Deny-guarantee Reasoning

Mike Dodds

University of Cambridge

Xinyu Feng

Toyota Technological Institute at Chicago

Matthew Parkinson

University of Cambridge

Viktor Vafeiadis

Microsoft Research Cambridge

overview

- Can verify programs that write to shared resources using rely-guarantee reasoning.
- However, RG can't handle fork and join.
- *Deny-guarantee* treats interference as a resource.
- DG can reason naturally about fork and join.

interference

- Interference: changes to the shared state.

- Modelled as sets of *Actions*

$$\text{Actions} \stackrel{\mathbf{def}}{=} \text{States} \times \text{States}$$

rely-guarantee

interference tolerated
from environment

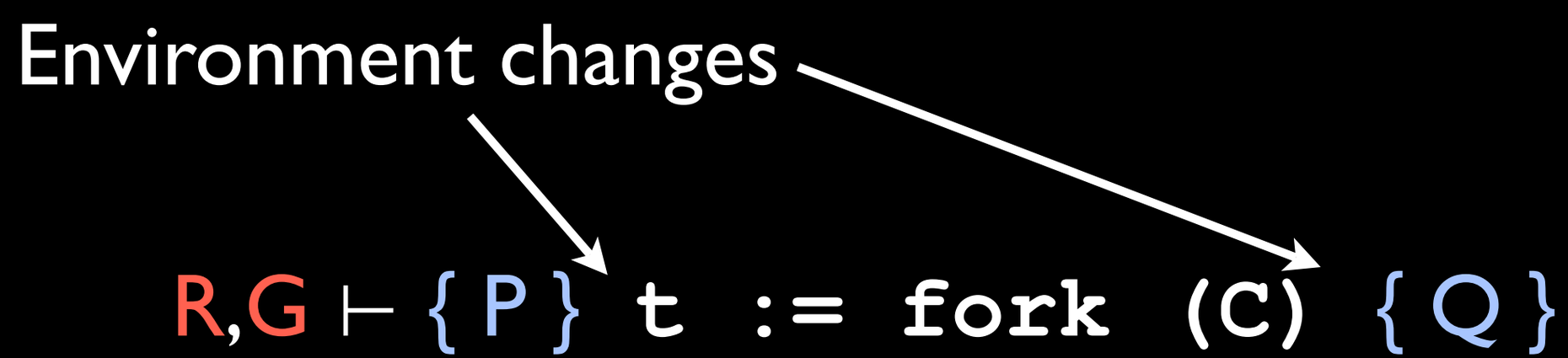
$$R, G \vdash \{ P \}\ C\ \{ Q \}$$

interference allowed
to thread

Interference same throughout C.
- R and G constant.

fork / join

Environment changes



Interference should include the
new interference caused by C

fork / join example

```
x := 0;
t1:= fork ( if(x==1) error;  x:=1;);
t2:= fork ( x:=2;  if(x==3) error;);
join t1;
x := 2;
if (x!=2) error;
join t2;
```

reasoning about fork & join

deny-guarantee

- Rely-guarantee cannot handle fork and join because it treats interference *statically*.
- Deny-guarantee treats interference as a resource.
- Can be dynamically split and joined.

deny-guarantee

- Separation logic uses * to dynamically split heap resources.

- Deny-guarantee uses * to dynamically split interference.

deny-guarantee

State and interference before

$\vdash \{P\}\ C\ \{Q\}$

State and interference after

fork rule

$$\frac{\vdash \{P\}\ \mathtt{C}\ \{Q\}}{\vdash \{P * F\}\ \mathtt{t := fork\ (C)}\ \{\ \mathrm{Thrd}(t,Q) * F\ \}}$$

join rule

$$\vdash \{ P * \mathrm{Thrd}(t, Q) \}\ \texttt{join t}\ \{ P * Q \}$$

assignment rule

$$\frac{P \Rightarrow [E \,/\, \mathtt{x}]\, Q \qquad \text{allowed}(\llbracket \mathtt{x} := E \rrbracket, P)}{\vdash \{P\}\ \mathtt{x} := E\ \{Q\}}$$

proving the example

fork / join example

```
x := 0;
t1:= fork ( if(x==1) error;  x:=1;);
t2:= fork ( x:=2;  if(x==3) error;);
join t1;
x := 2;
if (x!=2) error;
join t2;
```

interference predicates

- T_1 allows x:=1; denies environment doing x:=1
- G_2 allows x:=2;
- D_3 denies environment doing x:=3
- L allows x:=2; denies environment doing x:=N where $N \notin \{1,2\}$

