Noninfluence = Noninterference + Nonleakage

David von Oheimb

June 8, 2004

Contents

1 Automata

```
typedecl "state"
typedecl "action"
typedecl "output"
typedecl "domain"
```

System described as Moore (rather than Mealy) automaton

```
consts   step :: "action ⇒  state ⇒ state"
         Step :: "action ⇒ (state × state) set" — non-deterministic step
       "output" :: "domain ⇒ state ⇒ output set" — all observations of a domain
         run  :: "action list ⇒  state ⇒ state"
         Run  :: "action list ⇒ (state × state) set" — non-deterministic run

primrec "run []     = (λs. s)"
        "run (a#as) = run as ∘ step a"
primrec "Run []     = Id"
        "Run (a#as) = Run as O Step a"

consts s0 :: "state"
```

2 Generic notions

2.1 policies

```
consts dom          :: "action ⇒ domain"              — security domain
       policy       :: "domain ⇒ domain ⇒ bool"  ("(_ ⇝ _)")
syntax policy_neg :: "domain ⇒ domain ⇒ bool"  ("(_ ⇝̸ _)")
translations "u ⇝̸ v" ⇌ "¬(u ⇝ v)"

axioms policy_refl: "u ⇝ u"

locale policy_trans =
  assumes policy_trans: "⟦u ⇝ v; v ⇝ w⟧ ⟹ u ⇝ w"
```

2.2 allowed source domains

```
types sourcef = "action list ⇒ domain ⇒ domain set"
```

2.2.1 trivial source functions

```
constdefs
  singleton :: "sourcef"
 "singleton as u ≡ {u}"

  tsources :: "sourcef"
 "tsources as u ≡ {w. w ⇝ u}"
```

2.2.2 chains of domains

```
consts gen_chain :: "(domain ⇒ action ⇒ bool) ⇒ sourcef"
primrec
  Nil:  "gen_chain P []     u = {u}"
  Cons: "gen_chain P (a#as) u = gen_chain P as u ∪
                   {w. P w a ∧ (∃v. w ⇝ v ∧ v ∈ gen_chain P as u)}"

lemma gen_chain_refl: "u ∈ gen_chain P as u"
lemma gen_chain_trans:
  "⟦w ⇝ v; v ∈ gen_chain P as u; P w a⟧ ⟹ w ∈ gen_chain P (a#as) u"lemma gen_chain_subset_Cons
"gen_chain P as u ⊆ gen_chain P (a#as) u"lemma (in policy_trans) gen_chain_implies_policy:
— Rushby's Lemma 6
  "w ∈ gen_chain P as u ⟹ w ⇝ u"lemma (in policy_trans) in_gen_chain_Cons_eq:
  "P w a ⟹ w ∈ gen_chain P (a#as) u ⟷ w ⇝ u"
constdefs
  chain :: "sourcef"
 "chain ≡ gen_chain (λw a. True)"

lemma (in policy_trans) chain_subset_tsources: "chain as u ⊆ tsources as u"
constdefs
  sources :: "sourcef"
 "sources ≡ gen_chain (λw a. w = dom a)"

lemma sources_subset_chain: "sources as u ⊆ chain as u"—
```

2.3 unwinding relations

```
consts uwr :: "state ⇒ domain ⇒ state ⇒ bool"  ("(_ ∼_∼ _)")

axioms
  uwr_s0: "s0 ∼u∼ s0"
constdefs gen_uwr :: "state ⇒ domain set ⇒ state ⇒ bool"
  "s ≈us≈ t ≡ ∀u∈us. s ∼u∼ t"
constdefs nest_uwr :: "state ⇒ domain ⇒ state ⇒ bool"
    "s ≃u≃ t ≡ s ≈{v. v ⇝ u}≈ t"

lemma tsources_uwr_is_nest: "s ≈tsources as u≈ t ⟷ s ≃u≃ t"
lemma (in policy_trans) nesting: "⟦s ≃v≃ t; u ⇝ v⟧ ⟹ s ≃u≃ t"

constdefs
  output_consistent :: "bool"
 "output_consistent ≡ ∀u s t. s ∼u∼ t ⟶ output u s = output u t"
```

