \cdot Q \quad \exists \text{exists} \ldots

\exists_1 x : T \mid P \cdot Q \quad \exists_1 \text{exists} \ldots

\bigcup A \quad \exists \text{bigcup} A

\bigcap A \quad \exists \text{bigcap} A

F \ X \quad F \text{\_X}

\{x_1, \ldots, x_n\} \quad \{x_1, \ldots, x_n\}

\emptyset \quad \emptyset

\begin{aligned}
S \subseteq T & \quad S \subseteq T \\
S \setminus T & \quad S \setminus T \\
S \cup T & \quad S \cup T \\
S \cap T & \quad S \cap T \\
S \setminus T & \quad S \setminus T \\
\bigcup A & \quad \exists \text{\bigcup} A \\
\bigcap A & \quad \exists \text{\bigcap} A \\
F \ X & \quad F \_X \\
F_1 X & \quad F_1 \text{\_X}
\end{aligned}

\text{Set display}

\text{Set comprehension}

\emptyset \quad \text{Empty set}

S \subseteq T \quad S \subseteq T

S \setminus T \quad S \setminus T

S \cup T \quad S \cup T

S \cap T \quad S \cap T

S \setminus T \quad S \setminus T

\bigcup A \quad \exists \text{\bigcup} A

\bigcap A \quad \exists \text{\bigcap} A

F \ X \quad F \_X

F_1 X \quad F_1 \_X

\text{Finite sets}

\text{Finite sets} \neq \emptyset

Basic expressions

\begin{center}
x = y \quad x = y
\end{center}

Equality

\begin{center}
x \neq y \quad x \neq y
\end{center}

Inequality

\begin{center}
\text{if} \ P \ \text{then} \ E_1 \ \text{else} \ E_2
\end{center}

Conditional expression

\begin{center}
\theta \ E_1 \\
\text{where} \ \theta
\end{center}

Expression

\begin{center}
\mu x \mid T \mid P \cdot E
\end{center}

Selection

\begin{center}
\text{let} \ x = \text{E}_1 \text{\_E}_2
\end{center}

Let-expression

\begin{center}
\bigcup_1 X
\end{center}

Finite sets

Relations

\begin{center}
X \leftrightarrow Y
\end{center}

Binary relations

\begin{center}
x \leftrightarrow y
\end{center}

Maplet

\begin{center}
dom R \quad \text{\textbackslash dom} \ R
\end{center}

Domain

\begin{center}
ran R \quad \text{\textbackslash ran} \ R
\end{center}

Range

\begin{center}
id X \quad \text{\textbackslash id} \ X
\end{center}

Identity relation

\begin{center}
Q \subseteq R \quad Q \subseteq R
\end{center}

Composition

\begin{center}
Q \setminus R \quad Q \setminus R
\end{center}

Backwards comp.

\begin{center}
S \dres R \quad S \dres R
\end{center}

Domain restriction

\begin{center}
R \setminus S \quad R \setminus S
\end{center}

Range restriction

\begin{center}
S \dres R \quad S \dres R
\end{center}

Domain anti-res.

\begin{center}
R \setminus S \quad R \setminus S
\end{center}

Range anti-restr.

\begin{center}
R \setminus S \quad R \setminus S
\end{center}

Relational inverse

\begin{center}
R \setminus S \quad R \setminus S
\end{center}

Relational image

\begin{center}
Q \setminus R \quad Q \setminus R
\end{center}

Oversetting

\begin{center}
R^{*} \quad R^{*}
\end{center}

Iteration

\begin{center}
R^{*} \quad R^{*}
\end{center}

Transitive closure

\begin{center}
R^{*} \quad R^{*}
\end{center}

Refl. – trans. closure
Functions

\[ f(x) \]
\[ (\lambda x: T \mid P \bullet E) \]
\[ X \rightarrow Y \]
\[ X \rightarrow Y \]
\[ X \rightarrow Y \]
\[ X \rightarrow Y \]
\[ X \rightarrow Y \]
\[ X \rightarrow Y \]
\[ X \rightarrow Y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]
\[ x \rightarrow y \]

Numbers and arithmetic

\[ \mathbb{N} \]
\[ \mathbb{Z} \]
\[ + - \ast \div \mod \]
\[ < \leq \geq > \]
\[ \mathbb{N}_1 \]
\[ \text{succ} \]
\[ m \ldots n \]
\[ \# S \]
\[ \text{min} S \]
\[ \text{max} S \]

Sequences

\[ \text{seq} X \]
\[ \text{seq}_1 X \]
\[ \text{iseq} X \]
\[ \langle x_1, \ldots, x_n \rangle \]
\[ s^\ast t \]
\[ \text{rev} s \]
\[ \text{head} s \]
\[ \text{last} s \]
\[ \text{tail} s \]

