CsPL, system do weryfikacji bezpieczeństwa programów

XVII FIT, Karpacz 2003

Wiktor Zychla, Wojciech Tomanik

Uniwersytet Wrocławski
Instytut Informatyki
13 grudnia 2003

Non-Interference

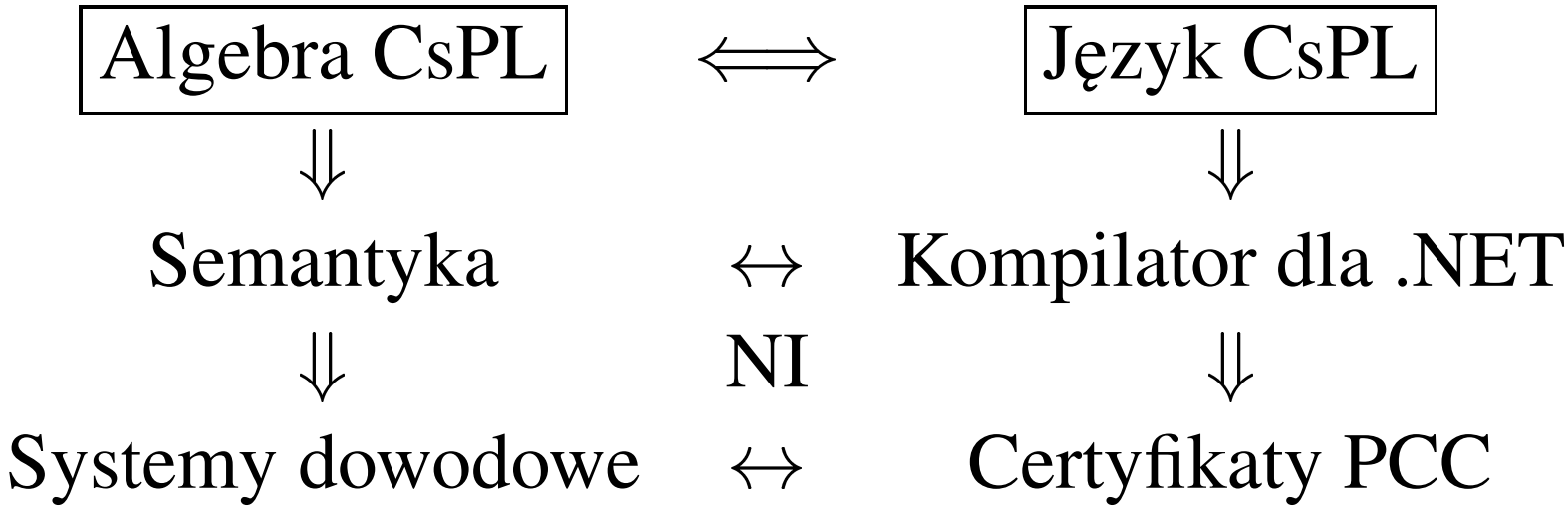
- Mamy dany model programu/systemu, złożonego z *n* równolegle działających komponentów.
- Wszystkie akcje klasyfikujemy jako jawne (bezpieczne, publiczne, L) lub tajne (niebezpieczne, prywatne, H). Podobnie użytkowników systemu dzielimy na użytkowników L i H.
- Dwa poziomy dostępu wystarczą do analizy bardziej złożonych scenariuszy.
- Taką klasyfikację traktujemy jako pewną *polisę bezpieczeństwa*.
- Klasyfikacja akcji jest punktem wyjścia do dalszej analizy systemu. Non-Interference nie narzuca tutaj jednak żadnych ograniczeń.

Non-Interference, definicja

- System nazwiemy bezpiecznym w sensie Non-Interference, jeśli nie ma żadnej interferencji między akcjami jawnymi i tajnymi, to znaczy że obserwując wykonanie się swojego programu, użytkownik L nie potrafi wydedukować czy w systemie wykonują się jakiekolwiek akcje tajne

Okazuje się, że w pewnych przypadkach jest to polisa zbyt silna. Mimo to, Non-Interference potrafi wykrywać wiele jawnych oraz niejawnych kanałów możliwego wycieku informacji.