2.4 the deterministic case

```
constdefs
  obs_equiv :: "state ⇒ action list ⇒ domain ⇒ action list ⇒ state ⇒ bool"
("(_ ≐_,_,_≐ _)")
  "s ≐as,u,bs≐ t ≡ output u (run as s) = output u (run bs t)"
constdefs
  weakly_step_consistent :: "bool" — sufficient also for transitive policies, new premise dom a ⇝ u
 "weakly_step_consistent ≡ ∀a u s t. dom a ⇝ u ⟶ s ∼dom a∼ t ⟶
                                    s ∼u∼ t ⟶ step a s ∼u∼ step a t"
constdefs
  step_respect :: "bool" — a consequence of local_respect
 "step_respect ≡ ∀a u s t. dom a ⇝̸ u ⟶ s ∼u∼ t ⟶ step a s ∼u∼ step a t"
constdefs
  gen_weak_step_consistent_respect :: "(domain ⇒ action ⇒ bool) ⇒ bool"
 "gen_weak_step_consistent_respect P ≡ ∀a u s t. (∀w. P w a ⟶ w⇝u ⟶ s ∼w∼ t) ⟶
                                    s ∼u∼ t ⟶ step a s ∼u∼ step a t"

lemma gen_weak_step_consistent_respect_action:
"⟦weakly_step_consistent; step_respect⟧ ⟹
  gen_weak_step_consistent_respect (λw a. w = dom a)"
lemma gen_chain_unwinding_step:
"⟦s ≈gen_chain P (a#as) u≈ t; gen_weak_step_consistent_respect P⟧ ⟹
  step a s ≈gen_chain P as u≈ step a t"
lemma sources_unwinding_step: — Rushby's Lemma 3
"⟦s ≈sources (a#as) u≈ t; weakly_step_consistent; step_respect⟧ ⟹
  step a s ≈sources as u≈ step a t"—
```

2.5 the nondeterministic case

— TODO: reachability

```
constdefs
  obs_PO :: "state ⇒ action list ⇒ domain ⇒ action list ⇒ state ⇒ bool"
("(_ ≏_,_,_≏ _)")
  "s ≏as,u,bs≏ t ≡ ∀s'. (s, s') ∈ Run as ⟶
                      (∃t'. (t, t') ∈ Run bs ∧ output u s' = output u t')"
```

2.5.1 simple version

```
constdefs
  Step_consistent :: "bool"
 "Step_consistent ≡ ∀a u s s' t. dom a ⇝ u ⟶
  (s, s') ∈ Step a ⟶ s ∼u∼ t ⟶ (∃t'. (t, t') ∈ Step a ∧ s' ∼u∼ t')"

  Step_respect :: "bool" — a consequence of Local_respect
 "Step_respect ≡ ∀a u s s' t. dom a ⇝̸ u ⟶
  (s, s') ∈ Step a ⟶ s ∼u∼ t ⟶ (∃t'. (t, t') ∈ Step a ∧ s' ∼u∼ t')"
lemma simple_unwinding_Step:
      "⟦(s, s') ∈ Step a; s ∼u∼ t; Step_consistent; Step_respect ⟧ ⟹
  ∃t'. (t, t') ∈ Step a ∧ s' ∼u∼ t'"
```

2.5.2 uniform version

```
constdefs
  uni_Step_consistent :: "bool" — uniform
 "uni_Step_consistent ≡ ∀a us s s' t. (∃u∈us. dom a ⇝ u) ⟶ s ∼dom a∼ t ⟶
      (s, s') ∈ Step a ⟶ s ≈us≈ t ⟶
(∃t'. (t, t') ∈ Step a ∧ s' ≈us≈ t')"

  uni_Step_respect :: "bool"
 "uni_Step_respect ≡ ∀a us s t s'.  ¬(∃u∈us. dom a ⇝ u) ⟶ (∃u. u∈us) ⟶
       (s, s') ∈ Step a ⟶ s ≈us≈ t ⟶
  (∃t'. (t, t') ∈ Step a ∧ s' ≈us≈ t')"
constdefs
  gen_uni_Step_consistent_respect :: "(domain ⇒ action ⇒ bool) ⇒ bool"
 "gen_uni_Step_consistent_respect P ≡ ∀a s us t s'.
   (∀w. P w a ⟶ (∃u∈us. w ⇝ u) ⟶ s ∼w∼ t) ⟶ (∃u. u∈us) ⟶
        (s, s') ∈ Step a ⟶ s ≈us≈ t ⟶
   (∃t'. (t, t') ∈ Step a ∧ s' ≈us≈ t')"

lemma gen_chain_unwinding_Step:
      "⟦(s, s') ∈ Step a; s ≈gen_chain P (a#as) u≈ t;
        gen_uni_Step_consistent_respect P⟧ ⟹
  ∃t'. (t, t') ∈ Step a ∧ s' ≈gen_chain P as u≈ t'"
lemma sources_unwinding_Step:
      "⟦(s, s') ∈ Step a; s ≈sources (a#as) u≈ t;
        uni_Step_consistent; uni_Step_respect⟧ ⟹
  ∃t'. (t, t') ∈ Step a ∧ s' ≈sources as u≈ t'"
locale Step_functional =
  assumes Step_functional: "⟦(x, y) ∈ Step a; (x, z) ∈ Step a⟧ ⟹ y = z"

lemma (in Step_functional) uni_Step_consistent:
  "Step_respect ⟹ Step_consistent ⟹ uni_Step_consistent"
lemma (in Step_functional) uni_Step_respect: "uni_Step_respect = Step_respect"
```