$\{ T_1 * G_2 * D_3 * L * x \neq 1\}$

```
if(x==1) error;   x:=1;
```

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
```

T_1

$x \neq 1$

```
if(x==1) error;   x:=1;
```

first thread

Denies env x:=1;
Allows x:=1

$\{ T_1 * x \neq 1 \}$

```
if(x==1) error;   x:=1;
```

$\{ T_1 \}$

```
if(x==1) error;   x:=1;
```

T_1

$\{ T_1 * G_2 * D_3 * L * x \neq 1\}$

```
t1:= fork ( if(x==1) error;   x:=1;);
```

T_1

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
```

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;   if(x==3) error;);
```

second thread

G_2 * D_3

```
x:=2;   if(x==3) error;
```

G_2 * D_3

second thread

Allows
x:=2

Denies
env x:=3;

$\{ G_2 * D_3 \}$

```
x:=2;   if(x==3) error;
```

$\{ G_2 * D_3 \}$

second thread

```
x:=2;   if(x==3) error;
```

$G_2 * D_3$

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;   if(x==3) error;);
```

$G_2 * D_3$

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;   if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
```

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;   if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
join t1;
```

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;   if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
join t1;
{ L * T1 * Thrd(t2,G2*D3)}
```

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;  x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;  if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
join t1;
{ L * T1 * Thrd(t2,G2*D3)}
x := 2;
```

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;   if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
join t1;
{ L * T1 * Thrd(t2,G2*D3)}
x := 2;
{ L * T1 * Thrd(t2,G2*D3)  *  x=2}
```

Denies x:=1

Denies x:=N where N∉{1,2}

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;   x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;   if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
join t1;
{ L * T1 * Thrd(t2,G2*D3)}
x := 2;
{ L * T1 * Thrd(t2,G2*D3)  *  x=2}
```

Denies x:=1

Remains true

Denies x:=N where N∉{1,2}

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;  x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;  if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
join t1;
{ L * T1 * Thrd(t2,G2*D3)}
x := 2;
{ L * T1 * Thrd(t2,G2*D3) * x=2}
```