CsPL - system kompletny



Algebra CsPL a algebra SPA

Prosta algebra SPA:

$$E ::= \underline{0} \mid \mu.E \mid \overline{\mu}.E \mid E+E \mid E|E \mid E\backslash\backslash L \mid E[f]$$

Algebra CsPL:

$$\begin{cases} P_1(x_1,\ldots,x_k) = E_1 \\ \vdots \\ P_n(x_1,\ldots,x_l) = E_n \end{cases}$$

gdzie

$$\begin{aligned} E_i \quad ::= \quad & \underline{0} \mid (\mu?x).E_i \mid (\mu!x).E_i \mid E_i+E_i \mid E_i|E_i \mid \\ & E_i\backslash\backslash L \mid E_i[f] \mid \text{if}\ldots\text{else} \mid P_j(y_1,\ldots,y_m) \end{aligned}$$

Algebra CsPL a algebra SPA

- Analiza bezpośrednia procesów SPA ma złożoność **podwójnie** wykładniczą
  - Wykładnicza złożoność translacji VSPA do SPA
  - Wykładnicza złożoność ewaluacji operatora $|$
  - O(m log n) złożoność algorytmu rozstrzygania NI
- CsPL+PCC pozwala na **podwójny** unik
  - Uniknięcie translacji do rachunku niesymbolicznego
  - Wykładnicza złożoność ewaluacji operatora $|$
  - Liniowa złożoność weryfikacji dowodu NI

Kontesty algebraiczne dla NI

- NI jest klasycznie definiowana jako własność dynamiczna, może być jednak określona algebraicznie za pomocą bisymulacji
- Różne odmiany polisy NI wyraża się przez tzw. konteksty algebraiczne bisymulacji
- NNI

$$(P\backslash\backslash_I Act_H)/Act_H \approx_T P/Act_H$$

- SNNI

$$P\backslash\backslash Act_H \approx_T P/Act_H$$

- $\vdots$

Bisymulacja, badanie NI

Na termy SPA można patrzeć jak na automaty skończone o klasycznej semantyce.
Dwa termy, P i Q są bisymulacyjnie równoważne ($P \approx Q$), gdy:

$$\begin{array}{ll} \forall \mu & P \xrightarrow{\mu} P' \\ \exists Q' & Q \xrightarrow{\mu} Q' \wedge P' \approx Q' \end{array}$$

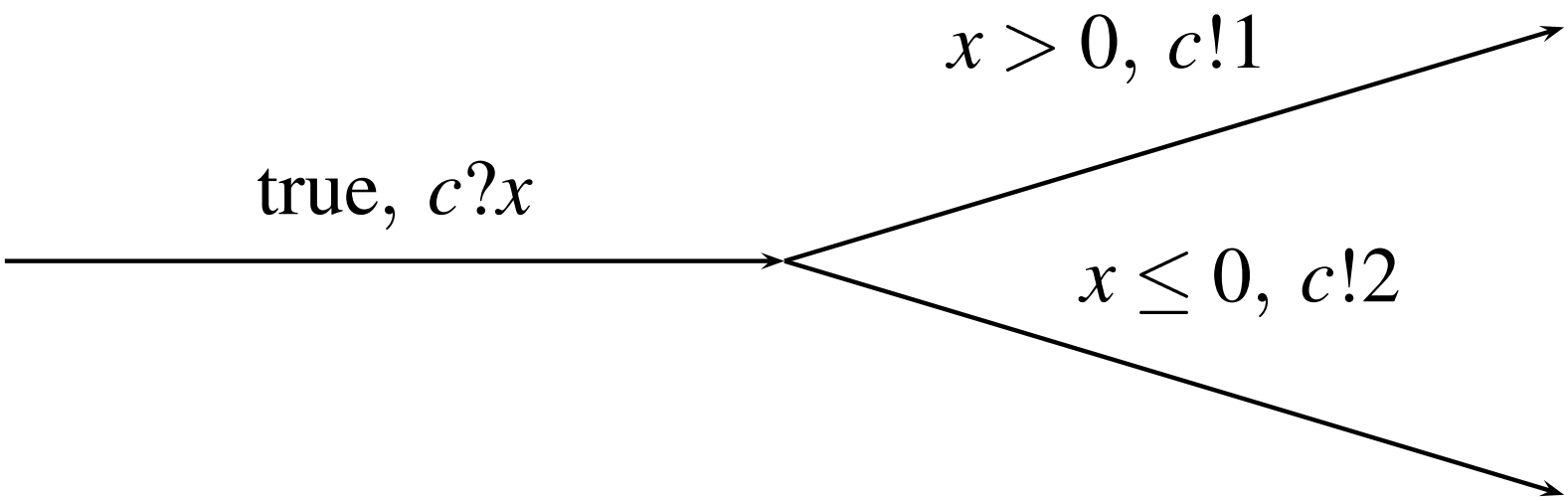
Algorytmy rozstrzygania równoważności dzielą przestrzeń stanów modelu na podzbiory stanów równoważnych.