3 Noninterference

3.1 purging

```
consts gen_purge :: "sourcef ⇒ domain ⇒ action list ⇒ action list"
primrec
  Nil : "gen_purge sf u []     = []"
  Cons: "gen_purge sf u (a#as) = (if dom a ∈ sf (a#as) u then [a] else [])
                                 @ gen_purge sf u as"

constdefs — also for transitive policies
  ipurge :: "domain ⇒ action list ⇒ action list"
 "ipurge ≡ gen_purge sources"
lemma sources_ipurge: "sources (ipurge u as) u = sources as u"lemma ipurge_sources_cong:

  "ipurge u as = ipurge u bs ⟹ sources as u = sources bs u"lemma ipurge_idempotent: "ipurge
u (ipurge u as) = ipurge u as"
constdefs — specical case of ipurge for transitive policies
  tpurge   :: "domain ⇒ action list ⇒ action list"
 "tpurge ≡ gen_purge tsources"
lemma tpurge_idempotent: "tpurge u (tpurge u as) = tpurge u as"lemma "tpurge u = filter (λa.
(dom a ↝ u))"
lemma (in policy_trans) tpurge_conincides: "tpurge = ipurge"
```

3.2 the deterministic case

3.2.1 general version

```
constdefs
  noninterference :: "bool"
 "noninterference ≡ ∀as u. s0 ≏as,u,ipurge u as≏ s0"

constdefs — common structure of noninterference and noninfluence
  gen_noninterference :: "sourcef ⇒ bool"
 "gen_noninterference sf ≡
 ∀u as s t. s ≈sf as u≈ t ⟶ run as s ∼u∼ run (ipurge u as) t"

lemma output_consistent_and_gen_noninterference_implies_noninterference:
  "output_consistent ⟹ gen_noninterference sf ⟹ noninterference"
constdefs
  local_respect_left :: "bool"
 "local_respect_left ≡ ∀a u s t. dom a ↝̸ u ⟶ s ∼u∼ t ⟶ step a s ∼u∼ t"

  local_respect_right :: "bool"
 "local_respect_right ≡ ∀a u s t. dom a ↝̸ u ⟶ s ∼u∼ t ⟶ s ∼u∼ step a t"

  local_respect :: "bool"
 "local_respect ≡ local_respect_left ∧ local_respect_right"
lemma (in uwr_refl) local_respect_classical:
  "local_respect ⟹ ∀a u s. dom a ↝̸ u ⟶ s ∼u∼ step a s"lemma (in uwr_trans) classical_local_

  "∀s u t. s ∼u∼ t ⟶ t ∼u∼ s ⟹
   ∀a u s. dom a ↝̸ u ⟶ s ∼u∼ step a s ⟹ local_respect"
lemma local_respect_implies_step_respect: "local_respect ⟹ step_respect"
lemma gen_noninterference_sources: — Rushby's Lemma 5
 "weakly_step_consistent ⟹ local_respect ⟹ gen_noninterference sources"
theorem noninterference: — Rushby's Theorem 7
"⟦weakly_step_consistent; local_respect; output_consistent⟧ ⟹ noninterference"
```

3.2.2 simple version

```
constdefs
 step_consistent :: "bool" — new premise dom a ↝ u
"step_consistent ≡ ∀a u s t. dom a ↝ u ⟶ s ∼u∼ t ⟶ step a s ∼u∼ step a t"
```

```
theorem simple_noninterference: — Rushby's Theorem 1
 "step_consistent ⟹ local_respect ⟹ gen_noninterference singleton"
```