```
{ T1 * G2 * D3 * L * x ≠ 1}
t1:= fork ( if(x==1) error;  x:=1;);
{ G2 * D3 * L * Thrd(t1,T1)}
t2:= fork ( x:=2;  if(x==3) error;);
{ L * Thrd(t1,T1) * Thrd(t2,G2 * D3)}
join t1;
{ L * T1 * Thrd(t2,G2*D3)}
x := 2;
{ L * T1 * Thrd(t2,G2*D3) * x=2}
if (x!=2) error;
join t2;
{ G2 * D3 * L * T1 * x=2}
```

semantics of interference

permission semantics

Actions are state updates

$$\text{Actions} \stackrel{\mathbf{def}}{=} \text{States} \times \text{States}$$

Permission gives each action a level of permission

$$\text{PermDG} \stackrel{\mathbf{def}}{=} \text{Actions} \to \text{FractionDG}$$

permission semantics

$$\text{Actions} \stackrel{\mathbf{def}}{=} \text{States} \times \text{States}$$

$$\text{PermDG} \stackrel{\mathbf{def}}{=} \text{Actions} \longrightarrow \text{FractionDG}$$

Level of permission recorded by FractionDG

$$\begin{aligned}\text{FractionDG} \stackrel{\mathbf{def}}{=} & \{(\text{deny}, k) \mid k \in (0,1)\} \\ \cup\ & \{(\text{guar}, k) \mid k \in (0,1)\} \\ \cup\ & \{0, 1\}\end{aligned}$$

permission semantics

$$
\begin{aligned}
\mathsf{FractionDG} \quad \stackrel{\mathbf{def}}{=} \quad & \{(\mathsf{deny}, k) \mid k \in (0,1)\} \\
\cup \ & \{(\mathsf{guar}, k) \mid k \in (0,1)\} \\
\cup \ & \{0, 1\}
\end{aligned}
$$

permission semantics

$$
\begin{aligned}
\mathsf{FractionDG} \quad \stackrel{\mathbf{def}}{=} \quad & \{(\mathsf{deny}, k) \mid k \in (0,1)\} \\
\cup \ & \{(\mathsf{guar}, k) \mid k \in (0,1)\} \\
\cup \ & \{0, 1\}
\end{aligned}
$$

