Formal Verification

Securitate informatică

March 18, 2020

Mihai-Lica Pura

Cuprins

- motivația verificării formale a protocoalelor de securitate;
- verificarea formală, pe scurt:
  - **modelarea** sistemelor funcționale, paralele și distribuite (cu accentul pe protocoale de securitate),
  - **specificarea** cerințelor pentru aceste sisteme,
  - **verificarea** dacă cerințele sunt sau nu îndeplinite de către sisteme.
- verificarea formală a protocoalelor de securitate.

Definiție

- "... actul demonstrării sau infirmării corectitudinii algoritmilor care compun un sistem, având în vedere o anumită specificare formală sau proprietate, și folosind metode matematice formale"
- "... limbaje, tehnici și unelte matematice pentru specificarea și verificarea sistemelor"
- "un set de unelte și notații, cu o semantică formală, folosite pentru a specifica neambiguu cerințele unui sistem, care admit demonstrarea de proprietăți ale acelei specificări și demonstrarea corectitudinii unei implementări în raport cu acea specificare"

Exemplu - Prânzul filosofilor

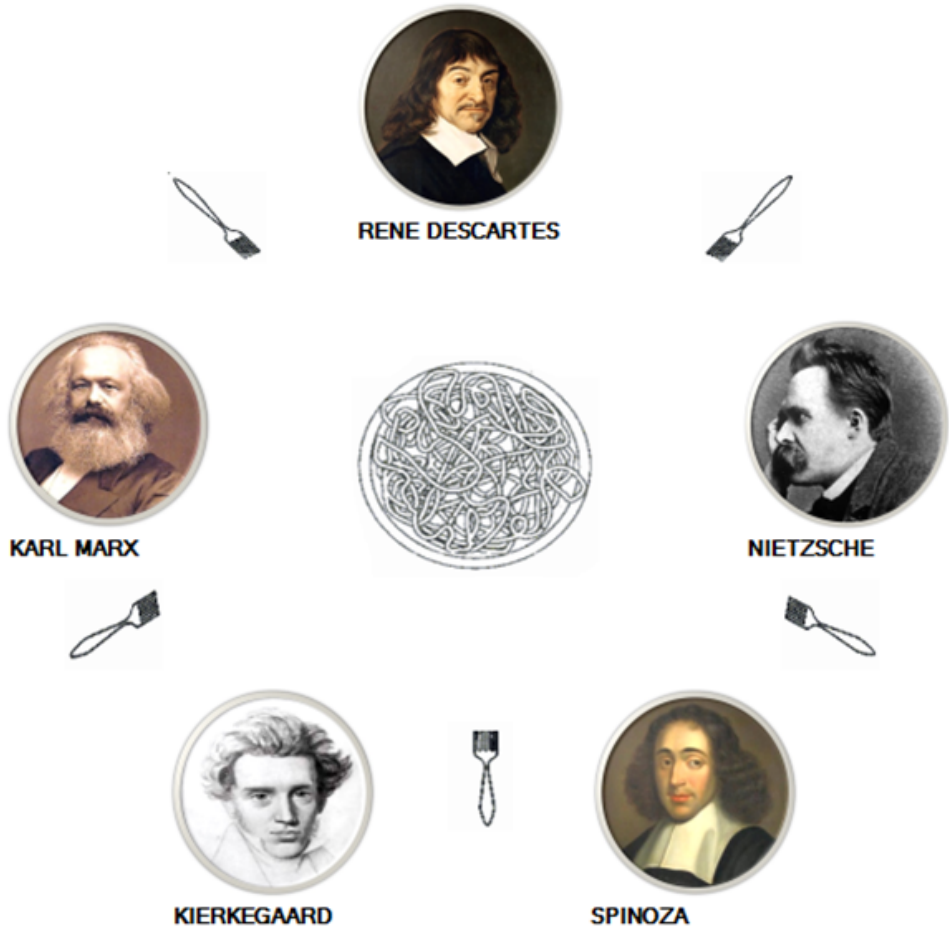


Figure 1: http://ingenieroprendiz4ever.blogspot.ro/2012/06/la-cena-de-los-filosofos-aplicacion-de.html

Exemplu - Prânzul filosofilor

- Inițial filosofii gândesc. Atunci când unui filosof i se face foame:
- filosoful ia furculița din stânga,
- filosoful ia furculița din dreapta și apoi începe să mănânce.
- Când filosoful se satură, pune pe masă ambele furculițe și apoi o ia de la capăt.

Cum se poate modela acest sistem?

Rețele Petri

- formalism elementar care permite modelarea sistemelor paralele și distribuite
- propus de Adam Petri în 1962 în teza sa de doctorat "Kommunicaktion mit Automaten"
- descrie schimbările de stare ale unui sistem prin intermediul tranzițiilor

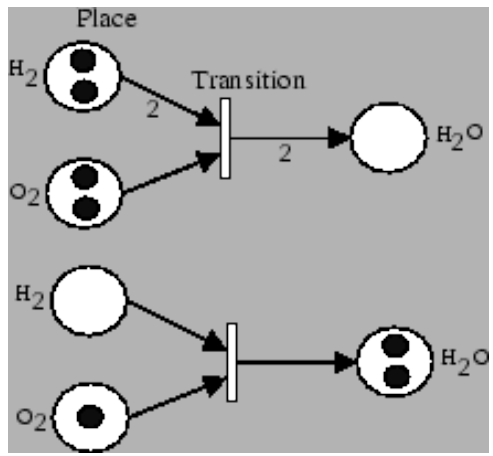


Figure 2:
https://fknussel.wordpress.com/2013/11/01/petri-nets-101/

Rețele Petri

O rețea Petri conține:

- **places** - reprezintă stările și/sau condițiile care trebuie îndeplinite înainte ca o acțiune să poată avea loc
- **transitions** - reprezintă acțiuni
- places and transitions may be connected by **directed arcs**
- places may contain **tokens** that may move to other places by executing actions

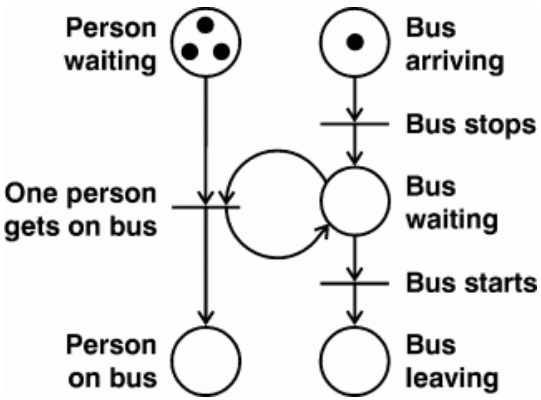


Figure 3:
`https://fknussel.wordpress.com/2013/11/01/petri-nets-101/`

Rețele Petri

Transition **One person gets on bus** may fire if there are tokens in place **Person waiting** and in place **Bus waiting**

Firing this transition once will remove a token from **Person waiting** and a token from **Bus waiting**, and will place a new token in **Person on bus** and a new token in **Bus waiting**

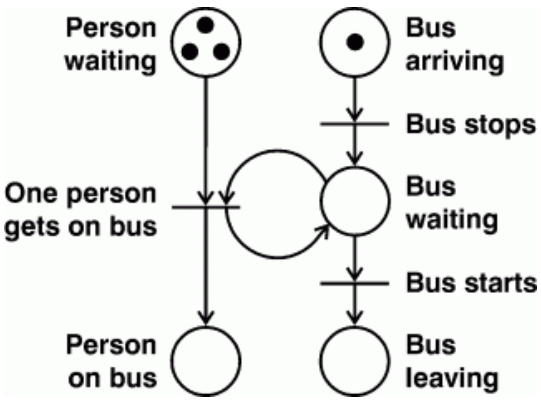


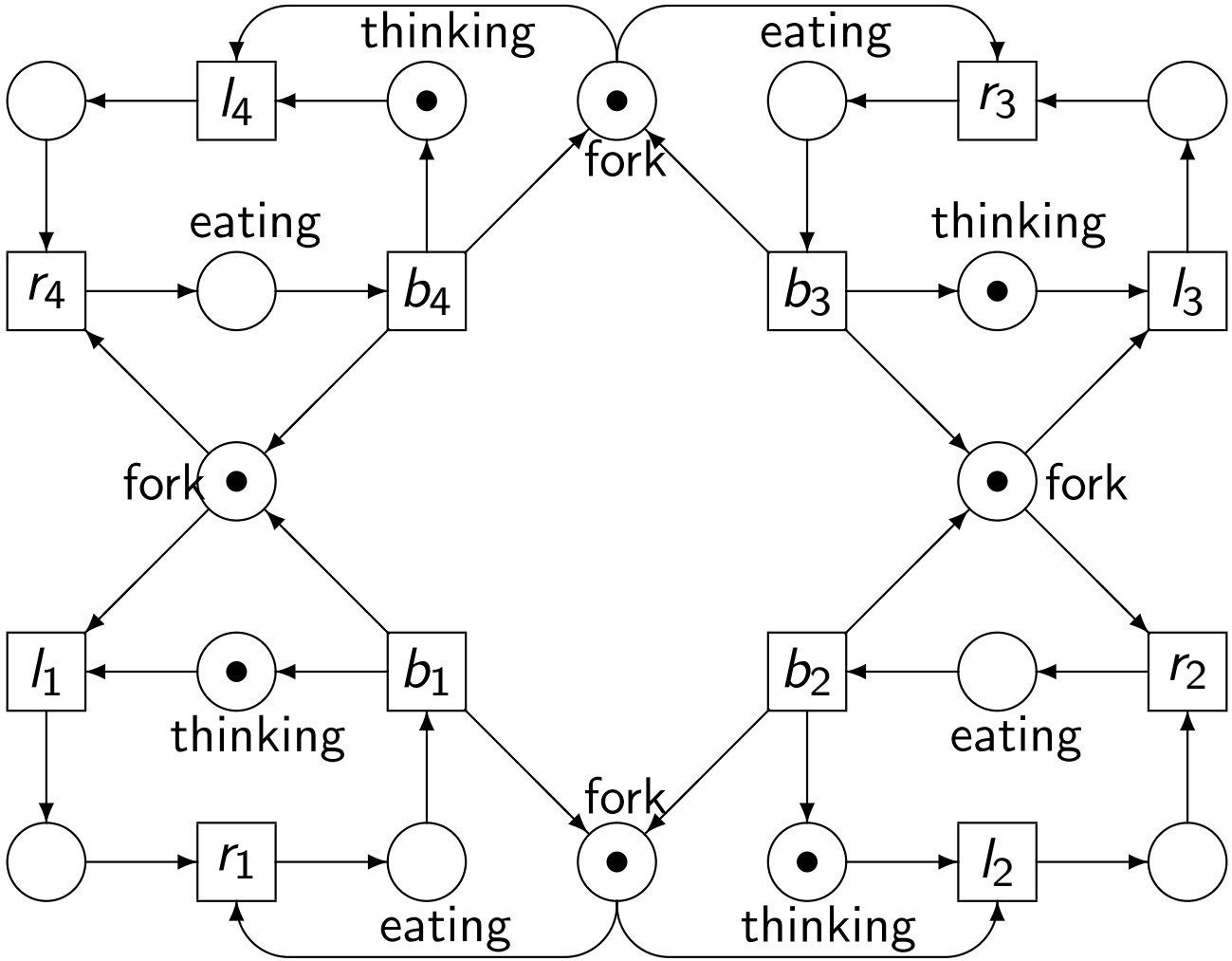
Figure 4:
`https://fknussel.wordpress.com/2013/11/01/petri-nets-101/`

Rețele Petri

Modele interactive:

`http://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/aalst/`

Exemplu - Prânzul filosofilor



Didier Buchs, cursul "Modélisation et vérification de logiciels" - Universitatea din Geneva

Prânzul filosofilor - Proprietăți

- Doi filosofi vecini pot mânca în același timp?
- Doi filosofi aflați față în față pot mânca în același timp?
- Se poate întâmpla ca filosofii să flămânzească până la moarte?
- Un anume filosof poate mânca în cele din urmă (presupunând că dorește să facă acest lucru)?

Cum pot fi exprimate aceste întrebări ca și proprietăți ale rețelei Petri care modelează sistemul?

Tipuri de rețele Petri

- only black tokens - **Place/Transition nets**
  StrataGEM `https://github.com/mundacho/stratagem`
- colored tokens - **Colored Petri nets**
  CPN Tools `http://cpntools.org/`
- tokens are terms of some Algebraic Abstract Data Types (AADT) - **Algebraic Petri nets**
  AlPiNA `http://alpina.unige.ch/`

Există multe alte tipuri de rețele Petri și instrumente asociate:
`https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html`

Algebraic Abstract Data Types - Exemplu booleans

Adt boolean
Sorts **bool**;
Generators
    true : bool;
    false : bool;

Algebraic Abstract Data Types - Exemplu naturals

Adt naturals
Sorts **nat**;
Generators
    zero : nat;
    suc : nat $\rightarrow$ nat;
Operations
    plus : nat, nat $\rightarrow$ nat;
    minus : nat, nat $\rightarrow$ nat;
    gt : nat, nat $\rightarrow$ bool;

Algebraic Abstract Data Types - Exemplu naturals (continuare)

Axioms

plus (zero, $x) = $x;
plus (suc($x), $y) = suc (plus ($x,$y));

minus($x, zero) = $x;
minus(suc($x), suc($y)) = minus($x, $y);

gt(zero, $x) = false;
gt(suc($x),zero) = true;
gt(suc($y), suc($x)) = gt ($y, $x);

Variables

x : nat;
y : nat;