3.2.3 strong version

```
constdefs
  strong_noninterference :: "bool"
 "strong_noninterference ≡ ∀as u bs. ipurge u as = ipurge u bs ⟶ s0 ≏as,u,bs≏ s0"

lemma strong_noninterference_implies_noninterference:
 "strong_noninterference ⟹ noninterference"
lemma ipurge_nilD [rule_format]: "local_respect_right ⟹
  [] = ipurge u bs ⟶ (∀t. s ∼u∼ t ⟶ s ∼u∼ run bs t)"
lemma ipurge_consD [rule_format]:
 "local_respect_right ⟹ a # as = ipurge u bs  ⟶
  (∃bsa bsc. bs = bsa @ a # bsc ∧ as = ipurge u bsc ∧
(∀t. s ≈sources (a#as) u≈ t ⟶ s ≈sources (a#as) u≈ run bsa t))"
theorem strong_noninterference:
 "⟦weakly_step_consistent; local_respect; output_consistent⟧ ⟹ strong_noninterference"
```

3.2.4 access control interpretation

```
typedecl "name"
typedecl "value"
consts contents :: "state ⇒ name ⇒ value"

consts observe :: "domain ⇒ name set"
       alter   :: "domain ⇒ name set"

defs uwr_def: "s ∼u∼ t ≡ ∀n∈observe u. contents s n = contents t n"

locale canonical_output = — special case: all observable values are output
  fixes value2output :: "value ⇒ output" — type coercion
  assumes output_def:
     "output u s ≡ {value2output (contents s n) |n. n ∈ observe u}"
lemma (in canonical_output) canonical_output_consistent: "output_consistent"
constdefs — Reference Monitor Assumptions
  RMA1 :: "bool"
 "RMA1 ≡ output_consistent"

  RMA2 :: "bool" — new premises dom a ⇝ u, s ∼u∼ t, and n ∈ observe u
 "RMA2 ≡ ∀a u s t n. s ∼dom a∼ t ⟶ dom a ⇝ u ⟶ s ∼u∼ t ⟶ n ∈ observe u ⟶
                     (contents (step a s) n ≠ contents s n ∨
                      contents (step a t) n ≠ contents t n) ⟶
                      contents (step a s) n = contents (step a t) n"

  RMA3 :: "bool"
 "RMA3 ≡ ∀a s n. contents (step a s) n ≠ contents s n ⟶ n ∈ alter (dom a)"

  AC_policy_consistent :: "bool"
 "AC_policy_consistent ≡ ∀u v. alter u ∩ observe v ≠ {} ⟶ u ⇝ v"
lemma RMA2_implies_weakly_step_consistent: "RMA2 ⟹ weakly_step_consistent"
lemma RMA3_AC_policy_consistent_implies_local_respect:
  "RMA3 ⟹ AC_policy_consistent ⟹ local_respect"
theorem access_control_secure:
  "⟦RMA1; RMA2; RMA3; AC_policy_consistent⟧ ⟹ noninterference"
```

3.3 the nondeterministic case

```
constdefs
  Noninterference :: "bool"
 "Noninterference ≡ ∀as u bs. ipurge u as = ipurge u bs ⟶ s0 ≏as,u,bs≏ s0"

  gen_Noninterference :: "sourcef ⇒ bool"
 "gen_Noninterference sf ≡ ∀as bs s s' u t. ipurge u as = ipurge u bs ⟶
```

```
        (s, s') ∈ Run as ⟶ s ≈sf as u≈ t ⟶
   (∃t'. (t, t') ∈ Run bs ∧ s' ∼u∼ t')"
lemma
  output_consistent_and_gen_Noninterference_implies_Noninterference:
  "output_consistent ⟹ gen_Noninterference sf ⟹ Noninterference"
```