permission semantics

$$\begin{aligned}\mathsf{FractionDG} \stackrel{\mathbf{def}}{=} & \quad \{(\mathsf{deny}, k) \mid k \in (0,1)\} \\ \cup & \quad \{(\mathsf{guar}, k) \mid k \in (0,1)\} \\ \cup & \quad \{0, 1\}\end{aligned}$$

Join elements of FractionDG by addition.

$$0 \oplus p = p \qquad\qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

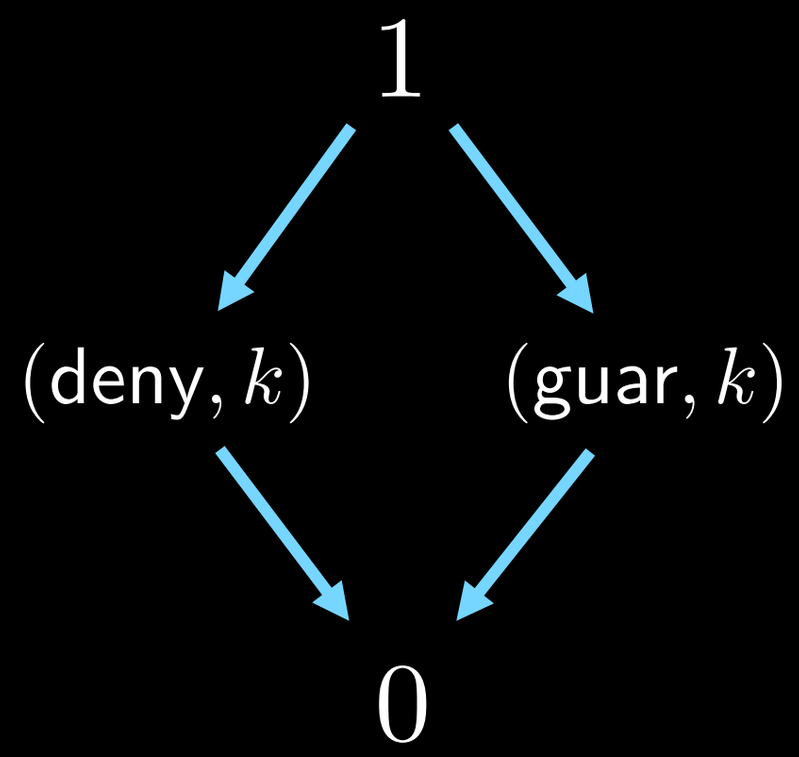
$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

permission semantics

$$0 \oplus p = p \qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

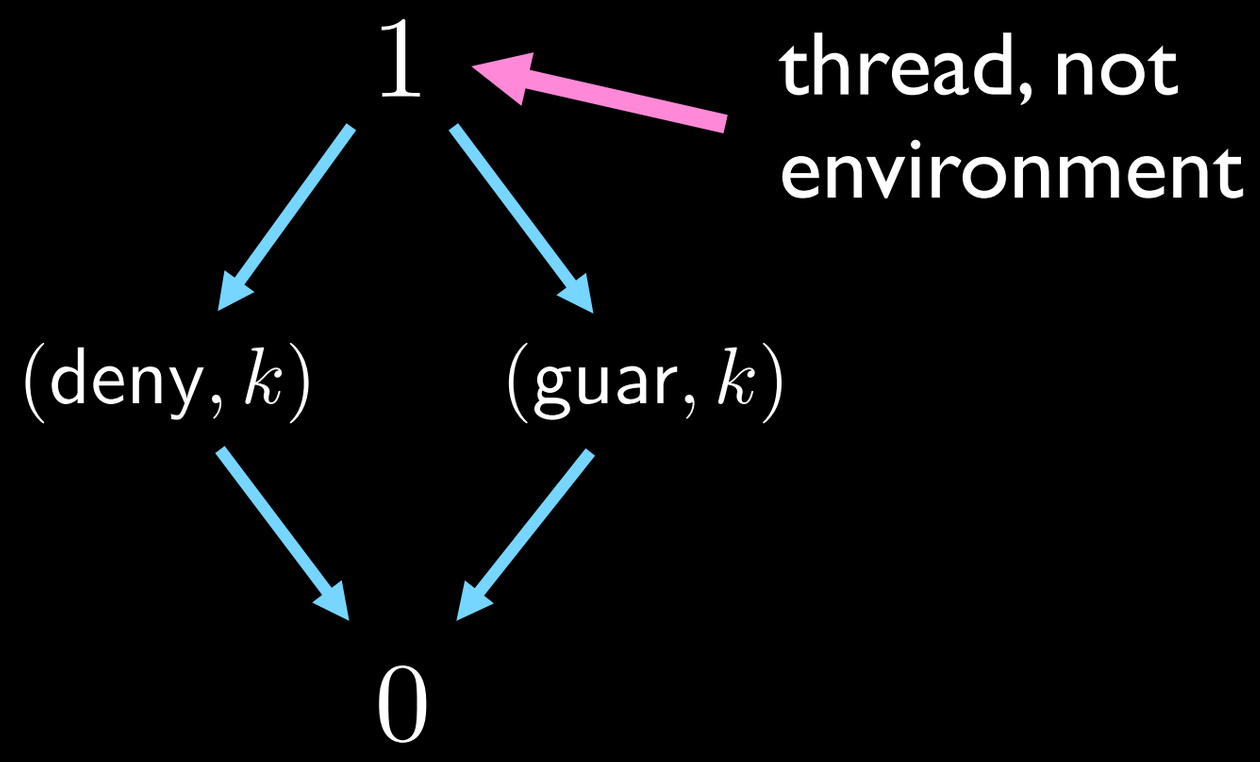


permission semantics

$$0 \oplus p = p \qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

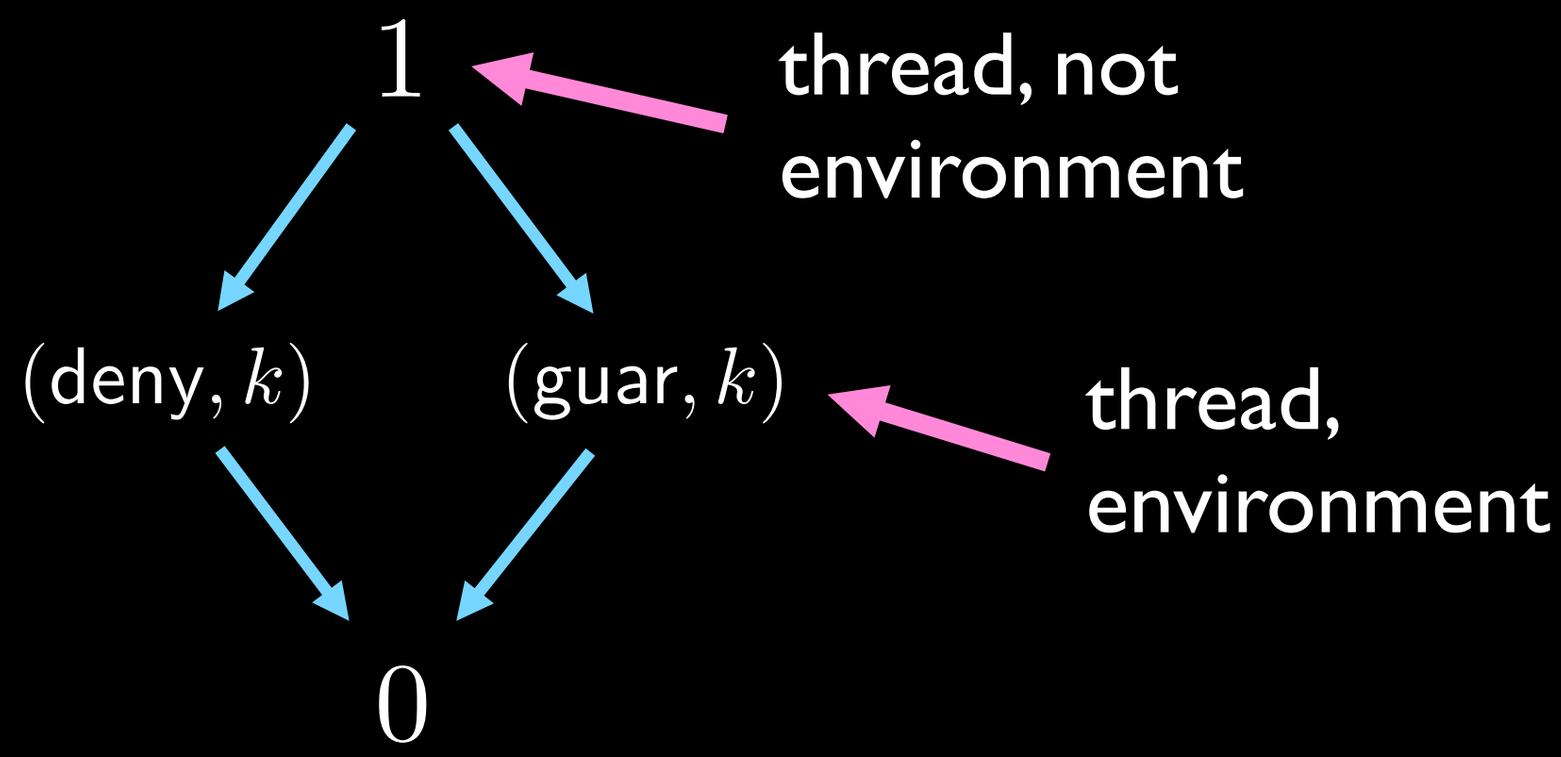


permission semantics

$$0 \oplus p = p \qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

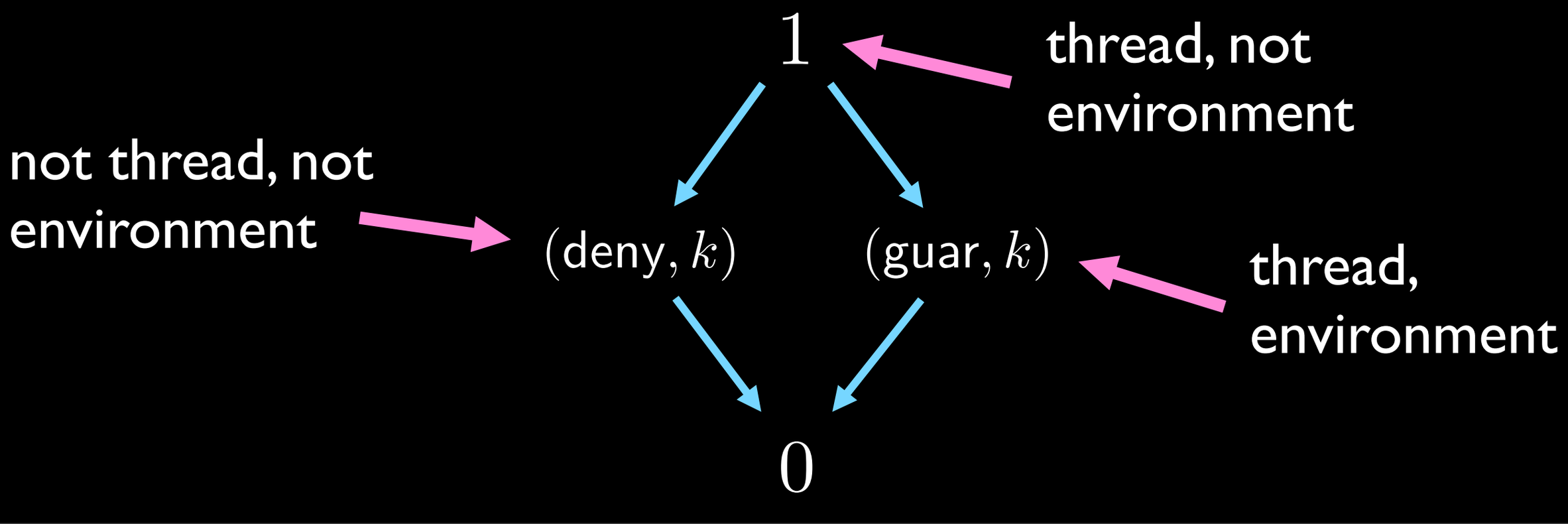


permission semantics

$$0 \oplus p = p \qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

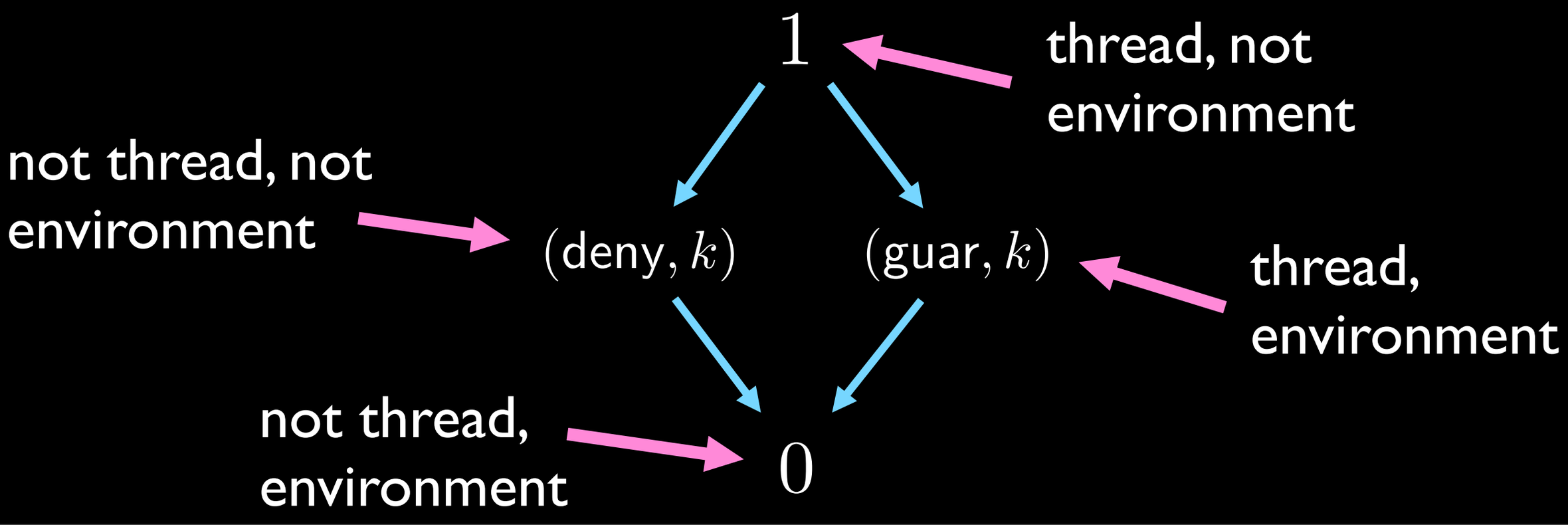


permission semantics

$$0 \oplus p = p \qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$



permission semantics

$$0 \oplus p = p \qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

permission semantics

$$0 \oplus p = p \qquad\qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

permission semantics

$$0 \oplus p = p \qquad\qquad 1 \oplus 0 = 1$$

$$(\mathsf{guar}, k) \oplus (\mathsf{guar}, k') = \begin{cases} (\mathsf{guar}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