Bisymulacja symboliczna

Modelami termów CsPL są automaty **symboliczne** o **symbolicznych** tranzycjach postaci (b, μ).
Na przykład term

$$P() = c?x.\text{if } (x > 0) \ c!1 \ \text{else } c!2$$

ma model



Bisymulacja symboliczna

Powiemy, że zbiór B wyrażeń booleowskich jest b-podziałem ($B = PART(b)$), gdy $\bigvee B = b$.

Rodzina relacji S $= \{ S^b, b$ - wyrażenie booleowskie $\}$ jest bisymulacją symboliczną, gdy

$$\forall (t,u) \in S^b \qquad t \overset{b_1,\alpha}{\rightarrow} t' \Rightarrow \exists PART(b \wedge b_1)$$

$$\forall b' \in PART(b \wedge b_1) \quad \exists b_2, \alpha', u'$$

$$b' \models b_2, \alpha =^{b'} \alpha', u \overset{b_2,\alpha'}{\rightarrow} u' \wedge$$

$$(t',u') \in S^{b'}$$

Systemy dowodowe: termy

System dla formuł postaci $P \simeq Q$.

$$\text{EQ} - \text{EQUIV} \qquad \frac{}{\mathcal{T} \simeq \mathcal{T}} \quad \frac{\mathcal{T} \simeq \mathcal{U}}{\mathcal{U} \simeq \mathcal{T}} \quad \frac{\mathcal{T} \simeq \mathcal{U},\ \mathcal{U} \simeq \mathcal{V}}{\mathcal{T} \simeq \mathcal{V}}$$

$$\text{EQ} - \text{PREFIX} \qquad \frac{\mathcal{T} \simeq \mathcal{U}}{\alpha.\mathcal{T} \simeq \alpha.\mathcal{U}}$$

$$\text{EQ} - \text{CHOICE} \qquad \frac{}{\mathcal{T} \simeq \mathcal{T} + \mathcal{T}} \quad \frac{\mathcal{T} \simeq \mathcal{T}'}{\mathcal{T} + \mathcal{U} \simeq \mathcal{T}' + \mathcal{U}}$$

$$\text{EQ} - \text{SIMPLE} \qquad \frac{t \simeq u}{\{A = t;\ A\} \simeq \{B = u;\ B\}}$$

Systemy dowodowe: rekursja

- Rekursja jest najczęściej definiowana przy pomocy operatora punktu stałego *fix*, gdzie przyjmuje się, że $fixX.F \approx F[fixX.F/X]$
- W takim ujęciu rekursji trudno jest modelować systemy wielokomponentowe. Ta trudność jest jedną z motywacji dla składni CsPL.

REC
$$\frac{}{\{X_1,\ldots,X_k;X_s\}\simeq\{X_1=x_1[\#X/X_S],\ldots,X_k=x_k[\#X/X_S],\#X=x_s[\#X/X_S];X_s\}}$$

RECPREFIX
$$\frac{\{X_1,\ldots,X_k;X_s\}\simeq\{Y_1,\ldots,Y_m;Y_t\}}{\{X_1=x_1[X_s/\mu.0],\ldots,X_k=x_k[X_s/\mu.0];X_s\}\simeq\{Y_1=y_1[Y_t/\mu.0],\ldots,Y_m=y_m[Y_t/\mu.0];Y_t\}}$$

RECEXPAND
$$\frac{\{X_1,\ldots,X_k;X_s\}\simeq\{Y_1=y_1[X_s/Y_t],\ldots,Y_m=y_m[X_s/Y_t],X_1,\ldots,X_k;Y_t\}}{\{X_1=x_1,\ldots,X_k=x_k;X_s\}\simeq\{Y_1=y_1,\ldots,Y_m=y_m;Y_t\}}$$

Systemy dowodowe: modele

System dla formuł postaci $\langle \mathcal{T} \rangle \vdash A \sim B$

$$\text{M} - \text{BISIM} \quad \frac{\bigwedge_{\mu}\left(\begin{array}{l} \bigwedge_{Y_i}((X_i, Y_i, \mu) \in \langle \mathcal{T} \rangle \Rightarrow \bigvee_{Y_j}(X_j, Y_j, \mu) \in \langle \mathcal{T} \rangle \wedge \langle \mathcal{T} \rangle \vdash Y_i \sim Y_j) \\ \bigwedge_{Y_j}((X_j, Y_j, \mu) \in \langle \mathcal{T} \rangle \Rightarrow \bigvee_{Y_i}(X_i, Y_i, \mu) \in \langle \mathcal{T} \rangle \wedge \langle \mathcal{T} \rangle \vdash Y_i \sim Y_j) \end{array}\right)}{\langle \mathcal{T} \rangle \vdash X_i \sim X_j}$$

$$\text{M} - \text{REC} \quad \frac{\begin{array}{c} [\langle \mathcal{T} \rangle \vdash X_i \sim X_j] \\ \vdots \\ \langle \mathcal{T} \rangle \vdash X_i \sim X_j \end{array}}{\langle \mathcal{T} \rangle \vdash X_i \sim X_j}$$