3.3.1 simple version

```
constdefs
  Local_respect_left :: "bool"
 "Local_respect_left ≡ ∀a u s t s'. dom a ↝̸ u ⟶
  s ∼u∼ t ⟶ (s, s') ∈ Step a ⟶ s' ∼u∼ t"

  Local_respect_right :: "bool"
 "Local_respect_right ≡ ∀a u s t. dom a ↝̸ u ⟶
  s ∼u∼ t ⟶ (∃t'. (t, t') ∈ Step a ∧ s ∼u∼ t')"
lemma Local_respect_implies_Step_respect:
 "⟦Local_respect_left; Local_respect_right⟧ ⟹ Step_respect"
lemma (in uwr_refl) Local_respect_left_Mantel:
  "Local_respect_left ⟹
   ∀a u s t s'. dom a ↝̸ u ⟶ (s, s') ∈ Step a ⟶ s' ∼u∼ s"
lemma (in uwr_refl) Local_respect_right_Mantel:
  "Local_respect_right ⟹
   ∀a u s t. dom a ↝̸ u ⟶ (∃t'. (t, t') ∈ Step a ∧ t ∼u∼ t')"
lemma (in uwr_trans) Mantel_Local_respect_left:
  "∀a u s t s'. dom a ↝̸ u ⟶ (s, s') ∈ Step a ⟶ s' ∼u∼ s ⟹
   Local_respect_left"
lemma (in uwr_trans) Mantel_Local_respect_right:
  "∀a u s t. dom a ↝̸ u ⟶ (∃t'. (t, t') ∈ Step a ∧ t ∼u∼ t') ⟹
   Local_respect_right"
lemma ipurge_NilD [rule_format]: "Local_respect_right ⟹
  [] = ipurge u bs ⟶ (∀t. s ∼u∼ t ⟶ (∃t'. (t, t') ∈ Run bs ∧ s ∼u∼ t'))"
lemma ipurge_ConsD [rule_format]: "Local_respect_right ⟹
  a # as = ipurge u bs ⟶ (∃bsa bsc. bs = bsa @ a # bsc ∧ as = ipurge u bsc ∧
  (∀t. s ∼u∼ t ⟶ (∃ta. (t, ta) ∈ Run bsa ∧ s ∼u∼ ta)))"
theorem simple_Noninterference:
 "Step_consistent ⟹ Local_respect_left ⟹ Local_respect_right ⟹
  output_consistent ⟹ Noninterference"
```

3.3.2 uniform version

```
constdefs
  uni_Local_respect_right :: "bool"
 "uni_Local_respect_right ≡ ∀a us s t. ¬(∃u∈us. dom a ↝ u) ⟶ (∃u. u∈us) ⟶
  s ≈us≈ t ⟶ (∃t'. (t, t') ∈ Step a ∧ s ≈us≈ t')"

  uni_Local_respect :: "bool"
 "uni_Local_respect ≡ Local_respect_left ∧ uni_Local_respect_right"

lemma uni_Local_respect_leftD: "⟦Local_respect_left;
  (s, s') ∈ Step a; ¬(∃u∈us. dom a ↝ u); s ≈us≈ t⟧ ⟹ s' ≈us≈ t"

lemma uni_Local_respect_right_implies_Local_respect_right:
  "uni_Local_respect_right ⟹ Local_respect_right"
lemma uni_Local_respect_implies_uni_Step_respect:
  "uni_Local_respect ⟹ uni_Step_respect"
lemma uni_ipurge_ConsD [rule_format]: "uni_Local_respect_right ⟹
  a # as = ipurge u bs ⟶ (∃bsa bsc. bs = bsa @ a # bsc ∧ as = ipurge u bsc ∧
  (∀t. s ≈sources (a#as) u≈ t ⟶ (∃ta. (t, ta) ∈ Run bsa ∧ s ≈sources (a#as) u≈ ta)))"
lemma gen_Noninterference_sources:
 "uni_Step_consistent ⟹ uni_Local_respect ⟹ gen_Noninterference sources"
theorem Noninterference: "uni_Step_consistent ⟹
 uni_Local_respect ⟹ output_consistent ⟹ Noninterference"
```

4 Nonleakage

4.1 the deterministic case

```
constdefs — generic nonleakage
  gen_nonleakage :: "sourcef ⇒ bool"
 "gen_nonleakage sf ≡ ∀as s u t. s ≈sf as u≈ t ⟶ run as s ∼u∼ run as t"
constdefs
  nonleakage :: "bool"
 "nonleakage ≡ ∀as s u t. s ≈sources as u≈ t ⟶ s ≏as,u,as≏ t"

theorem nonleakage:
   "⟦weakly_step_consistent; step_respect; output_consistent⟧ ⟹ nonleakage"
```

4.1.1 weak nonleakage

```
constdefs
  weak_nonleakage :: "bool"
 "weak_nonleakage ≡ ∀as s u t. s ≈chain as u≈ t ⟶ s ≏as,u,as≏ t"

lemma nonleakage_implies_weak_nonleakage: "nonleakage ⟹ weak_nonleakage"
constdefs
  weak_step_consistent_respect :: "bool"
 "weak_step_consistent_respect ≡ ∀s u t. s ≃u≃ t ⟶ (∀a. step a s ∼u∼ step a t)"

lemma weak_step_consistent_respect_is_gen_weak_step_consistent_respect_True:
 "weak_step_consistent_respect = gen_weak_step_consistent_respect (λw a. True)"

theorem weak_nonleakage:
  "⟦weak_step_consistent_respect; output_consistent⟧ ⟹ weak_nonleakage"
```

