Theory Design

data theory

program theory

Theory Design

data theory

$$push\ s\ x$$

program theory

Theory Design

data theory

$s := push\ s\ x$

program theory

Theory Design

data theory

$s := push\ s\ x$

program theory

$push\ x$

Theory Design

data theory

$s := push\ s\ x$

program theory

$push\ x$

user's variables, implementer's variables

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'=x \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'=x \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

ok

$\Leftarrow \quad push\ x.\ pop$

Program-Stack Theory

syntax

$push$	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'=x \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

ok

$\Leftarrow \quad push\ x.\ pop$

$= \quad push\ x.\ ok.\ pop$

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'=x \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

ok

$\Leftarrow \quad push\ x.\ pop$

$= \quad push\ x.\ ok.\ pop$

$\Leftarrow \quad push\ x.\ push\ y.\ pop.\ pop$

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'=x \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'{=}x \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

$top'{=}x$

$\Leftarrow \quad push\ x$

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'=x \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

$top'=x$

$\Leftarrow \quad push\ x.\ ok$

Program-Stack Theory

syntax

push	a procedure with parameter of type X
pop	a program
top	expression of type X

axioms

$top'=x \;\Leftarrow\; push\; x$

$ok \;\Leftarrow\; push\; x.\; pop$

$top'=x$

$\Leftarrow \quad push\; x.\; ok$

$\Leftarrow \quad push\; x.\; push\; y.\; push\; z.\; pop.\; pop$

Program-Stack Implementation

var s: [*X] implementer's variable

Program-Stack Implementation

var s: [$*X$] implementer's variable

$push = \langle x: X\cdot\ s:= s;;[x]\rangle$

Program-Stack Implementation

var s: [$*X$] implementer's variable

$push \;=\; \langle x{:}\ X\cdot\ \ s{:}{=}\ s;;[x]\rangle$

$pop \;=\; s{:}{=}\ s\ [0;..\#s-1]$

Program-Stack Implementation

var s: $[*X]$ implementer's variable

$push = \langle x: X \cdot\ s := s;;[x] \rangle$

$pop = s := s\ [0;..\#s-1]$

$top = s\ (\#s-1)$

Program-Stack Implementation

$\textbf{var}\ s\colon [*X]$ implementer's variable

$push\ =\ \langle x\colon X\cdot\ s{:=}\ s;;[x]\rangle$

$pop\ =\ s{:=}\ s\ [0;..\#s-1]$

$top\ =\ s\ (\#s-1)$

Proof (first axiom):

	$(\ top'{=}x\ \Leftarrow\ push\ x\ )$	definitions of *push* and *top*
$=$	$(\ s'(\#s'-1){=}x\ \Leftarrow\ s{:=}\ s;;[x]\ )$	rewrite assignment with one variable
$=$	$(\ s'(\#s'-1){=}x\ \Leftarrow\ s'=s;;[x]\ )$	List Theory
$=$	$\top$	

Program-Stack Implementation

var s: [*X] implementer's variable

$push = \langle x: X \cdot\ s := s;;[x] \rangle$

$pop = s := s\ [0;..\#s-1]$

$top = s\ (\#s-1)$

Proof (first axiom):

	$(\ top'=x \ \Leftarrow\ push\ x\ )$	definitions of *push* and *top*
$=$	$(\ s'(\#s'-1)=x \ \Leftarrow\ s:= s;;[x]\ )$	rewrite assignment with one variable
$=$	$(\ s'(\#s'-1)=x \ \Leftarrow\ s' = s;;[x]\ )$	List Theory
$=$	$\top$	

consistent? yes, implemented.