PCC dla NI, termy

Aby zbudować infrastrukturę PCC dla NI trzeba pokazać jak budować warunki weryfikacyjne i jak ich dowodzić.
Dla logiki dowodzącej równość termów mówimy:

$$V(P) = P\backslash\backslash Act_H \sim P/Act_H$$

Tw.: $V(P)$ jest dowodliwy $\Leftrightarrow$ P ma własność NI.
Dowód: $V(P)$ definiuje kontekst algebraiczny dla Non-Interference.

PCC dla NI, modele

Dla logiki dowodzącej równość modeli definiujemy modele dla kontekstów algebraicznych:

$$\mathcal{P}_1 = \langle P \backslash\backslash Act_H \rangle, \mathcal{P}_2 = \langle P / Act_H \rangle$$

Warunek weryfikacyjny definiujemy jako:

$$V(P) = \mathcal{P}_1 \cup \mathcal{P}_2 \vdash init(\mathcal{P}_1) \sim init(\mathcal{P}_2)$$

gdzie $\mathcal{P}_1 \cup \mathcal{P}_2$ jest mnogościową sumą $\mathcal{P}_1$ and $\mathcal{P}_2$.
Tw.: $V(P)$ jest dowodliwy $\Leftrightarrow$ P ma własność NI.
Dowód: $\mathcal{P}_1 \cup \mathcal{P}_2 \vdash init(\mathcal{P}_1) \sim init(\mathcal{P}_2)$ oznacza, że stany początkowe modeli $\mathcal{P}_1$ i $\mathcal{P}_2$ są równoważne, więc $P \backslash\backslash Act_H \sim P / Act_H$.

Przykładowy program w CsPL

```
chanof int   comm1; chanof comm1 comm2
var comm1 c1;       var comm2 c2;

env Sender( 5 ) | ...

proc Sender( int x ) {
  %{ Console.WriteLine( "{0}", x );
     comm1 privatec = null; int y;
  %}
  c2 ? privatec;
  ( privatec ? y ) {
    Sender( y );
  } +
  ( c1 ? y ) {
    Sender( y+1 );
  };
}

proc ...
```

Coq - definicje i reguły

- Zmienne i akcje

```
Axiom Channels : Set.
Axiom Vars :Set.

Inductive Actions : Set := tau: Actions
| inCh : Channels -> Vars -> Actions
| outCh: Channels -> Vars -> Actions.
```

- Procesy

```
Inductive SProcessBody : Set := nil : SProcessBody
                    | cons:  Actions -> SProcessBody -> SProcessBody
                    | plus: SProcessBody -> SProcessBody -> SProcessBody
                    | nameProc: SProcessName -> SProcessBody.
Inductive SProcess: Set := proc: ProcNames -> SProcessBody -> SProcess.
```

Coq - jak używać

- Certyfikowanie
  - Translacja programu CsPL -> Coq
  - Wczytanie logiki i opisu programów.
  - Wprowadzenie dowodów o równoważności.
  - Utworzenie dowodów.
- Weryfikacja
  - Wczytanie logiki
  - Wczytanie opisu programów
  - Wczytanie twierdzeń o równoważności wraz z dowodami.

CsPL $\leftrightarrow$ Coq

`proc Nadawca {`
`  [...]`
`}`
	`Definition NadawcaBody := (SimpleP.cons [..]).`
	`Definition NadawcaProcess :=`
	`(proc Nadawca NadawcaBody).`

`int x;` `[...]`	`(SimpleP.cons SimpleP.tau [...]).`
`c ! x;` `[...]`	`(SimpleP.cons (SimpleP.outCh c x) [...]).`

Przykład - Certyfikat

```
Load SimpleP.

Load testw02.

Lemma m1:
(SimpleP.Equiv Nadawca1 Nadawca2).

Proof.
Unfold Nadawca1.Unfold Nadawca2.
Apply SimpleP.equiv_body.
Unfold NadawcaBody1.Unfold NadawcaBody2.
Apply SimpleP.prefix.Apply SimpleP.choice_m.
Qed.
```