Algebraic Petri net - Exemplu - Sumator

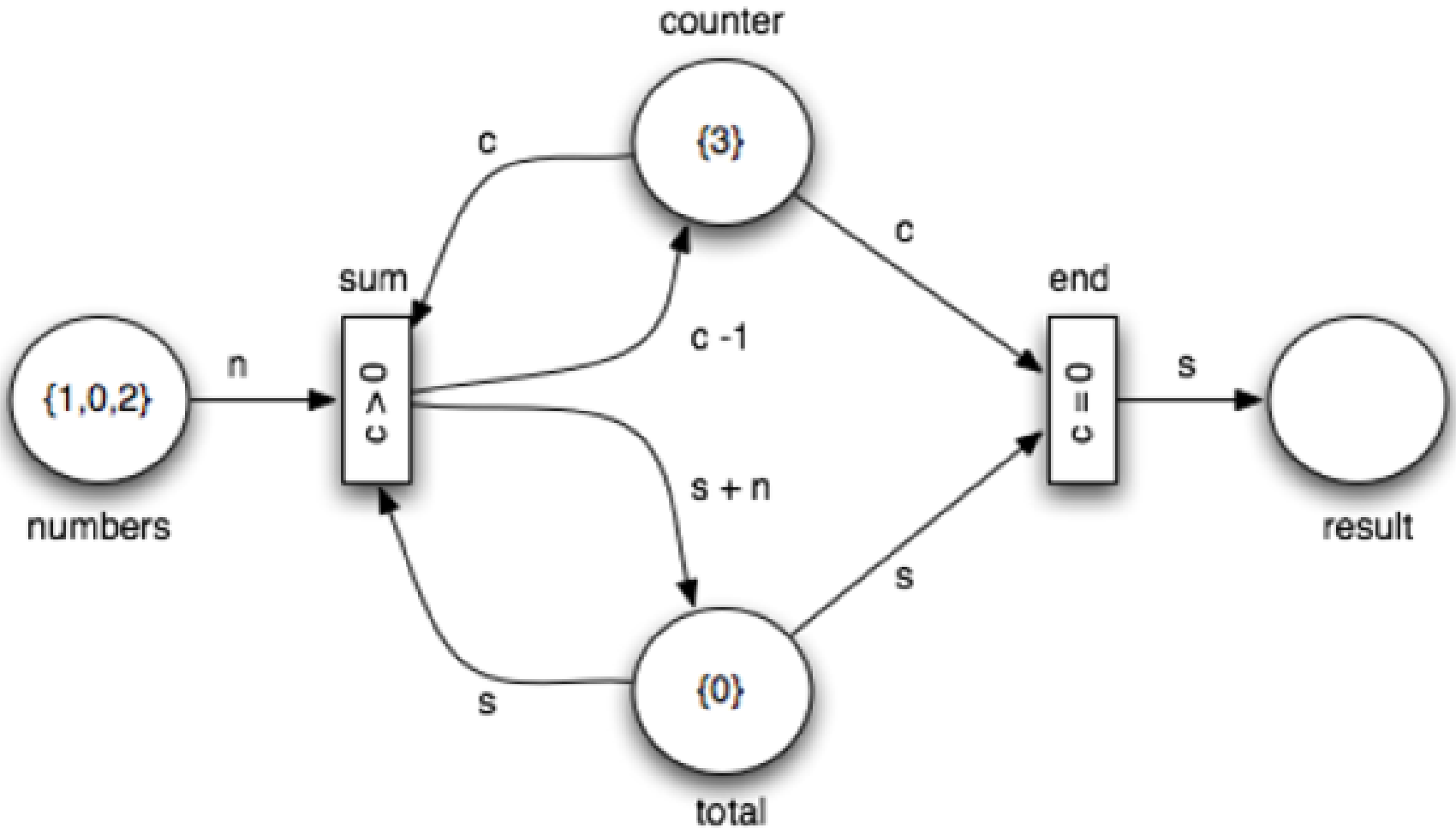


Figure 5:

Prânzul filosofilor - APN

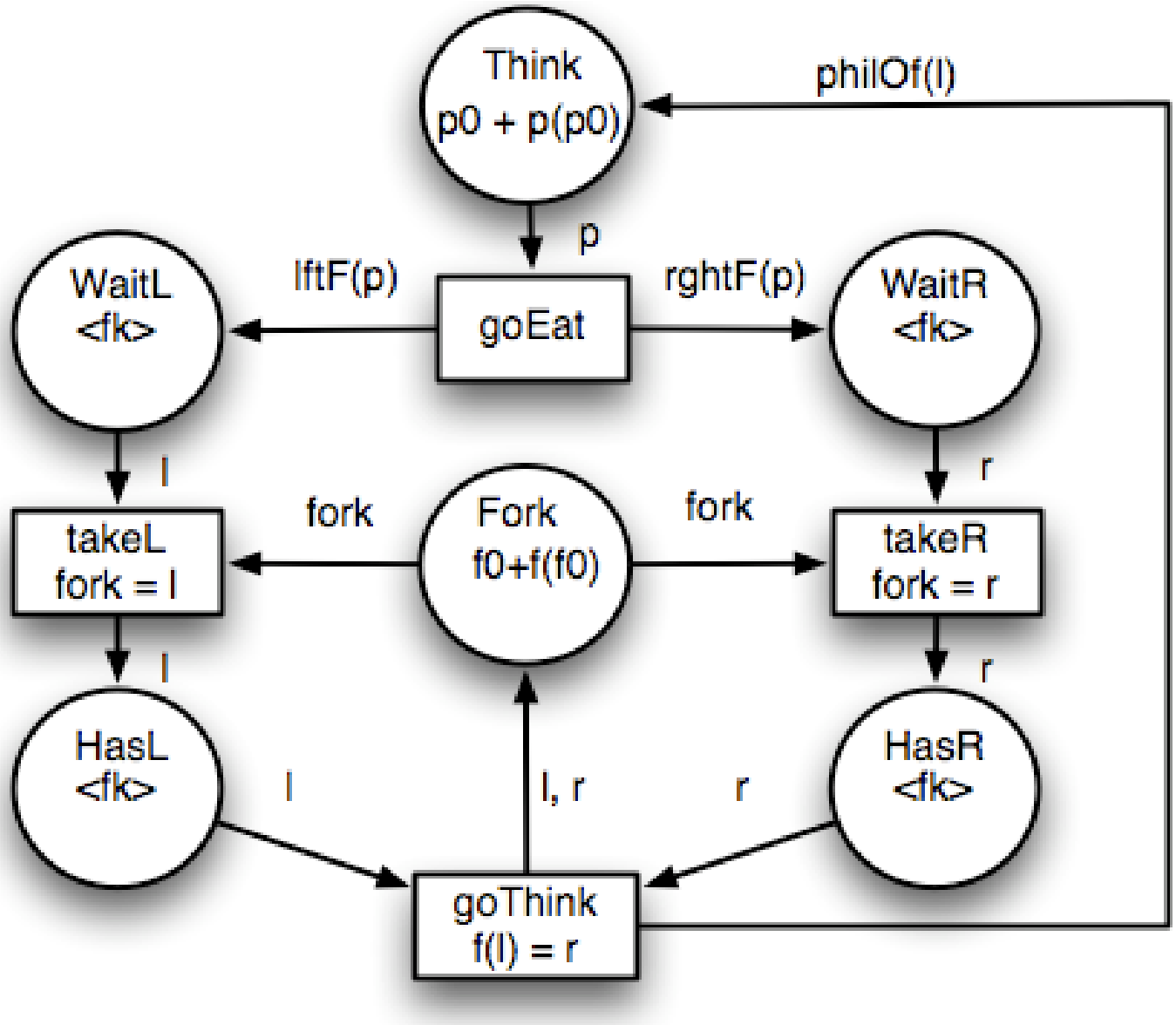


Figure 6: `http://en.wikipedia.org/wiki/Algebraic_Petri_net`

Prânzul filosofilor - AADT

Adt philosophers
Sorts **philo**;
Generators
    p0 : philo;
    p : philo $\rightarrow$ philo;
Operations
    leftFork : philo $\rightarrow$ fork;
    rightFork : philo $\rightarrow$ fork;
    philoOf : fork $\rightarrow$ philo;

Prânzul filosofilor - AADT (continuare)

Axioms

p(p(p0)) = p0;
leftFork(p0) = f0;
rightFork(p0) = f(f0);
philoOf(f0) = p0;
leftFork(p($vphilo)) = f(leftFork($vphilo));
rightFork(p($vphilo)) = f (rightFork($vphilo));
philoOf(f($vfork)) = p(philoOf($vfork));

Variables

vphilo : philo;
vfork : fork;
vl : fork;
vr : fork;

Prânzul filosofilor - AADT (continuare)

Adt fork
Sorts **fork**;
Generators
    f0 : fork;
    f : fork $\rightarrow$ fork;
Axioms
    f(f(f0)) = f0;

Construirea spațiului stărilor

- Starea unei rețele Petri este reprezentată de către distribuția jetoanelor în locuri
- Starea se schimbă de fiecare dată când se execută o tranziție
- Spațiul stărilor reprezintă mulțimea stărilor prin care trece o rețea Petri pornind din starea inițală și apoi executând toate tranzițiile posibile
- Poate fi reprezentat printr-un graf:
  - nodurile grafului corespund distribuției jetoanelor în locuri
  - arcurile grafului corespund execuției tranzițiilor