Program-Stack Implementation

var s: [*X] implementer's variable

$push = \langle x: X \cdot\ s := s;;[x] \rangle$

$pop = s := s\ [0;..\#s-1]$

$top = s\ (\#s-1)$

Proof (first axiom):

	$(\ top'=x \ \Leftarrow\ push\ x\ )$	definitions of *push* and *top*
$=$	$(\ s'(\#s'-1)=x \ \Leftarrow\ s:= s;;[x]\ )$	rewrite assignment with one variable
$=$	$(\ s'(\#s'-1)=x \ \Leftarrow\ s' = s;;[x]\ )$	List Theory
$=$	$\top$	

consistent? yes, implemented.

complete? no, we can prove very little if we start with *pop*

Fancy Program-Stack Theory

$top' = x \land \neg isempty' \Leftarrow push\ x$

$ok \Leftarrow push\ x.\ pop$

$isempty' \Leftarrow mkempty$

Fancy Program-Stack Theory

$top'=x \land \neg isempty' \quad \Leftarrow \quad push\ x$

$ok \quad \Leftarrow \quad push\ x.\ pop$

$isempty' \quad \Leftarrow \quad mkempty$

Fancy Program-Stack Theory

$top'{=}x \land \neg isempty' \quad \Leftarrow \quad push\ x$

$ok \quad \Leftarrow \quad push\ x.\ \ pop$

$isempty' \quad \Leftarrow \quad mkempty$

Fancy Program-Stack Theory

$top'{=}x \wedge \neg isempty' \;\Leftarrow\; push\ x$

$ok \;\Leftarrow\; push\ x.\ pop$

$isempty' \;\Leftarrow\; mkempty$

↑

Weak Program-Stack Theory

$top'{=}x \quad \Leftarrow \quad push\ x$

$top'{=}top \quad \Leftarrow \quad balance$

$balance \quad \Leftarrow \quad ok$

$balance \quad \Leftarrow \quad push\ x.\ \ balance.\ \ pop$

Weak Program-Stack Theory

$top'{=}x \;\Leftarrow\; push\ x$

$top'{=}top \;\Leftarrow\; balance$

$balance \;\Leftarrow\; ok$

$balance \;\Leftarrow\; push\ x.\ balance.\ pop$

$count' = 0 \;\Leftarrow\; start$

$count' = count{+}1 \;\Leftarrow\; push\ x$

$count' = count{+}1 \;\Leftarrow\; pop$

Program-Queue Theory

$$isemptyq' \Leftarrow mkemptyq$$

$$isemptyq \Rightarrow front'=x \land \neg isemptyq' \Leftarrow join\ x$$

$$\neg isemptyq \Rightarrow front'=front \land \neg isemptyq' \Leftarrow join\ x$$

$$isemptyq \Rightarrow (join\ x.\ leave = mkemptyq)$$

$$\neg isemptyq \Rightarrow (join\ x.\ leave = leave.\ join\ x)$$

Program-Queue Theory

$$isemptyq' \;\Leftarrow\; mkemptyq \qquad \longleftarrow$$

$$isemptyq \;\Rightarrow\; front'{=}x \wedge \neg isemptyq' \;\Leftarrow\; join\ x$$

$$\neg isemptyq \;\Rightarrow\; front'{=}front \wedge \neg isemptyq' \;\Leftarrow\; join\ x$$

$$isemptyq \Rightarrow (join\ x.\ leave \;=\; mkemptyq)$$

$$\neg isemptyq \Rightarrow (join\ x.\ leave \;=\; leave.\ join\ x)$$

Program-Queue Theory

$isemptyq' \Leftarrow mkemptyq$

$isemptyq \Rightarrow front'{=}x \wedge \neg isemptyq' \Leftarrow join\ x$ ⟵

$\neg isemptyq \Rightarrow front'{=}front \wedge \neg isemptyq' \Leftarrow join\ x$

$isemptyq \Rightarrow (join\ x.\ leave = mkemptyq)$

$\neg isemptyq \Rightarrow (join\ x.\ leave = leave.\ join\ x)$

Program-Queue Theory

$$isemptyq' \Leftarrow mkemptyq$$

$$isemptyq \Rightarrow front'=x \land \neg isemptyq' \Leftarrow join\ x$$

$$\neg isemptyq \Rightarrow front'=front \land \neg isemptyq' \Leftarrow join\ x \quad \leftarrow$$