$$(\mathsf{deny}, k) \oplus (\mathsf{deny}, k') = \begin{cases} (\mathsf{deny}, k + k') & \text{if } k + k' < 1 \\ 1 & \text{if } k + k' = 1 \end{cases}$$

Lift the join operator to PermDG.

$$P \oplus Q \quad = \quad \lambda(s, s').\ P(s, s') \oplus Q(s, s')$$

This gives a semantics for * on permissions.

T1 allows x:=1; denies environment doing x:=1

TI allows x:=I ; denies environment doing x:=I

Define a class of actions for x:= i

$$[\![\ \text{x:= } i\ ]\!] \quad \stackrel{\text{def}}{=} \quad \{(s, s') \in \text{Actions} \mid s' = s[x \mapsto i]\}$$

T1 allows x:=1; denies environment doing x:=1

$[\![\ \mathsf{x{:=}}\ i\ ]\!] \quad \stackrel{\mathbf{def}}{=} \quad \{(s, s') \in \mathsf{Actions} \mid s' = s[x \mapsto i]\}$

TI allows x:=1; denies environment doing x:=1

$$\mathsf{TI}(s,s') \stackrel{\mathbf{def}}{=} \begin{cases} 1 & (s,s') \in [\![\text{x:=1}]\!] \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\text{x:= } i]\!] \stackrel{\mathbf{def}}{=} \{(s,s') \in \text{Actions} \mid s' = s[x \mapsto i]\}$$

TI allows x:=I ; denies environment doing x:=I

$$\mathsf{TI}(s,s') \stackrel{\mathbf{def}}{=} \begin{cases} 1 & (s,s') \in [\![\text{x:=I}]\!] \\ 0 & \text{otherwise} \end{cases}$$

L allows x:=2; denies env. doing x:=N for N∉{1,2}

$$[\![\text{x:=} i]\!] \stackrel{\mathbf{def}}{=} \{(s,s') \in \text{Actions} \mid s' = s[x \mapsto i]\}$$

TI allows x:=1; denies environment doing x:=1

$$\mathsf{TI}(s,s') \stackrel{\text{def}}{=} \begin{cases} 1 & (s,s') \in [\![\text{x:=1}]\!] \\ 0 & \text{otherwise} \end{cases}$$