Spațiul stărilor - Prânzul filosofilor

$$\langle [p_0,\ p(p_0)], [], [], [f_0, f(f_0)], [], [] \rangle$$

Spațiul stărilor - Prânzul filosofilor

$$\langle [p(p_0)],[f_0],[f(f_0)],[f_0,f(f_0)],[],[]\rangle$$

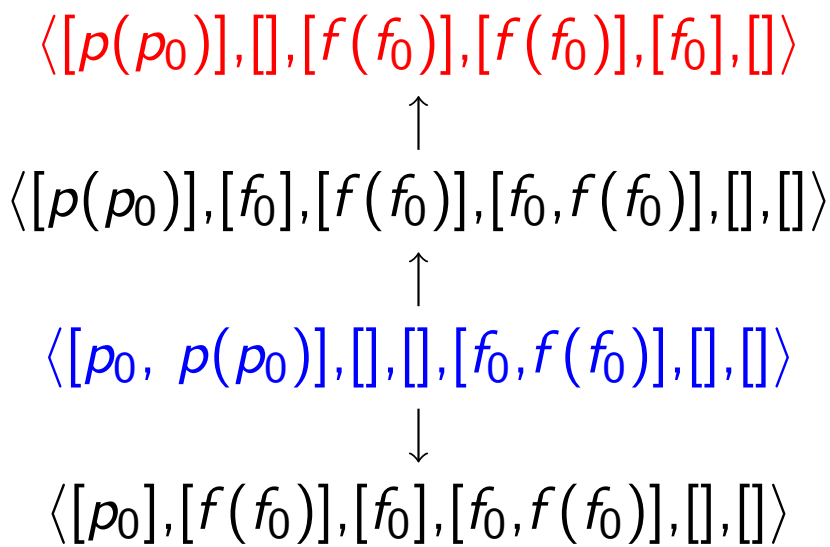
$$\uparrow$$

$$\langle [p_0,\ p(p_0)],[],[],[f_0,f(f_0)],[],[]\rangle$$

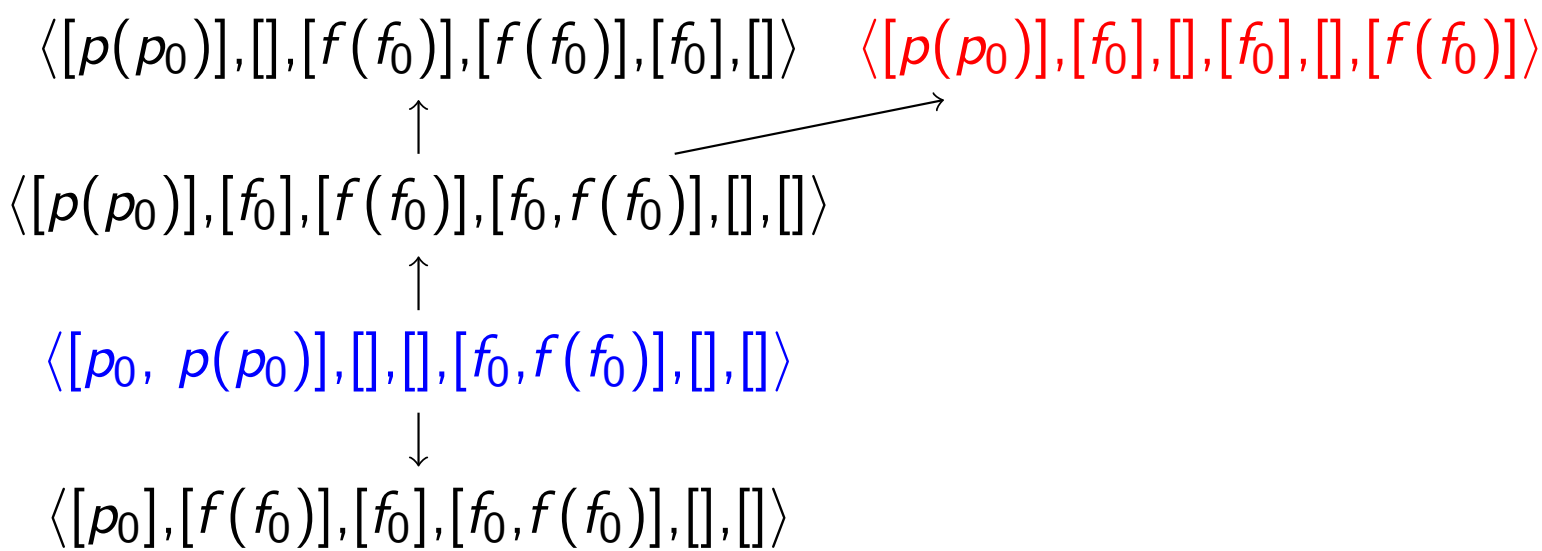
$$\downarrow$$

$$\langle [p_0],[f(f_0)],[f_0],[f_0,f(f_0)],[],[]\rangle$$

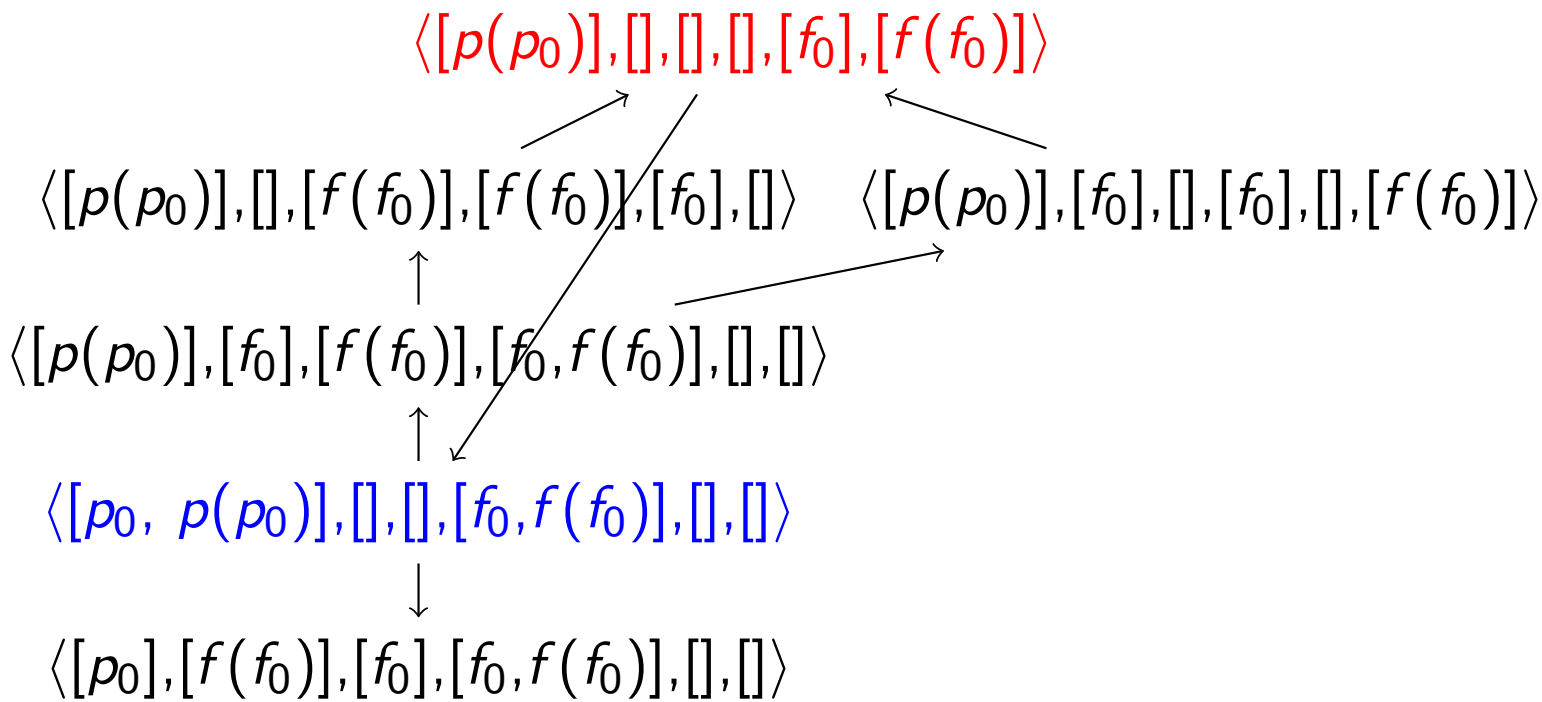
Spațiul stărilor - Prânzul filosofilor



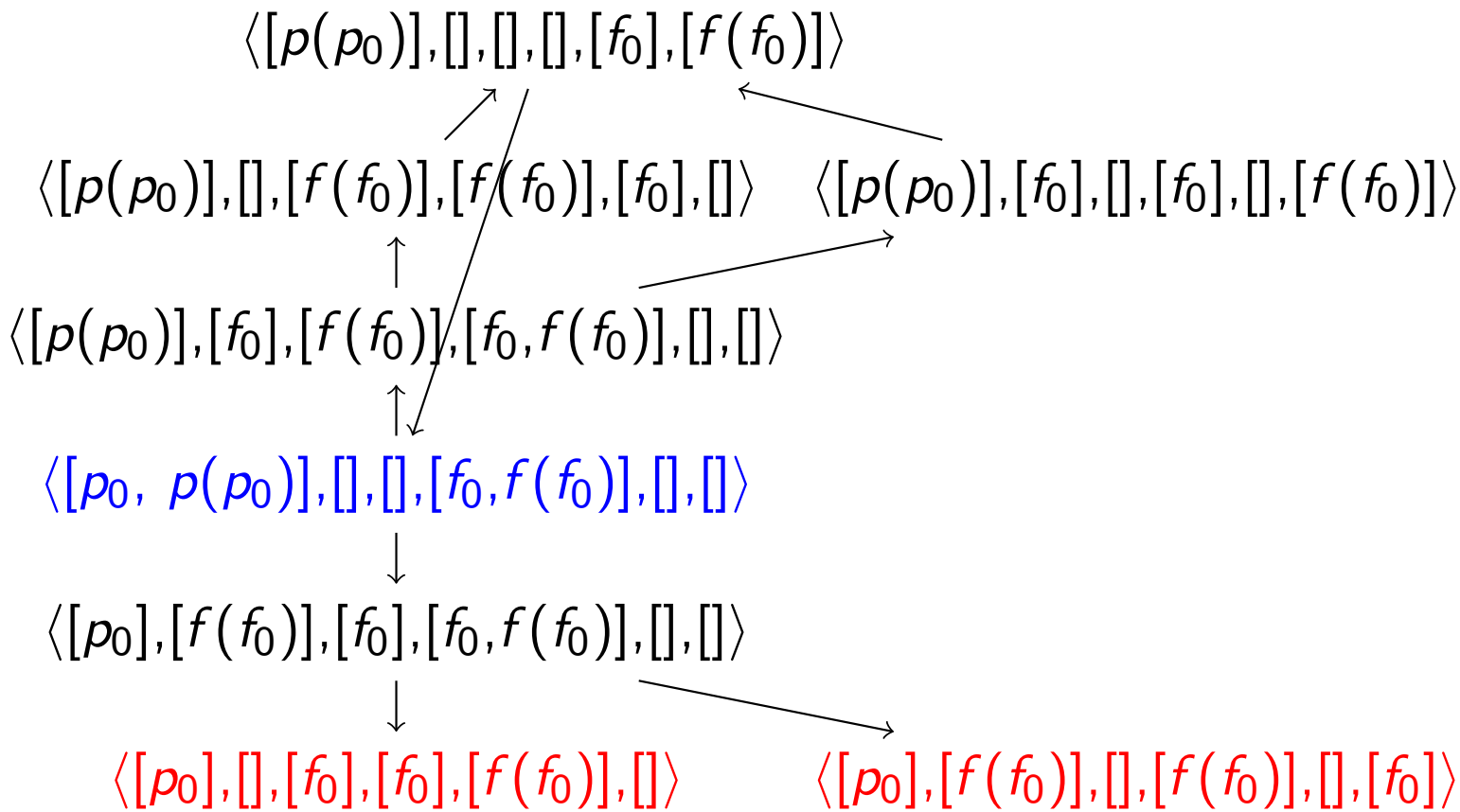
Spațiul stărilor - Prânzul filosofilor



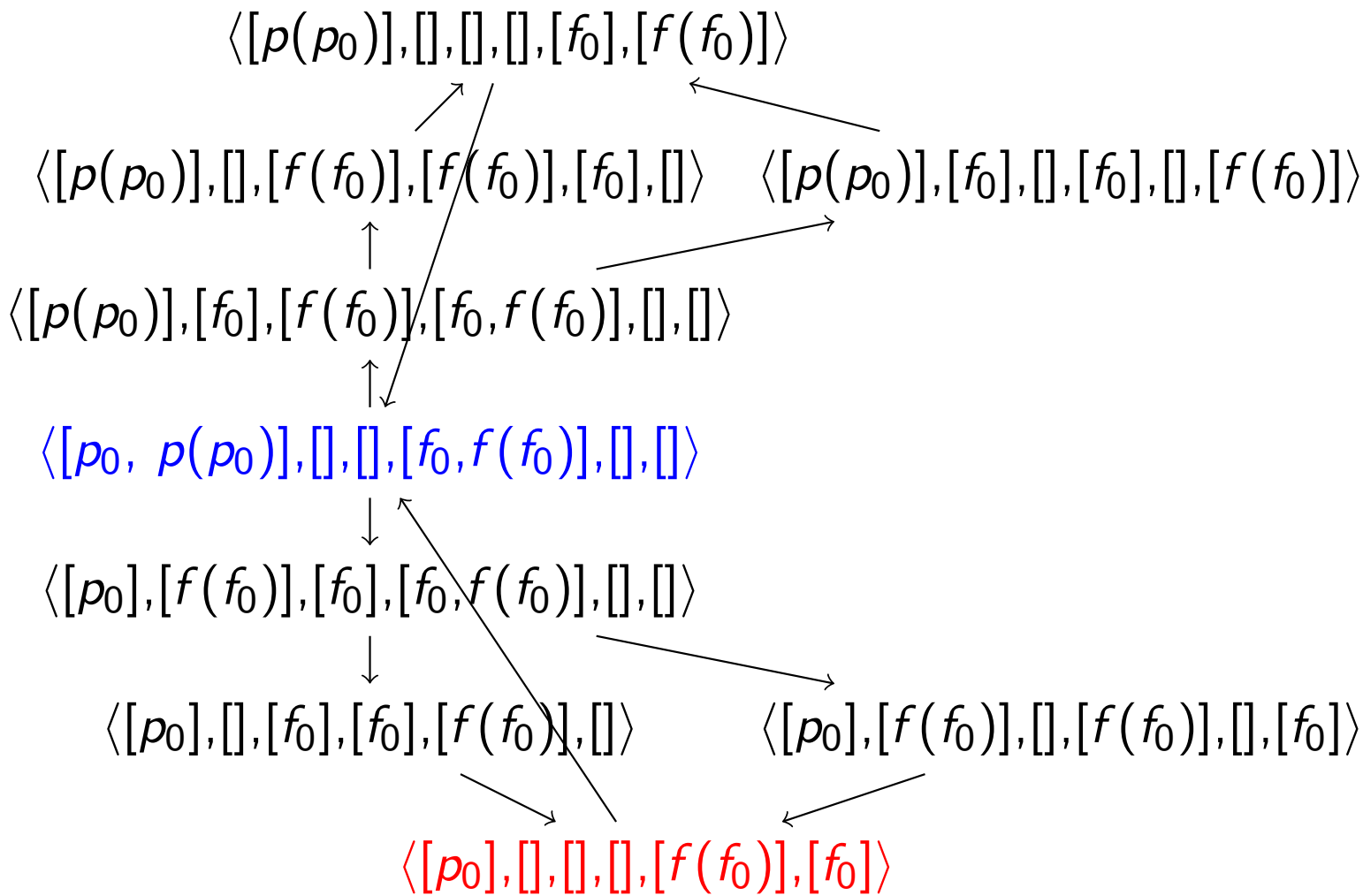
Spațiul stărilor - Prânzul filosofilor



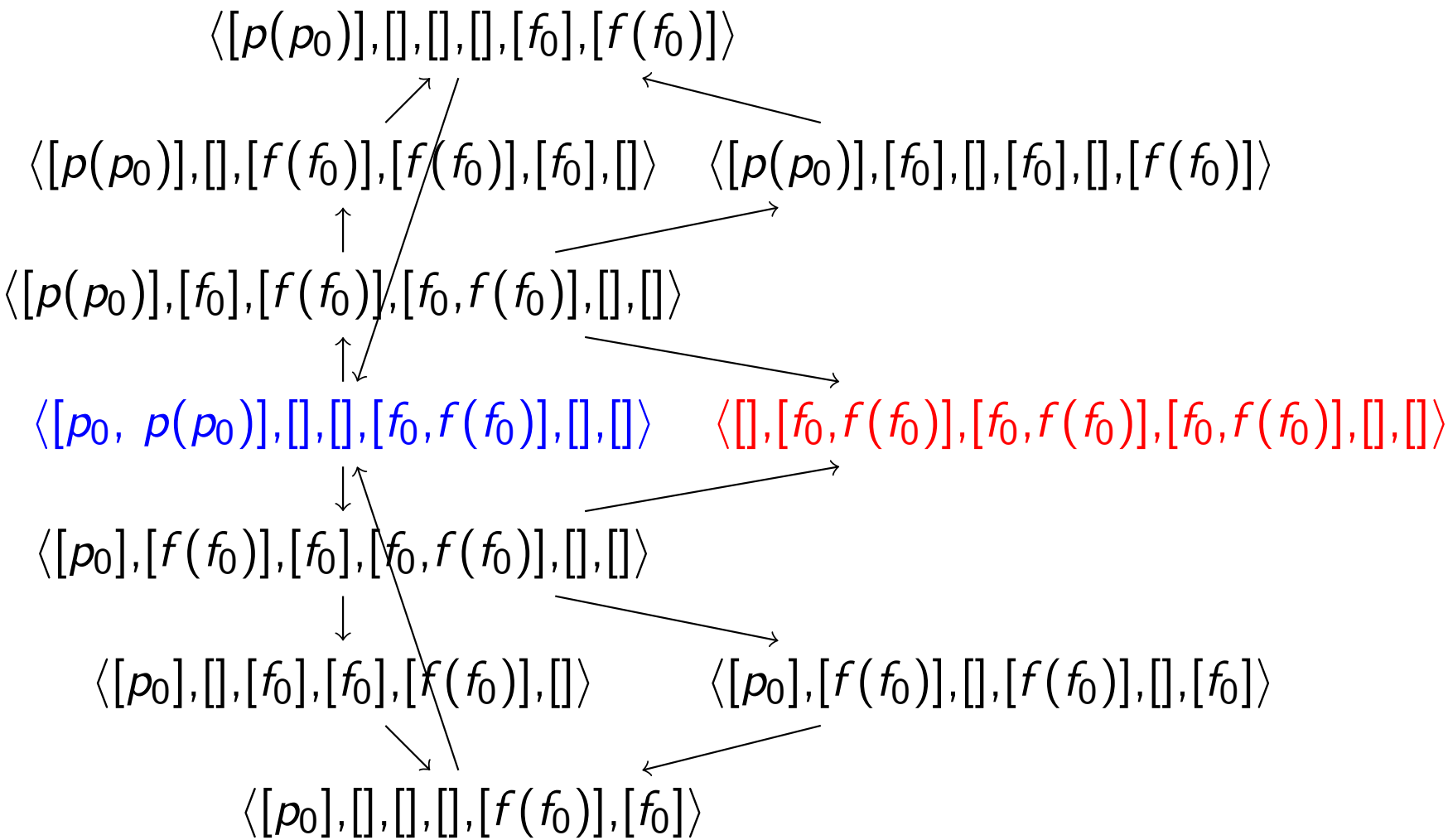
Spațiul stărilor - Prânzul filosofilor



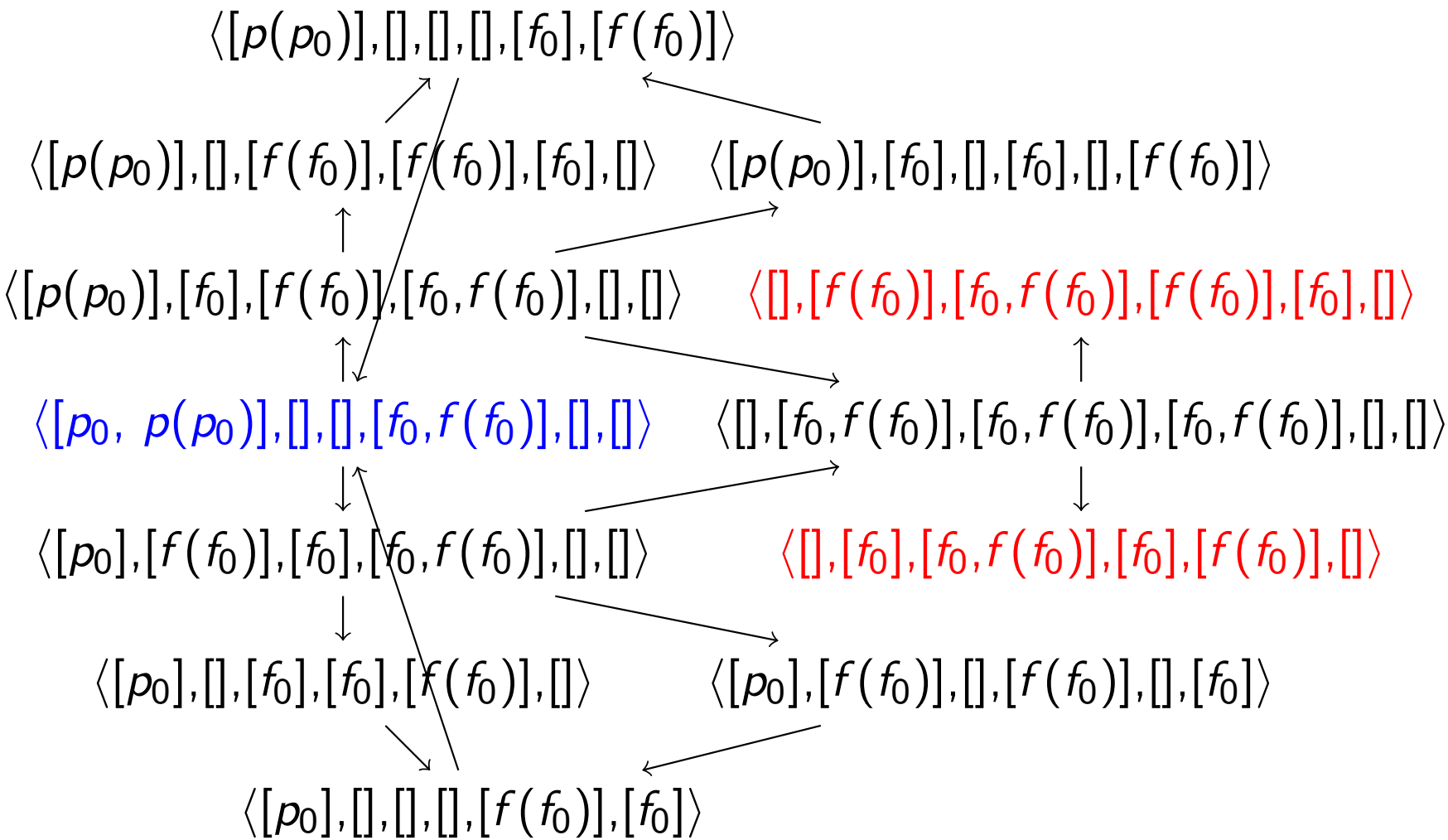
Spațiul stărilor - Prânzul filosofilor



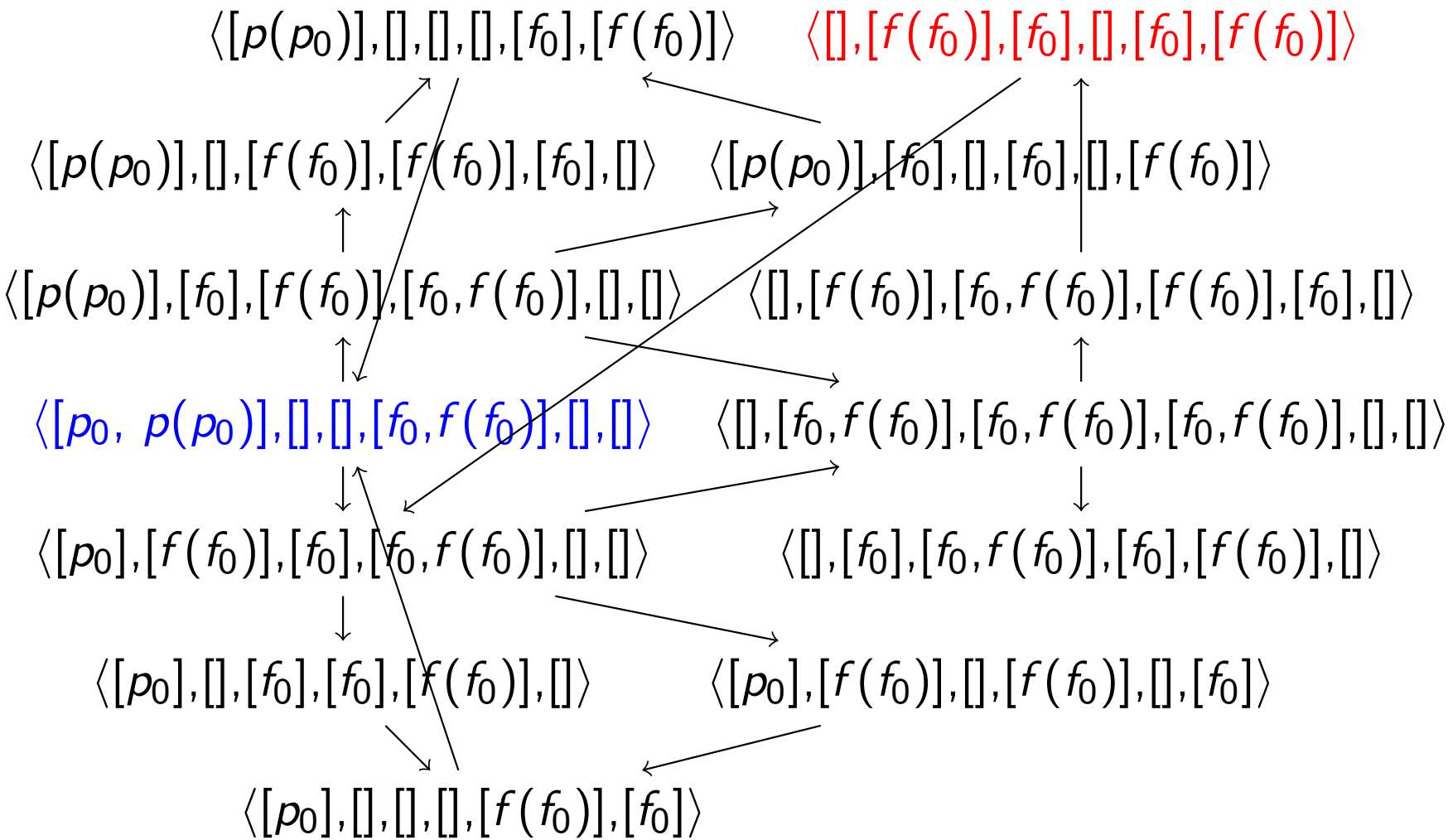
Spațiul stărilor - Prânzul filosofilor



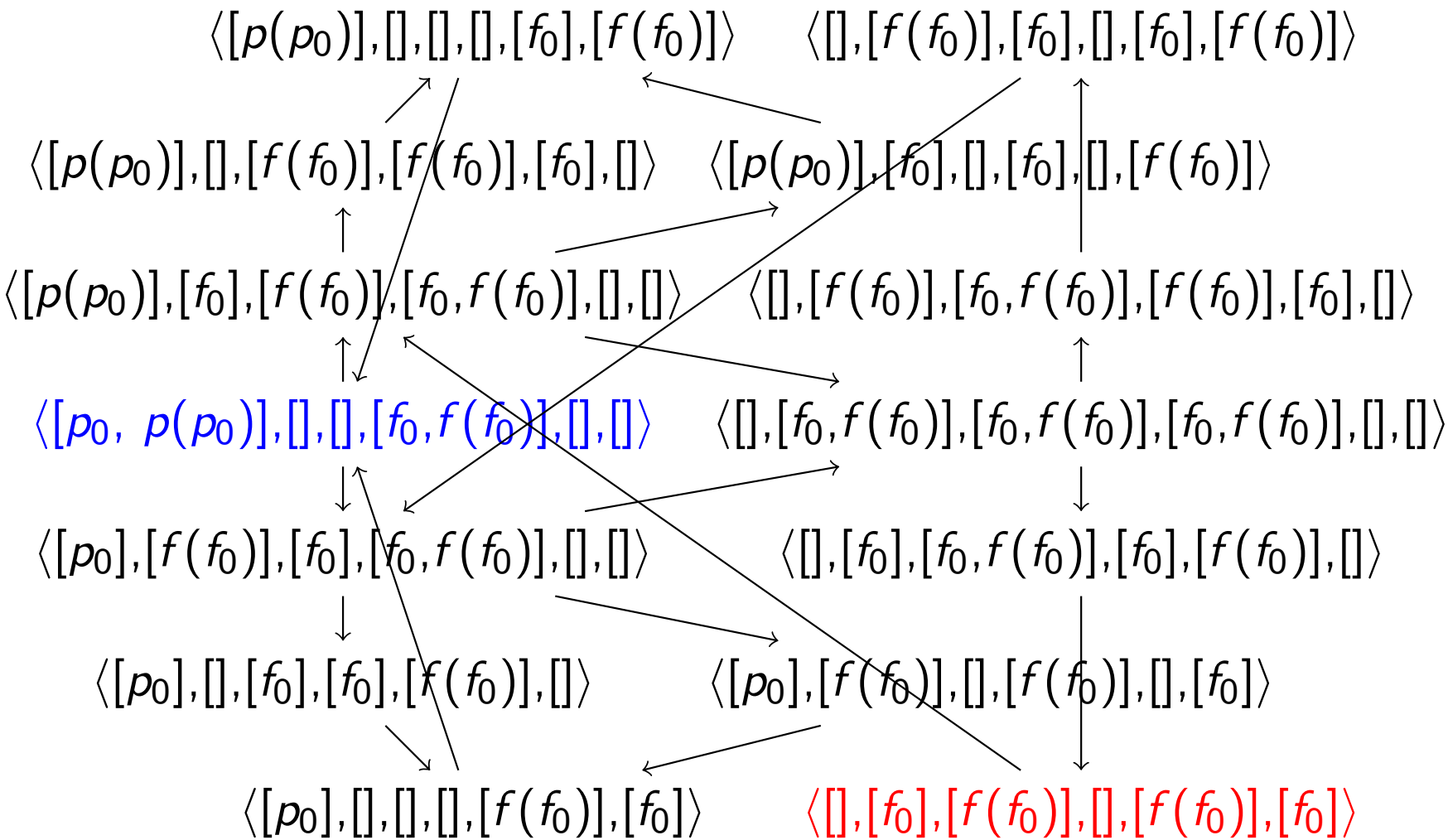
Spațiul stărilor - Prânzul filosofilor



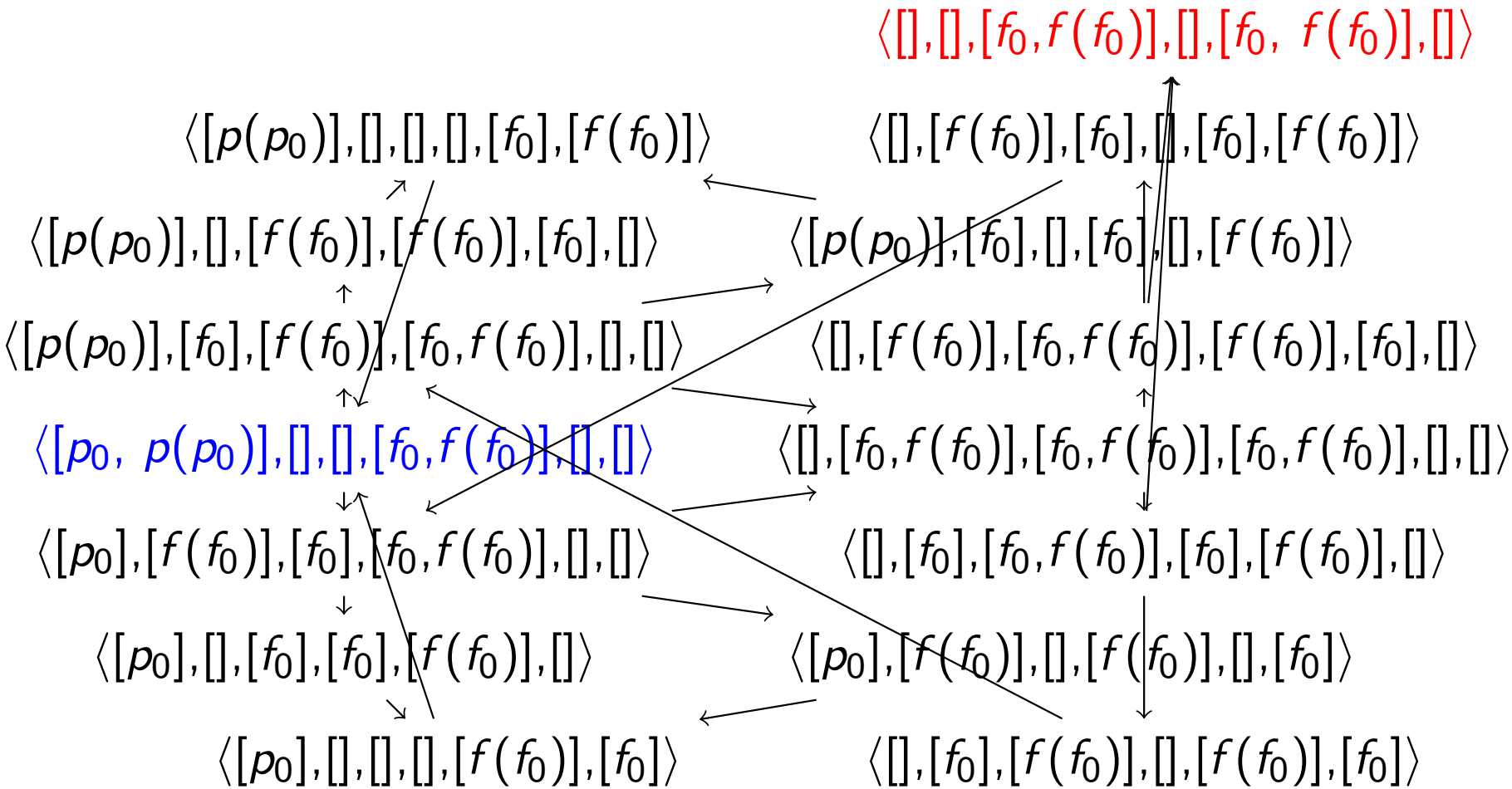
Spațiul stărilor - Prânzul filosofilor



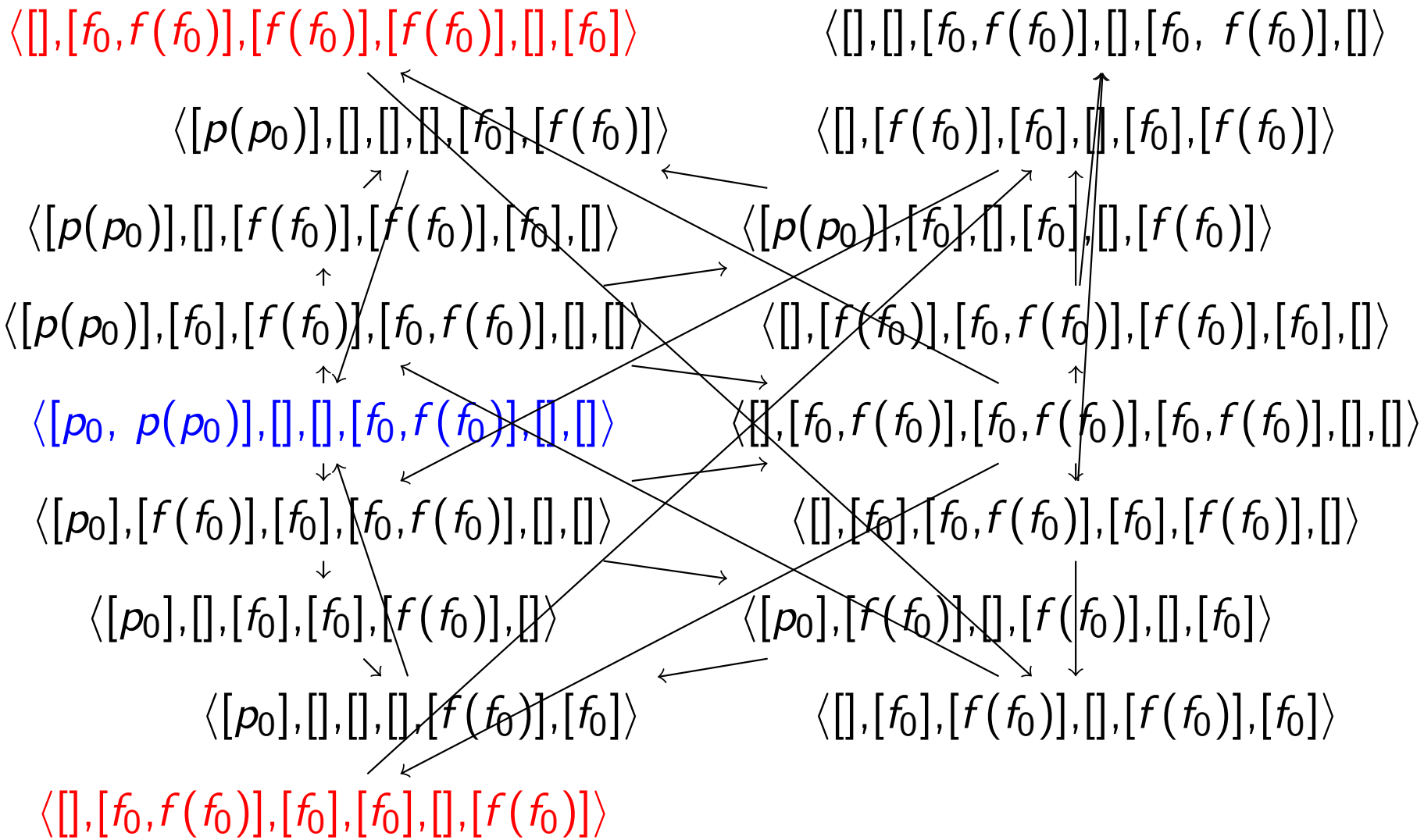
Spațiul stărilor - Prânzul filosofilor



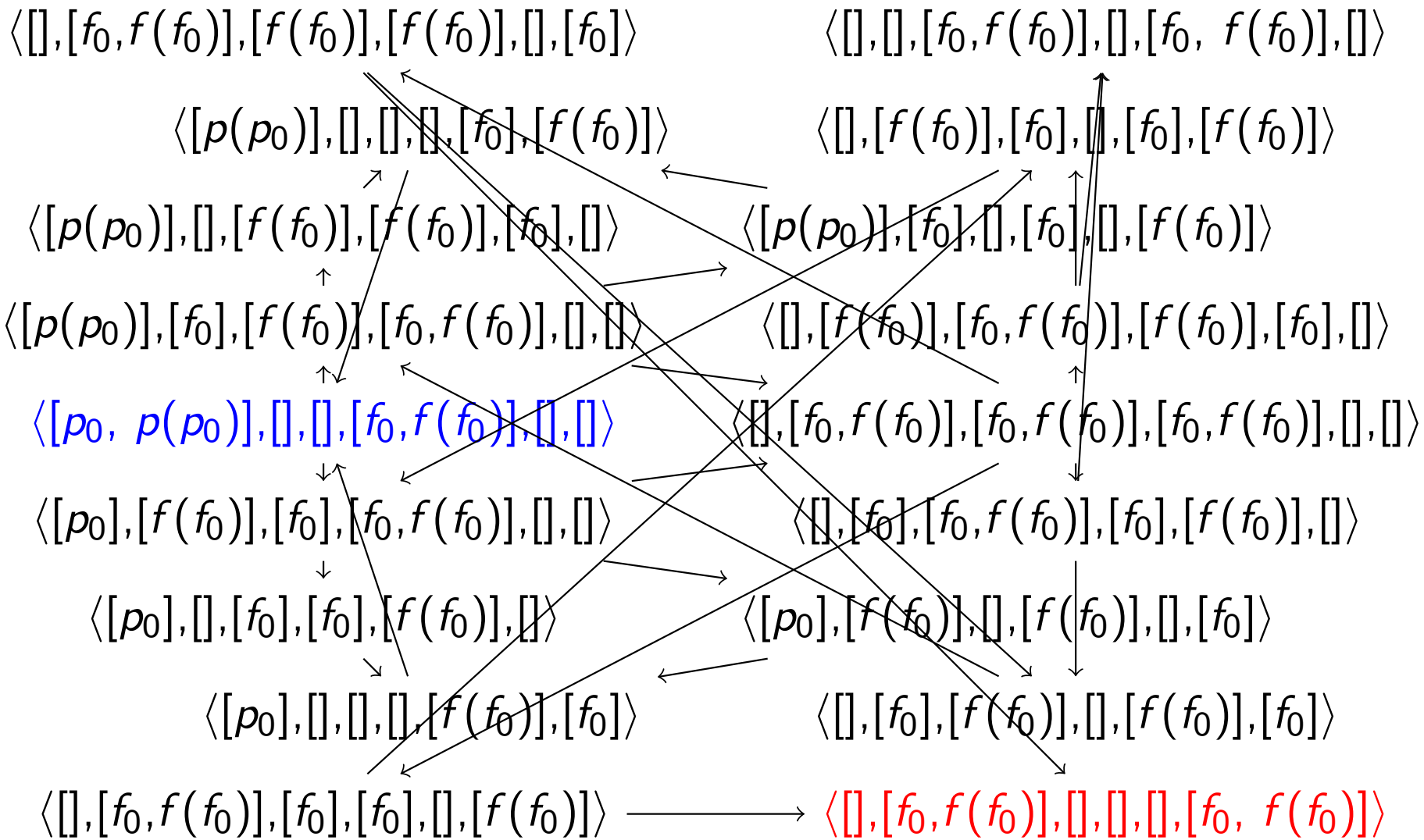
Spațiul stărilor - Prânzul filosofilor



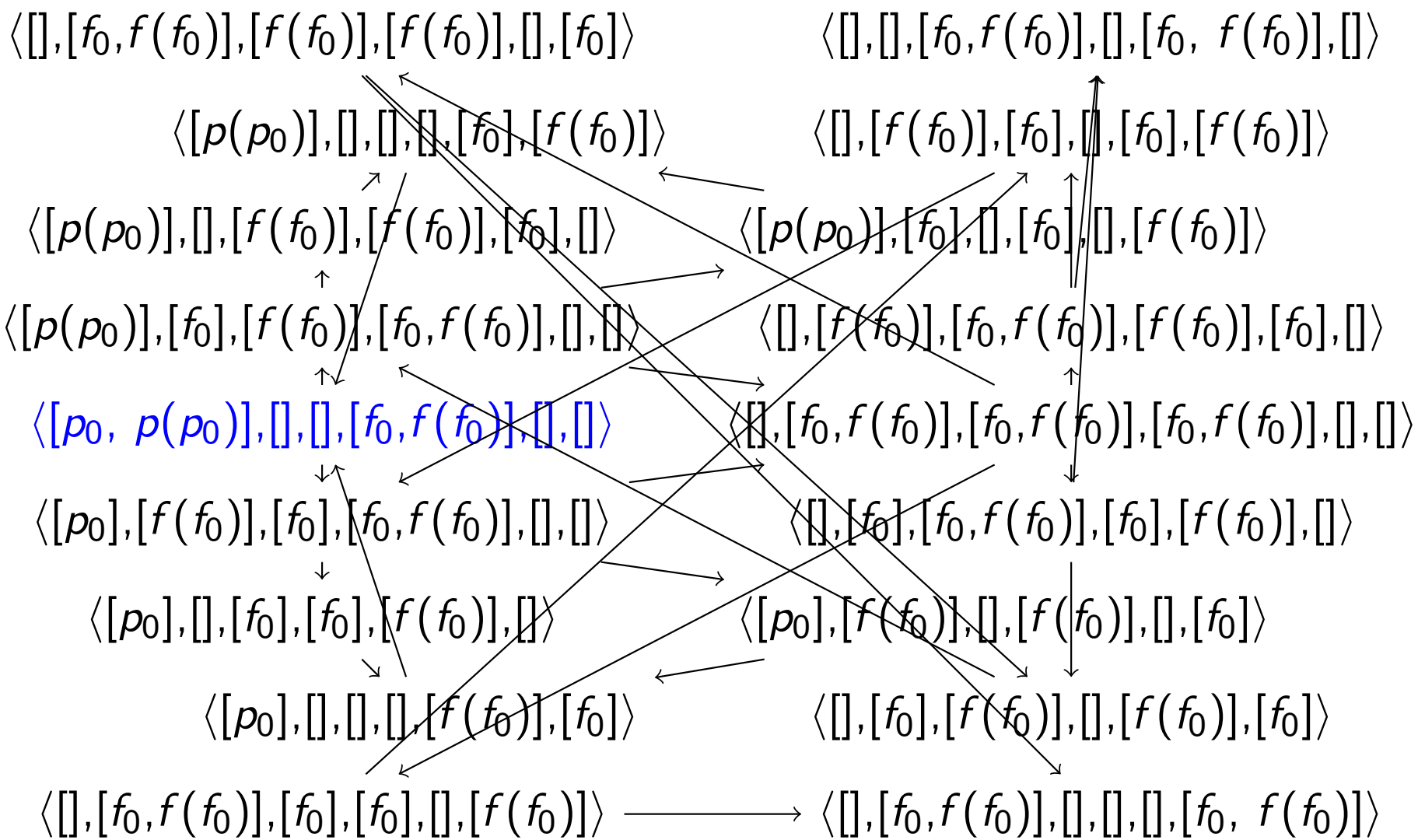
Spațiul stărilor - Prânzul filosofilor



Spațiul stărilor - Prânzul filosofilor



Spațiul stărilor - Prânzul filosofilor - 2 filosofi - 18 stări



Modelarea versus proiectarea sistemelor

Modelarea sistemelor: Ce sistem să se producă?

versus

Proiectarea și implementarea sistemelor: Cum se produce sistemul respectiv?

Modelarea folosește limbaje expresive pentru a descrie funcționalitățile dorite ale sistemului în cauză.

Validarea versus Verificarea sistemelor

Validarea sistemelor: Producem produsul corect?

versus

Verificarea sistemelor: Producem corect produsul?

Verificarea sistemelor controlează dacă sistemul îndeplinește corect funcționalitățile specificate.

Analiza versus Modelarea sistemelor

Analiza

- reprezintă domeniile problemă din mai multe perspective