4.1.2 transitive weak nonleakage

```
constdefs
  trans_weak_nonleakage :: "bool"
 "trans_weak_nonleakage ≡ ∀s u t. s ≃u≃ t ⟶ (∀as. s ≏as,u,as≏ t)"

lemma (in policy_trans) weak_nonleakage_implies_trans_weak_nonleakage:
  "weak_nonleakage ⟹ trans_weak_nonleakage"
theorem (in policy_trans) trans_weak_nonleakage:
  "⟦weak_step_consistent_respect; output_consistent⟧ ⟹ trans_weak_nonleakage"—
```

4.2 the nondeterministic case

```
constdefs
  gen_Nonleakage :: "sourcef ⇒ bool"
 "gen_Nonleakage sf ≡ ∀u as s s' t.
       (s, s') ∈ Run as ⟶ s ≈sf as u≈ t ⟶
  (∃t'. (t, t') ∈ Run as ∧ s' ∼u∼ t')"

lemma gen_Nonleakage:
 "gen_uni_Step_consistent_respect P ⟹ gen_Nonleakage (gen_chain P)"
constdefs
  Nonleakage :: "bool"
 "Nonleakage ≡ ∀as s u t. s ≈sources as u≈ t ⟶ s ⥲as,u,as⥲ t"

theorem Nonleakage:
 "⟦uni_Step_consistent; uni_Step_respect; output_consistent⟧ ⟹ Nonleakage"
```

4.2.1 weak Nonleakage

```
constdefs
  weak_Nonleakage :: "bool"
 "weak_Nonleakage ≡ ∀as s u t. s ≈chain as u≈ t ⟶ s ⥲as,u,as⥲ t"

lemma Nonleakage_implies_weak_Nonleakage: "Nonleakage ⟹ weak_Nonleakage"
constdefs
  weak_uni_Step_consistent_respect :: "bool"
 "weak_uni_Step_consistent_respect ≡ ∀a s s' us t. (∃u. u∈us) ⟶
        (s, s') ∈ Step a ⟶ (∀u∈us. s ≃u≃ t) ⟶
   (∃t'. (t, t') ∈ Step a ∧ s' ≈us≈ t')"

lemma weak_uni_Step_consistent_respect_is_gen_uni_Step_consistent_respect_True:
  "weak_uni_Step_consistent_respect = gen_uni_Step_consistent_respect (λw a. True)"

theorem weak_Nonleakage:
  "⟦weak_uni_Step_consistent_respect; output_consistent⟧ ⟹ weak_Nonleakage"
```

4.2.2 transitive weak Nonleakage

```
constdefs
  trans_weak_Nonleakage :: "bool"
 "trans_weak_Nonleakage ≡ ∀s u t. s ≃u≃ t ⟶ (∀as. s ⥲as,u,as⥲ t)"

lemma (in policy_trans) weak_Nonleakage_implies_trans_weak_Nonleakage:
  "weak_Nonleakage ⟹ trans_weak_Nonleakage"
theorem (in policy_trans) trans_weak_Nonleakage:
"⟦weak_uni_Step_consistent_respect; output_consistent⟧ ⟹ trans_weak_Nonleakage"
```

5 Noninfluence

5.1 the deterministic case

```
constdefs
  noninfluence :: "bool"
 "noninfluence ≡ ∀as u s t. s ≈sources as u≈ t ⟶ s ≏as,u,ipurge u as≏ t"

lemma noninfluence_implies_noninterference: "noninfluence ⟹ noninterference"
theorem noninfluence:
  "⟦weakly_step_consistent; local_respect; output_consistent⟧ ⟹ noninfluence"
```

5.2 the nondeterministic case

```
constdefs
  Noninfluence :: "bool"
 "Noninfluence ≡
  ∀as bs u s t. s ≈sources as u≈ t ⟶ ipurge u as = ipurge u bs ⟶ s ≏as,u,bs≏ t"

lemma Noninfluence_implies_Noninterference: "Noninfluence ⟹ Noninterference"
theorem Noninfluence:
  "⟦uni_Step_consistent; uni_Local_respect; output_consistent⟧ ⟹ Noninfluence"
```