L allows x:=2; denies env. doing x:=N for N∉{1,2}

$$\mathsf{L}(s,s') \stackrel{\text{def}}{=} \begin{cases} (\text{guar}, 1/2) & (s,s') \in [\![\text{x:=2}]\!] \\ & \end{cases}$$

$$[\![\text{x:= } i]\!] \stackrel{\text{def}}{=} \{(s,s') \in \text{Actions} \mid s' = s[x \mapsto i]\}$$

T1 allows x:=1; denies environment doing x:=1

$$\mathsf{T1(s,s')} \stackrel{\text{def}}{=} \begin{cases} 1 & (s, s') \in [\![\text{ x:=1 }]\!] \\ 0 & \text{otherwise} \end{cases}$$

L allows x:=2; denies env. doing x:=N for N∉{1,2}

$$\mathsf{L(s,s')} \stackrel{\text{def}}{=} \begin{cases} (\text{guar}, 1/2) & (s, s') \in [\![\text{ x:=2 }]\!] \\ (\text{deny}, 1/2) & (s, s') \in [\![\text{ x:= Z} \setminus \{1,2\}]\!] \end{cases}$$

$$[\![\text{ x:= } i \text{ }]\!] \stackrel{\text{def}}{=} \{(s, s') \in \text{Actions} \mid s' = s[x \mapsto i]\}$$

TI allows x:=1; denies environment doing x:=1

$$\mathsf{TI}(s,s') \stackrel{\text{def}}{=} \begin{cases} 1 & (s,s') \in [\![\text{x:=1}]\!] \\ 0 & \text{otherwise} \end{cases}$$

L allows x:=2; denies env. doing x:=N for N∉{1,2}

$$\mathsf{L}(s,s') \stackrel{\text{def}}{=} \begin{cases} (\text{guar}, 1/2) & (s,s') \in [\![\text{x:=2}]\!] \\ (\text{deny}, 1/2) & (s,s') \in [\![\text{x:= Z} \setminus \{1,2\}]\!] \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\text{x:= } i]\!] \stackrel{\text{def}}{=} \{(s,s') \in \text{Actions} \mid s' = s[x \mapsto i]\}$$

assertions and stability