- descoperă caracteristcile sistemului

Modelarea sistemelor

- sistemul este descris în întregime și neambiguu
- folosește un limbaj foarte expresiv, adaptat domeniului problemelor avute în vedere
- se concentreaza pe aspecte funcționale sau non-funcționale

Specificarea formală

"Exprimarea într-un limbaj formal și la un anumit nivel de abstractizare, a unei colecții de proprietăți pe care un sistem ar trebui să le îndeplinească".

Specificarea formală

Caracteristicile unei specificări formale:

- **Adecvată** - să descrie într-un mod potrivit problema la care se referă
- **Consistentă** - interpretarea sa din punct de vedere semantic să fie plină de înțeles
- **Nu trebuie să fie ambiguă** - nu trebuie să aibă mai multe interpretări, toate adevărate
- **Minimală** - să se refere numai la proprietățile relevante ale problemei

Specificarea formală

- Rezultatele verificării formale depind de calitatea specificărilor formale: **modelul verificat va fi la fel de bun ca și specificarea formală care stă la baza sa**
- Orice greșeală sau inadvertență de la acest nivel se va propaga și asupra verificării formale propriu-zise.
- Specificarea formală ca și modelul de verificat trebuie deci ele însele validate.

Specificarea formală: proprietăți ale sistemelor

- **Safety**: sistemul nu ajunge niciodată într-o stare nepotrivită
  - o anumită proprietate este valabilă de-a lungul întregii execuții
  ex: deadlock freedom, mutual exclusion
- **Liveness**: sistemul progresează în sarcina pe care o îndeplinește
  - unele acțiuni au loc infinit de des
- **Inevitability**: în cele din urmă, un anumit lucru va avea loc
- **Response**: ori de câte ori are loc X, în cele din urmă va avea loc și Y
  ex: sistemul răspunde la fiecare mesaj primit, fiecare cerere este rezolvată la un moment dat
- **Fairness assumptions**: X are loc, presupunând că procesorul este partajat de către toate procesele

De ce să se utilizeze metodele formale?

În prezent calculatoarele au devenit o prezență ubicuuă: calculatoare presonale, telefoane inteligente, tablete, ceasuri, mașini, trenuri, avioane, ș.a.