Bags

\[ \text{bag} X \]
\[ \langle x_1, \ldots, x_n \rangle \]
\[ \text{count} B x \]
\[ B \times x \]
\[ n \otimes B \]
\[ x \inbag B \]
\[ B \inbag C \]
\[ B \cup C \]
\[ \text{items} s \]

\textbf{fuzz flags}

Usage: \texttt{fuzz [-aqstv] [-p prelude] [file ...]}

- \texttt{-a} Don't use type abbreviations
- \texttt{-p prelude} Use prelude in place of the standard one
- \texttt{-q} Implicit quantifiers
- \texttt{-d} Dependency analysis
- \texttt{-s} Syntax check only
- \texttt{-t} Report types of global definitions
- \texttt{-v} Echo formal text as it is parsed
Outline

Introduction

Glossary of Notation

Example Z Specification

Refinement
An Information System for a Theatre Box Office

- database for one performance at a theatre
- keep track of seat allocations
- provide the following robust operations
  - initialise with a particular allocation
  - purchase a ticket
  - return a ticket
- refunds are given only to the original purchaser
- only one person can sit in a particular seat
- a customer can buy many tickets
System state

Given sets

\([SEAT, CUSTOMER]\)

State schema

\[
\begin{align*}
BoxOffice & \\
seating : \text{F} & SEAT \\
sold : SEAT \rightarrow CUSTOMER
\end{align*}
\]

\[
\text{dom } sold \subseteq seating
\]

System State Initialisation

\[
\begin{align*}
BoxOfficeInit & \\
BoxOffice' & \\
allocation? : \text{F} & SEAT \\
seating' &= allocation? \\
sold' &= \emptyset
\end{align*}
\]
Correctness

**Theorem** Initialisation

\[ \forall \text{ allocation? : F SEAT} \bullet \]
\[ \exists \text{ seating'} : F SEAT; \text{ sold'} : SEAT \rightarrow CUSTOMER \bullet \]

BoxOfficeInit
Purchasing a ticket

Operation schema

\[\text{Purchase0} \]
\[\Delta \text{BoxOffice} \]
\[s? : \text{SEAT} \]
\[c? : \text{CUSTOMER} \]

\[s? \in \text{seating} \setminus \text{dom sold} \]
\[\text{sold'} = \text{sold} \cup \{(s? \mapsto c?)\} \]
\[\text{seating'} = \text{seating} \]
Correctness

**Theorem** Applicability

∀ s? : SEAT; c? : CUSTOMER; BoxOffice •

s? ∈ seating \ dom sold ⇒ pre Purchase0
Handling errors

Free type

\[ \text{Response ::= okay | notAvailable} \]

SEAT not available

\[
\text{PurchaseError} \\
\exists \text{BoxOffice} \\
s? : \text{SEAT} \\
r! : \text{Response}
\]

\[
s? \notin \text{seating} \setminus \text{dom sold} \\
r! = \text{notAvailable}
\]
Making a robust operation

Dealing with success

\[
\text{Success} \\
\text{r! : Response} \\
\text{r! = okay}
\]

Putting it all together

\[
\text{Purchase } \equiv \text{Purchase0 } \land \text{Success } \lor \text{PurchaseError}
\]
Exercise

Return

$\Delta BoxOffice$

c? : CUSTOMER

s? : SEAT
Implementation

- how would you implement the box office specification?
- C++, C#, Visual Basic .NET, and Java don’t provide inherent language features for working with sets
- idea: create a set class using ArrayList to hold elements
- how do we do set union of $S_1$ and $S_2$?
  1. create new set class $T$ to hold $S_1 \cup S_2$
  2. iterate through elements of $S_1$ adding it to $T$
  3. iterate through elements of $S_2$: if element doesn't already exist in $T$, then add it
- how many steps would performing the union take?
- quadratic running time for union of $O(m \times n)$
- Hashtable or Dictionary would be better choice
- refinement:
  - choosing correct, efficient data structure representation
Outline

Introduction

Glossary of Notation

Example Z Specification

Refinement
Retrieve relation

<table>
<thead>
<tr>
<th>Retrieve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract State</td>
</tr>
<tr>
<td>Concrete State</td>
</tr>
</tbody>
</table>

relationship

Proof obligations

Installation

\[ \forall C' \ | \ CI \ \bullet \ (\exists A' \ | \ AI \ \bullet \ R') \]

Preconditions

\[ \forall A; \ C \ | \ pre \ AO \ \land \ R' \ \bullet \ pre \ CO \]

Correctness

\[ \forall A; \ C ; \ C' \ | \ pre \ AO \ \land \ R \ \land \ CO \ \bullet \ (\exists A' \ \bullet \ R' \ \land \ AO) \]
A building entry system

:\[ Staff : \]

\[ \text{maxentry} : \mathbb{N} \]