assertions

Assertion P modelled as follows:

$$s, pr, \gamma \models P$$

assertions

Assertio odelled as follows:

Program state

$$s, pr, \gamma \models P$$

assertions

Assertio odelled as f

Program state

Thread queue

$$s, pr, \gamma \models P$$

assertions

Assertio odelled as f

Program state

Thread queue

$$s, pr, \gamma \models P$$

Permission

assertions

Assertion P modelled as follows:

$$s, pr, \gamma \models P$$

stability

$$pr.R \stackrel{\mathbf{def}}{=} \{(s, s') \mid pr(s, s') \in \{(\mathsf{guar}, k), 0\}\}$$

$$pr.G \stackrel{\mathbf{def}}{=} \{(s, s') \mid pr(s, s') \in \{(\mathsf{guar}, k), 1\}\}$$

stability

$$pr.R \stackrel{\mathbf{def}}{=} \{(s, s') \mid pr(s, s') \in \{(\mathsf{guar}, k), 0\}\}$$

$$pr.G \stackrel{\mathbf{def}}{=} \{(s, s') \mid pr(s, s') \in \{(\mathsf{guar}, k), 1\}\}$$

$\mathsf{stable}(P)$ *iff* $s, pr, \gamma \models P \wedge (s, s') \in pr.R \implies s', pr, \gamma \models P$

conclusions

formal results

- Encoding: all rely-guarantee proofs can be encoded directly onto deny-guarantee.

- Soundness: mechanical verification of the soundness of deny-guarantee.

summary

- Deny-guarantee treats interference as a resource.
- Interference is modelled by permissions on actions.
- We reason about permissions using separation logic.
- We handle fork and join naturally by splitting permissions.

future work

- Higher-order deny-guarantee: permissions that can rewrite permissions

- Local deny-guarantee: permissions with footprint and scope