Sistemele software devin din ce în ce mai complexe.

- producerea de software lipsit de bug-uri este din ce în ce mai dificilă
- efectele negative ale bug-urilor pot fi enorme: financiar, reputație, vieți omenești
- cu cât o eroare este descoperită mai târziu, cu atât costul eliminării ei este mai mare

(In)famous bugs

European Space Agency's Ariane 5 Flight 501 (US$1 billion) - destroyed 40 seconds after takeoff (June 4, 1996) - self-destructed due to a **bug in the on-board guidance software**

Smart ship USS Yorktown - dead in the water for nearly 3 hours - **divide by zero** error (1997)

OpenSSL vulnerability (introduced 2012, disclosed April 2014) - **removed confidentiality** from affected services

- `http://www.computerworld.com/article/2515483/enterprise-applications/epic-failures--11-infamous-software-bugs.html`
- `https://en.wikipedia.org/wiki/List_of_software_bugs`

(In)famous bugs

Debugging Day - September 9

- Descoperire bug-uri după ce sistemul este deja distribuit la clienți (în urma efectelor produse)
- Corectare bug-uri în codul sursă
- Redistribuire sistem sau distribuire patch-uri

versus

- descoperire erori de proiectare în modelul sistemului (verificare formală)
- implementare corectă distribuită la clienți

Aplicabilitatea metodelor formale

Nu pot fi aplicate pentru a analiza automat toate proprietățile tuturor programelor scrise

- programele sunt în general mult **prea mari**
- problemele respective ar putea fi **indecidabile** în principiu
  The **halting problem** is undecidable: there is no computer program that can correctly determine, given any program P as input, whether P eventually halts when run with a particular given input.
  (`https://en.wikipedia.org/wiki/Halting_problem`)

Metodele formale se folosesc pentru:

- părțile **critice** ale sistemelor
- subclase decidabile de probleme sau proprietăți

Aplicabilitatea metodelor formale

Comunicații

- verificarea și testarea protocoalelor de comunicații (inclusiv proprietăți de securitate)
- evaluarea performanțelor (throughput, queuing time, energy consumption, etc.)