Abstract system

\[ ASystem \triangleq [ s : \mathcal{P} \text{Staff} \mid \#s \leq \text{maxentry} ] \]

\[ ASystemInit \triangleq [ ASystem' \mid s' = \emptyset ] \]

\( \text{AEnterBuilding} \)
\( \Delta ASystem \)
\( p? : \text{Staff} \)

\( \#s < \text{maxentry} \)
\( p? \notin s \)
\( s' = s \cup \{ p? \} \)

\( \text{ALeaveBuilding} \)
\( \Delta ASystem \)
\( p? : \text{Staff} \)

\( p? \in s \)
\( s' = s \setminus \{ p? \} \)
Concrete system

\[ CSystem \triangleq [\ l : \text{iseq} \ Staff \ | \ \#l \leq \text{maxentry} ] \]

\[ CSystemInit \triangleq [\ CSystem' \ | \ l' = \emptyset ] \]

\underline{CEnterBuilding}

\[ \Delta CSystem \]

\[
p? : \ Staff
\]

\#l < \text{maxentry}

\[
p? \notin \text{ran} \ l
\]

\[
l' = l \setminus \langle p? \rangle
\]

\underline{CLeaveBuilding}

\[ \Delta CSystem \]

\[
p? : \ Staff
\]

\[
p? \in \text{ran} \ l
\]

\[
l' = l \upharpoonright (\text{Staff} \setminus \{p?\})
\]
Refinement

\[ \text{ListRetrieveSet} \]
\[ \text{ASystem} \]
\[ \text{CSystem} \]

\[ s = \text{ran } l \]
A mean machine

\[ AMemory \equiv [ s : \text{seq} \mathbb{N} ] \]

\[ AMemoryInit \equiv [ AMemory' \mid s' = \langle \rangle ] \]

\[ A\text{Enter} \quad ∆ AMemory \]
\[ n? : \mathbb{N} \]
\[ s' = s \wedge \langle n? \rangle \]

\[ AMean \quad \Xi AMemory \]
\[ m! : R \]
\[ s \neq \langle \rangle \]
\[ m! = \frac{\sum_{i=1}^{\#s} s(i)}{\#s} \]
$CMemory \triangleq [\text{sum} : \mathbb{N}; \text{size} : \mathbb{N}]$

$InitCMemory \triangleq [CMemory' | \text{sum}' = 0 \land \text{size}' = 0]$

\[
\begin{align*}
\text{CEnter} \quad & \quad \Delta CMemory \\
\quad & \quad n? : \mathbb{N} \\
\text{sum}' & = \text{sum} + n? \\
\text{size}' & = \text{size} + 1
\end{align*}
\]

\[
\begin{align*}
\text{CMean} \quad & \quad \Xi CMemory \\
\quad & \quad m! : \mathbb{R} \\
\text{size} & \neq 0 \\
m! & = \frac{\text{sum}}{\text{size}}
\end{align*}
\]
Retrieve relation

\[ \text{SumSizeRetrieve} \]
\[ \text{AMemory} \]
\[ \text{CMemory} \]

\[
\text{sum} = \sum_{i=1}^{\#s} s(i)
\]

\[
\text{size} = \#s
\]
Dictionary

\( ADict \triangleq [\ ad : \mathbb{P} \ Word ] \)

\( CDict_1 \)

\( cd_1 : \text{iseq Word} \)

\( \forall \ i, j : \text{dom} \ cd_1 \mid i \leq j \bullet (cd_1 \ i) \leq_W (cd_1 \ j) \)

\( CDict_2 \)

\( cd_2 : \text{seq}(\mathbb{P} \ Word) \)

\( \forall \ i : \text{dom} \ cd_2 \bullet \forall \ w : (cd_2 \ i) \bullet \#w = i \)

Word trees

\( \text{WordTree} ::= \text{tree} \langle \langle \text{Letter} \rightarrow_1 \text{WordTree} \rangle \rangle \)

\( \mid \text{treeNode} \langle \langle \text{Letter} \rightarrow \text{WordTree} \rangle \rangle \)

\( CDict_3 \triangleq [\ cd_3 : \text{WordTree} ] \)
Example

\[
\text{tree}\{ a \mapsto \text{tree}\{ n \mapsto \text{tree}\{ d \mapsto \text{treeNode} \emptyset, t \mapsto \text{treeNode} \emptyset\}\}\}, \\
b \mapsto \text{tree}\{ e \mapsto \text{tree}\{ e \mapsto \text{treeNode} \emptyset\}\}, \\
c \mapsto \text{tree}\{ a \mapsto \text{tree}\{ n \mapsto \text{treeNode} \emptyset, t \mapsto \text{treeNode} \emptyset\}\} \}
\]
Example

\[
\text{tree}\{t \mapsto \text{tree}\{i \mapsto \text{tree}\{n \mapsto \text{treeNode}\{y \mapsto \text{treeNode}\emptyset}\}\}\}\}\}
\]