$$isemptyq \Rightarrow (join\ x.\ leave = mkemptyq)$$

$$\neg isemptyq \Rightarrow (join\ x.\ leave = leave.\ join\ x)$$

Program-Queue Theory

$$isemptyq' \Leftarrow mkemptyq$$

$$isemptyq \Rightarrow front'=x \land \neg isemptyq' \Leftarrow join\ x$$

$$\neg isemptyq \Rightarrow front'=front \land \neg isemptyq' \Leftarrow join\ x$$

$$isemptyq \Rightarrow (join\ x.\ leave = mkemptyq) \quad \longleftarrow$$

$$\neg isemptyq \Rightarrow (join\ x.\ leave = leave.\ join\ x)$$

Program-Queue Theory

$$isemptyq' \quad \Leftarrow \quad mkemptyq$$

$$isemptyq \;\Rightarrow\; front'{=}x \wedge \neg isemptyq' \quad \Leftarrow \quad join\ x$$

$$\neg isemptyq \;\Rightarrow\; front'{=}front \wedge \neg isemptyq' \quad \Leftarrow \quad join\ x$$

$$isemptyq \Rightarrow (join\ x.\ \ leave \quad = \quad mkemptyq)$$

$$\neg isemptyq \Rightarrow (join\ x.\ \ leave \quad = \quad leave.\ \ join\ x) \qquad \longleftarrow$$

Program-Tree Theory

Program-Tree Theory

Variable *node* tells the value of the item where you are.

Program-Tree Theory

Variable *node* tells the value of the item where you are.

$node := 3$

Program-Tree Theory

Variable *node* tells the value of the item where you are.

$node := 3$

Variable *aim* tells what direction you are facing.

Program-Tree Theory

Variable *node* tells the value of the item where you are.

$node:= 3$

Variable *aim* tells what direction you are facing.

$aim:= up$ $aim:= left$ $aim:= right$

Program-Tree Theory

Variable *node* tells the value of the item where you are.

$node := 3$

Variable *aim* tells what direction you are facing.

$aim := up$ $aim := left$ $aim := right$

Program *go* moves you to the next node in the direction you are facing,

and turns you facing back the way you came.

Program-Tree Theory

Variable *node* tells the value of the item where you are.

$node := 3$

Variable *aim* tells what direction you are facing.

$aim := up$ $\quad$ $aim := left$ $\quad$ $aim := right$

Program *go* moves you to the next node in the direction you are facing,

and turns you facing back the way you came.

Auxiliary specification *work* says do anything, but

do not *go* from this node (your location at the start of *work*)

in this direction (the value of variable *aim* at the start of *work*).

End where you started, facing the way you were facing at the start.

Program-Tree Theory

$(aim{=}up) = (aim'{\ne}up) \quad \Leftarrow \quad go$

$node'{=}node \wedge aim'{=}aim \quad \Leftarrow \quad go.\ work.\ go$

$work \quad \Leftarrow \quad ok$

$work \quad \Leftarrow \quad node{:=}\ x$

$work \quad \Leftarrow \quad a{=}aim{\ne}b \wedge (aim{:=}\ b.\ go.\ work.\ go.\ aim{:=}\ a)$

$work \quad \Leftarrow \quad work.\ work$

Program-Tree Theory

$(aim{=}up) = (aim'{\neq}up) \quad \Leftarrow \quad go$

$node'{=}node \land aim'{=}aim \quad \Leftarrow \quad go.\ work.\ go \qquad \longleftarrow$

$work \quad \Leftarrow \quad ok$

$work \quad \Leftarrow \quad node{:=}\ x$

$work \quad \Leftarrow \quad a{=}aim{\neq}b \land (aim{:=}\ b.\ go.\ work.\ go.\ aim{:=}\ a)$

$work \quad \Leftarrow \quad work.\ work$

Program-Tree Theory

$(aim=up) = (aim' \neq up) \quad \Leftarrow \quad go$

$node'=node \wedge aim'=aim \quad \Leftarrow \quad go.\ work.\ go$

$work \quad \Leftarrow \quad ok$

$work \quad \Leftarrow \quad node:= x$

$work \quad \Leftarrow \quad a=aim \neq b \wedge (aim:= b.\ go.\ work.\ go.\ aim:= a)$

$work \quad \Leftarrow \quad work.\ work$