Sisteme critice din punct de vedere al siguranței

- sistemele de comandă și control ale aeronavelor
- sistemele de control ale traficului aerian
- sistemele de comandă ale traficului feroviar

Proiectarea componentelor hardware

- procesoare, memorii cahe, interfețe periferice, magistrale

Metode de verificare formală

- Verificarea modelelor - **Model checking** (verificatoare de modele)
  - Exemplificată în slide-urile anterioare
- Demonstrarea teoremelor - **Theorem proving** (demonstratoare de teoreme)
- Verificarea echivalenței - **Equivalence checking** (verificatoare de echivalență)

Verificarea modelelor

Construirea și explorarea sistematică exhaustivă a spațiului stărilor modelului matematic al sistemului

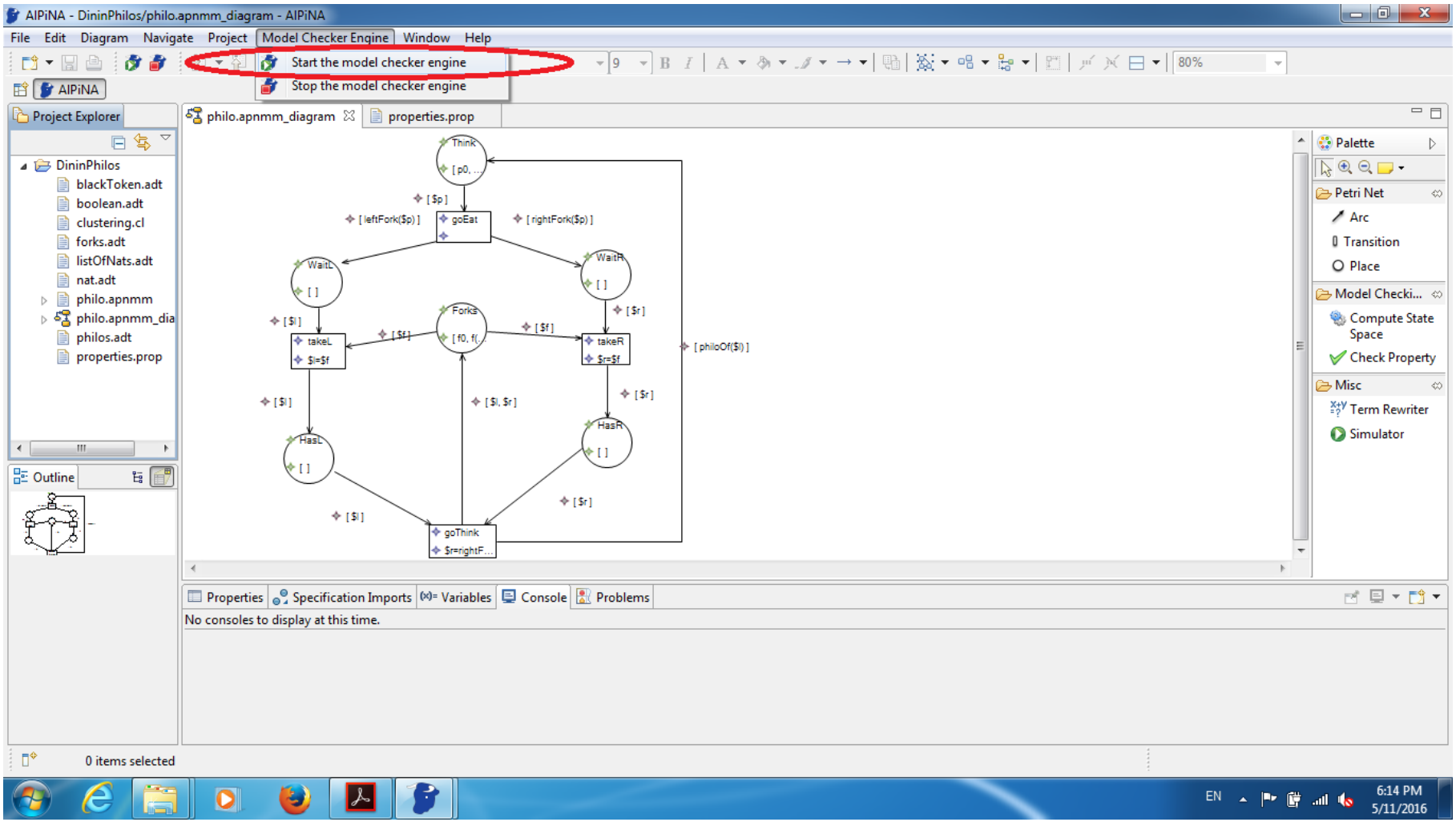
1. Construirea unui model al sistemului de verificat - limbaj de descriere a sistemului
   ex: rețele Petri algebrice pentru prânzul filosofilor
2. Specificarea proprietăților sistemului care trebuie verificate - limbaj de descriere a proprietăților
3. Verificarea propriu-zisă

Verificarea modelelor

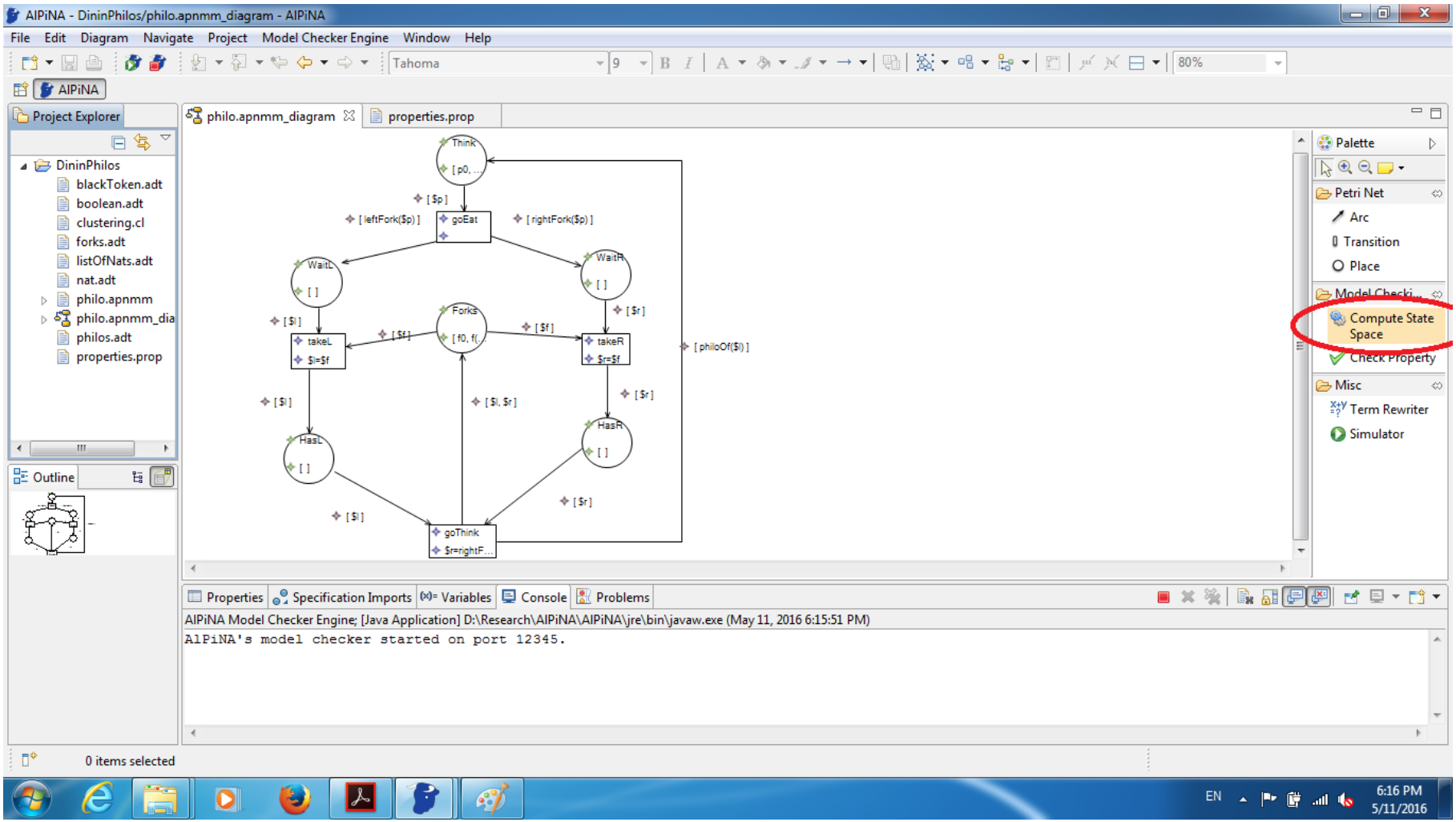
Instrumente special create

- Intrare:
  - modelul sistemului
  - proprietățile care trebuie verificate
- Ieșire
  - modelul are proprietatea/proprietățile specificate
  - modelul NU are proprietatea/proprietățile specificate
    - **contraexemplu** - configurație a sistemului în care proprietate nu este îndeplinită

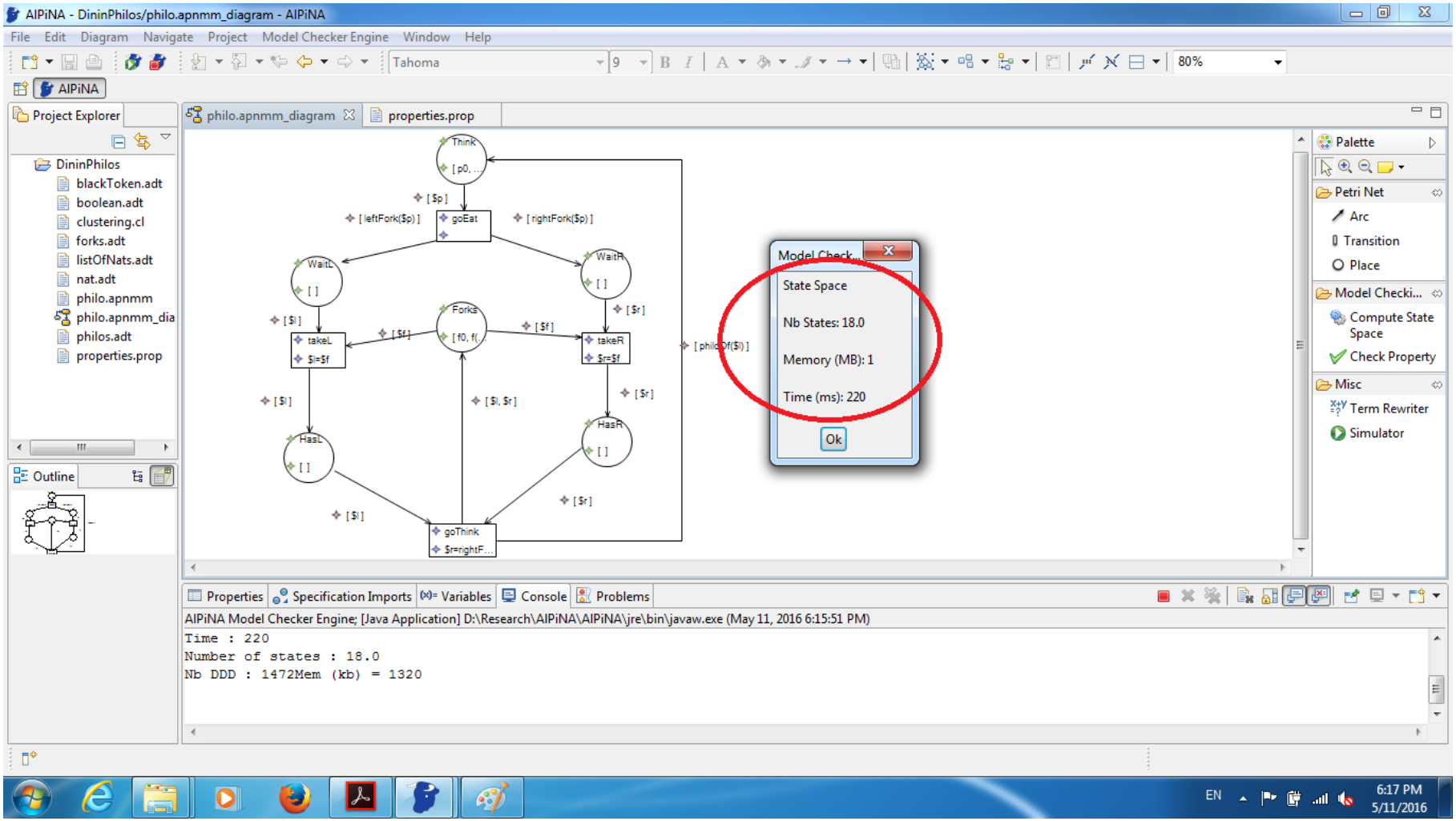
Verificarea modelelor - AlPiNA - Prânzul filosofilor



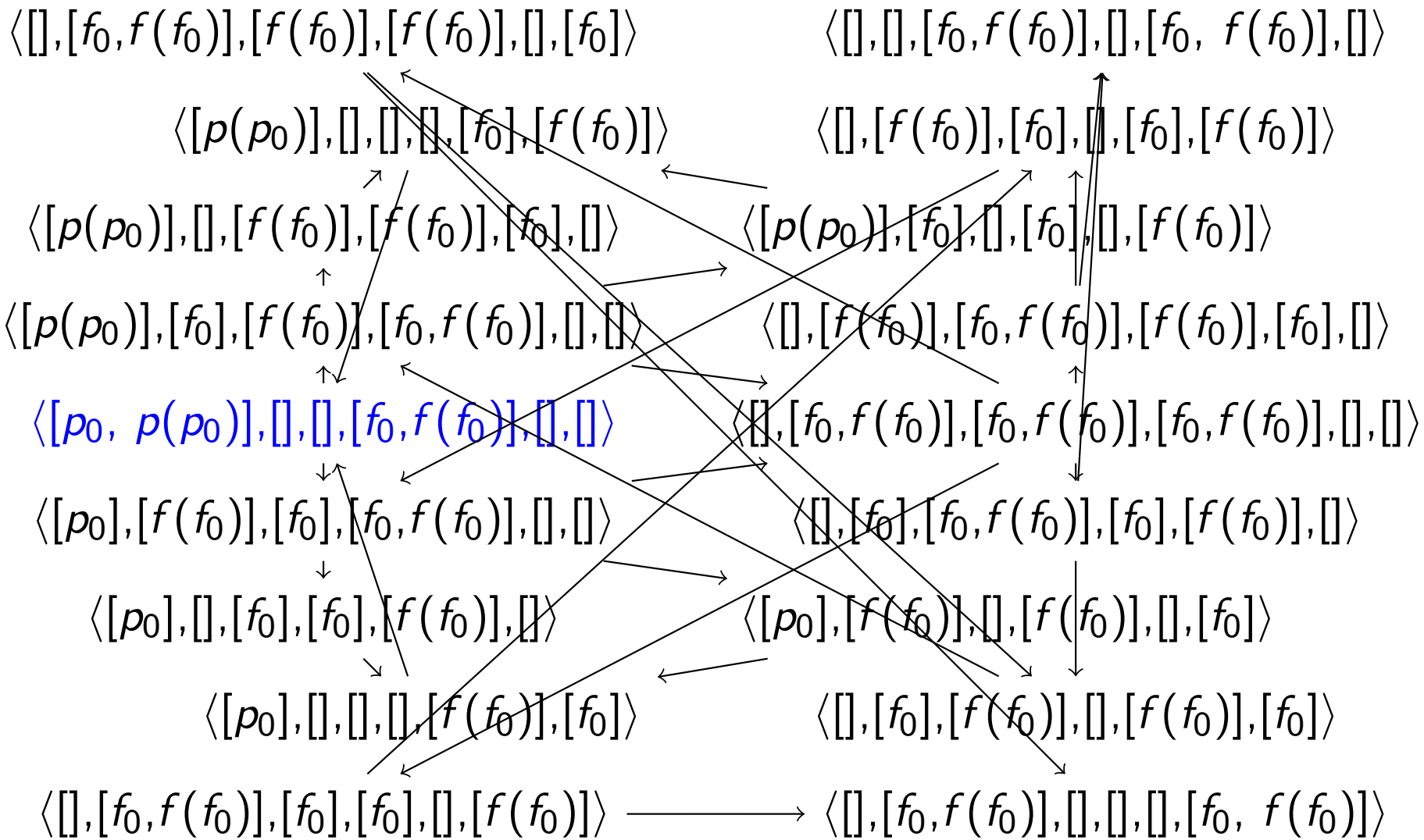
Verificarea modelelor - AlPiNA - Prânzul filosofilor



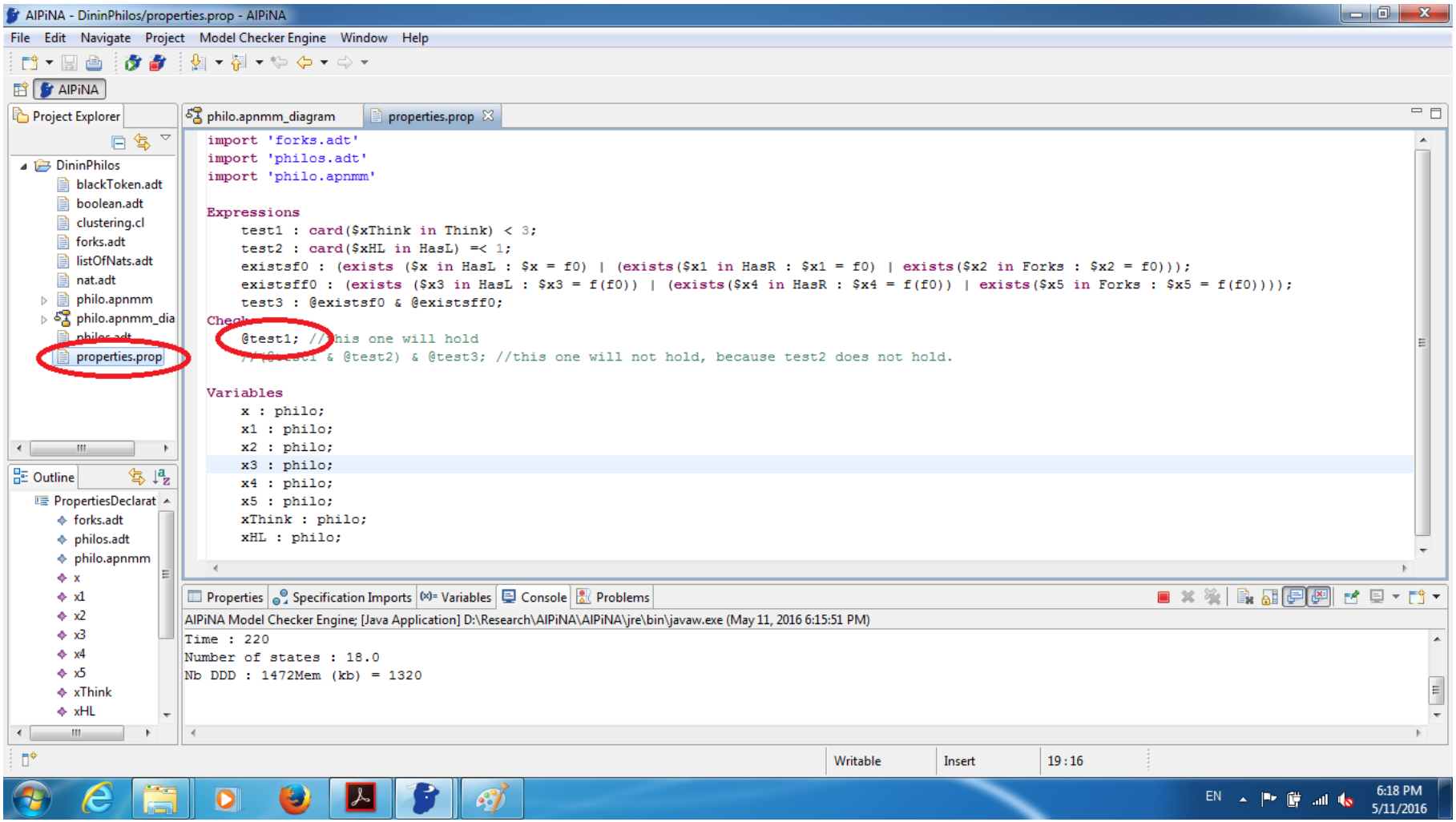
Verificarea modelelor - AIPiNA - Prânzul filosofilor



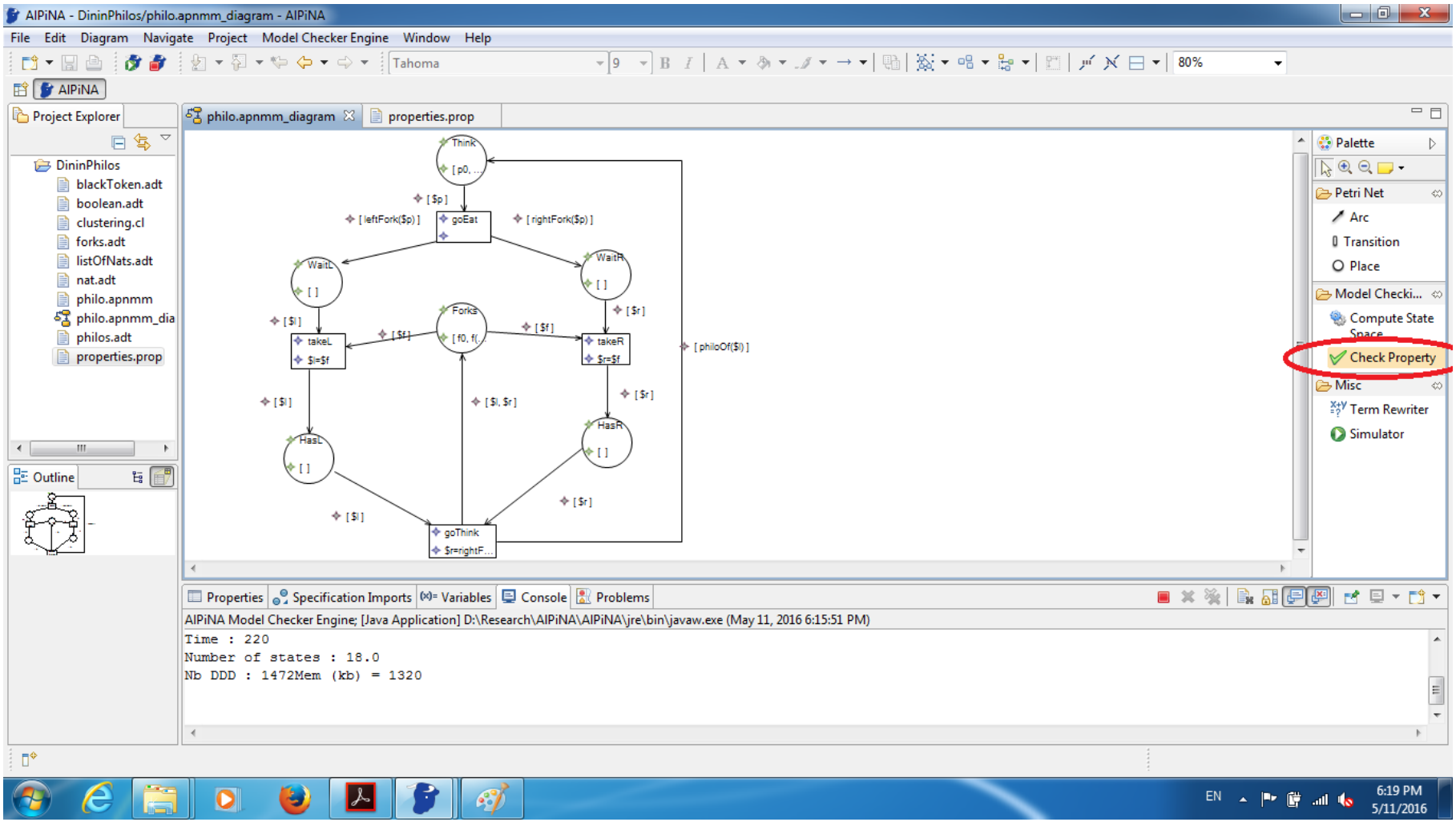
Verificarea modelelor - AIPiNA - Prânzul filosofilor



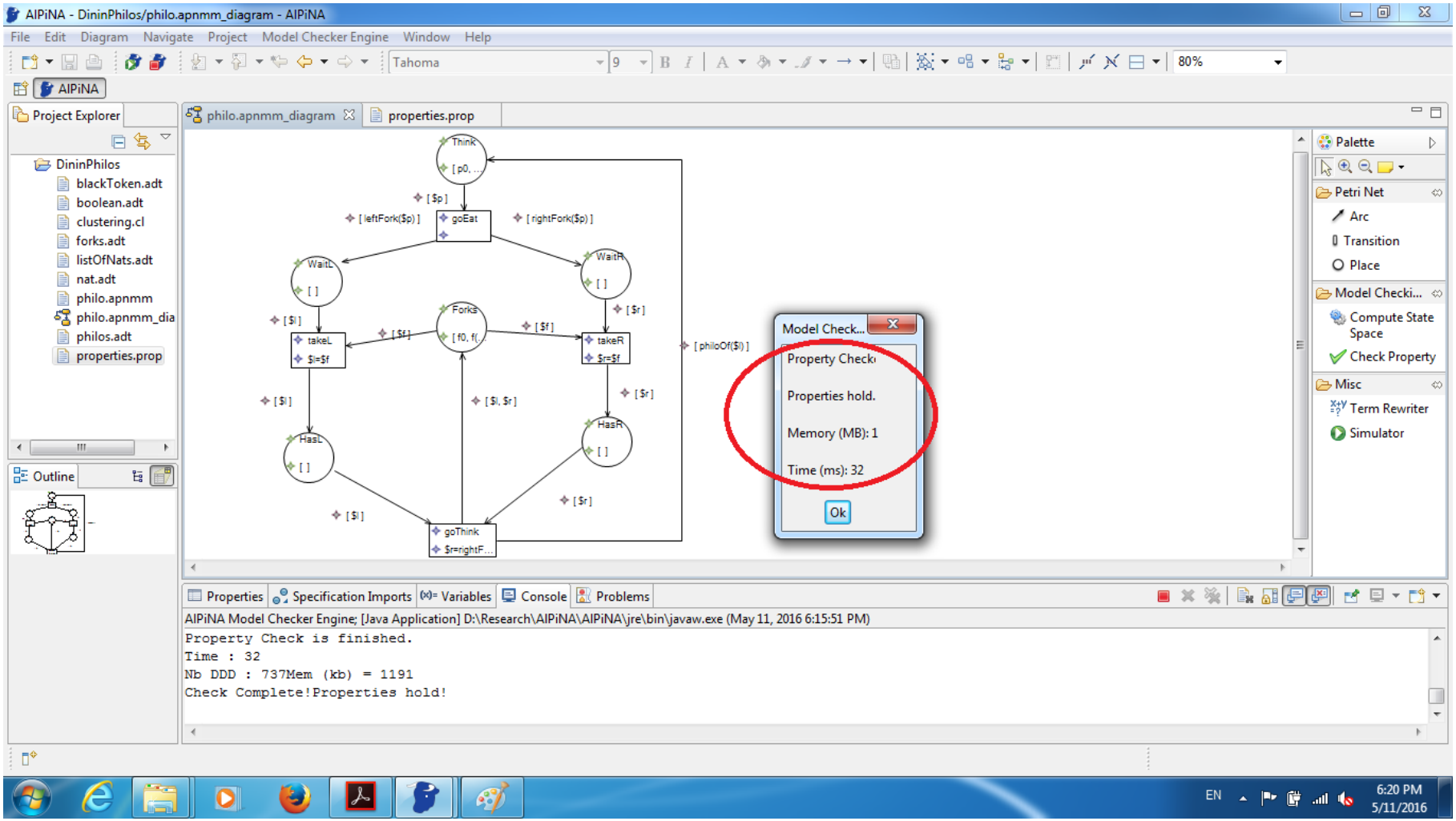
Verificarea modelelor - AIPiNA - Prânzul filosofilor



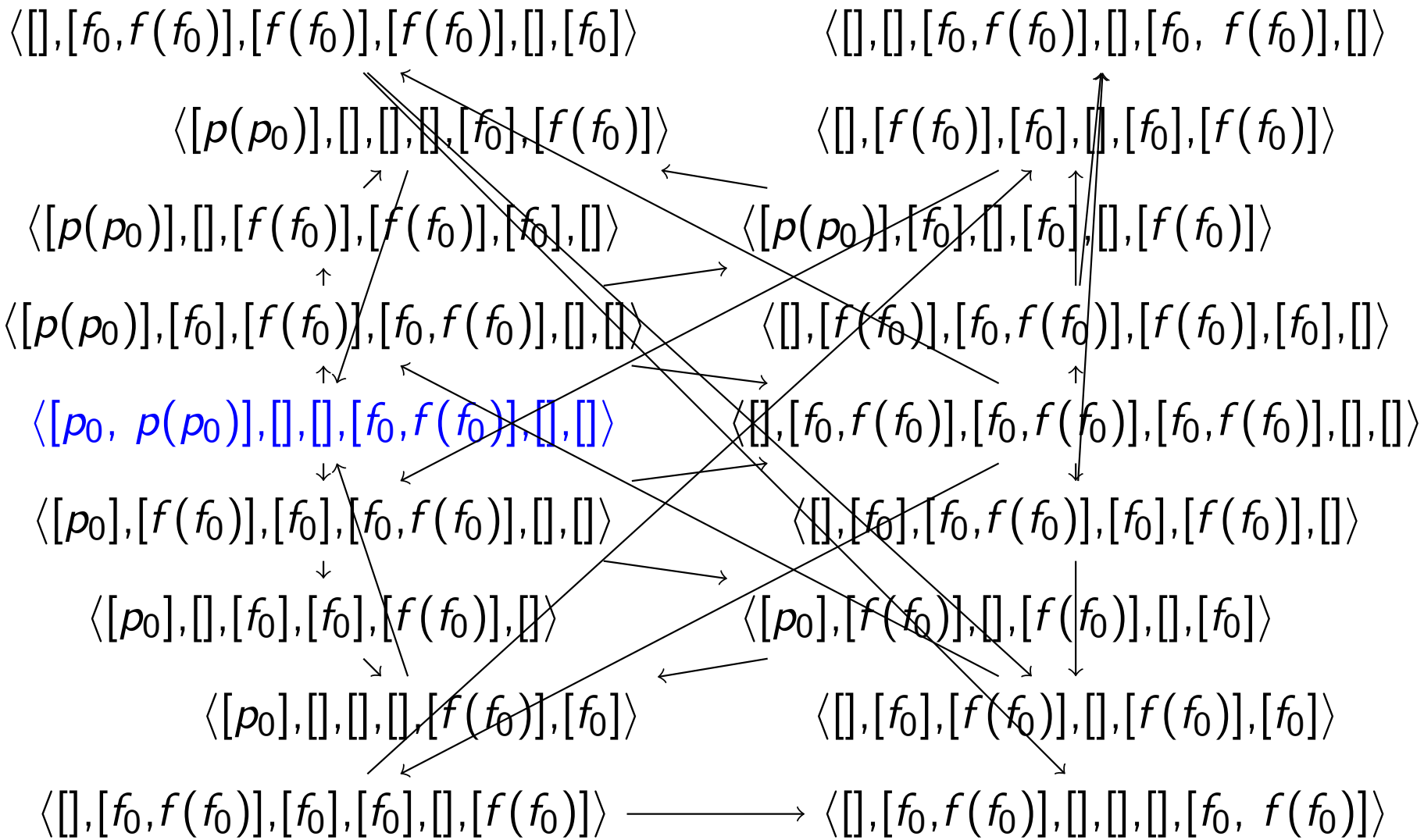
Verificarea modelelor - AlPiNA - Prânzul filosofilor



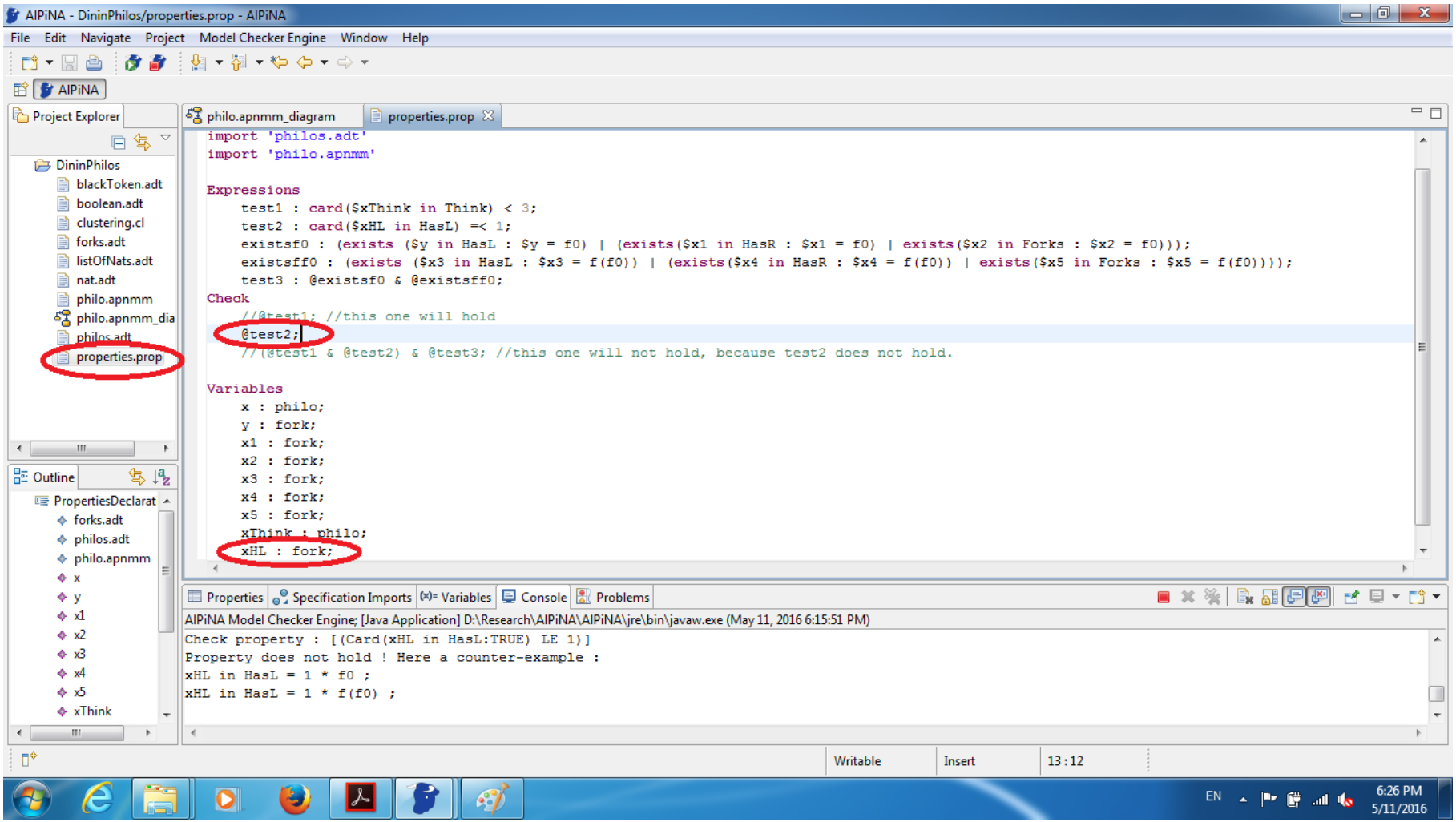
Verificarea modelelor - AIPiNA - Prânzul filosofilor



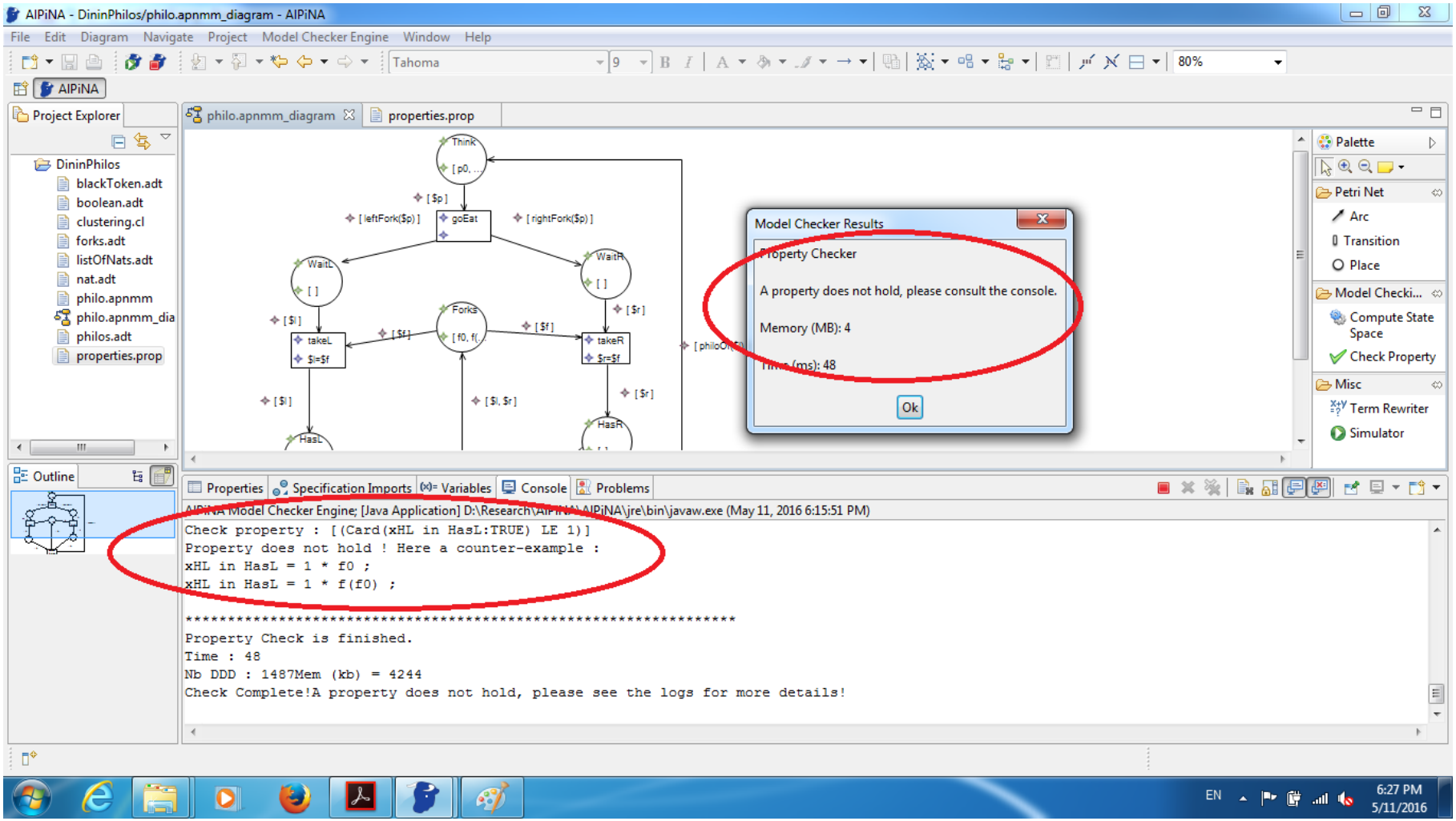
Verificarea modelelor - AIPiNA - Prânzul filosofilor



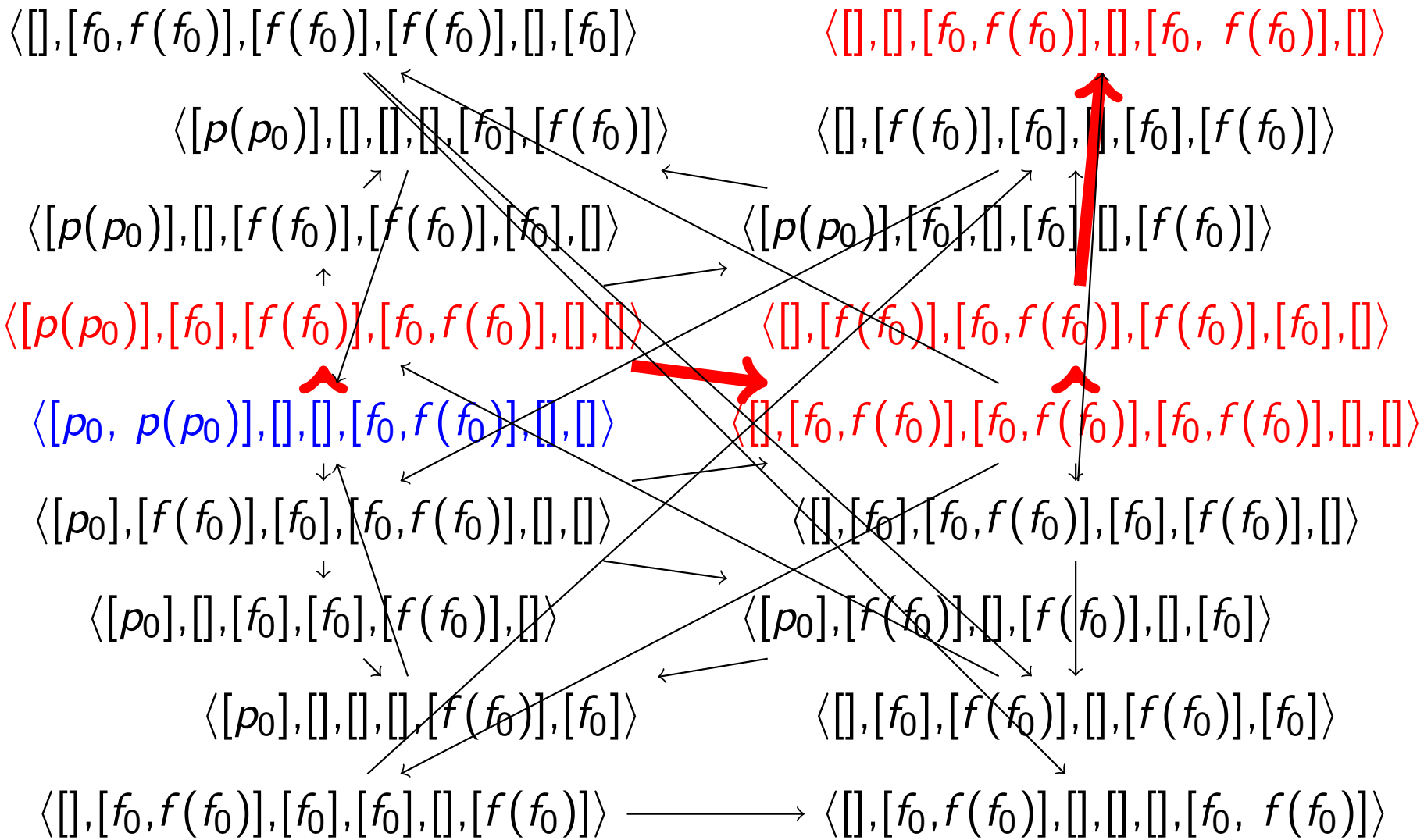
Verificarea modelelor - AIPiNA - Prânzul filosofilor



Verificarea modelelor - AlPiNA - Prânzul filosofilor



Verificarea modelelor - AIPiNA - Prânzul filosofilor



Verificarea modelelor - AIPiNA - Prânzul filosofilor

Exerciții:

- Doi filosofi vecini pot mânca în același timp?
  !exists($f1 in HasL, $f2 in HasR : $f1 = $f2);
- Doi filosofi aflați față în față pot mânca în același timp?
  ...
- Se poate întâmpla ca filosofii să flămânzească până la moarte?
  Deadlock;
- Un anume filosof poate mânca în cele din urmă (presupunând că dorește să facă acest lucru)?
  forall($x in Think : $x != p0) & exists($f1 in HasL : $f1 = f0) & exists($f2 in HasR : $f2 = f(f0));

Verificarea modelelor - AIPiNA - Prânzul filosofilor

Number of philosophers	Number of states
2	18
3	76
4	322
5	1364
6	5778
7	24476
8	103682
...	...
200	2.5×10^{125}

Table 1: Evoluția dimensiunii spațiului stărilor în funcție de numărul de filosofi

Problema exploziei spațiului stărilor

Explozia spațiului stărilor constă în existența unui număr foarte mare de stări în care sistemul poate să ajungă, chiar și pentru modele de dimensiuni mici

Cauzele exploziei spațiului stărilor pot fi:

- concurența
- variabilele care pot lua o gamă largă de valori (în cazul lmbajelor de modelare care permit utilizarea variabilelor)
- numărul mare de variabile de care depinde modelul

Contramăsuri la problema exploziei spațiului stărilor

- **Abstractizarea** - eliminarea informațiilor irelevante și simplificarea modelului
- **Compresia** - operațiile se fac asupra unor **mulțimi de stări**, pentru codarea cărora se folosesc structuri eficiente (BDD: Binary Decision Diagrams) - Symbolic Model Checking
- **Reducerea** - evitarea explorării multiple a execuțiilor echivalente, reducându-se astfel numărul de combinații considerate
  ex: dacă pentru verificarea în curs nu contează care dintre procesele P și Q se execută primul, nu este nevoie să fie analizată și varianta PQ, și varianta QP, fiind suficientă analiza uneia dintre ele

Perspective multiple asupra aceluiași sistem

Există multiple pentru descrierea sistemelor:

- rețele Petri (foarte multe tipuri)
- unele limbaje de programare (ex: PROMELA)
- communicating automata
- process algebra
- limbaje descriptive semi-formale (ex: UML)

Alegerea se face în funcție de:

- tipul sistemului modelat
- tipul proprietăților care vor fi verificate
- precizia urmărită

Verificarea modelelor

Verificatoare de modele:

- exemplele anterioare pentru rețele Petri
- **FDR4** - CSP (Communicating Sequential Processes)
  `https://www.cs.ox.ac.uk/projects/fdr/`
- **LTSmin** - language independent (process algebra, timed automata, DiVinE, PROMELA, etc.)
  `http://fmt.cs.utwente.nl/tools/ltsmin/`

Verificarea modelelor

Verificatoare de modele:

- **SPIN** - PROMELA
  `http://spinroot.com/spin/whatispin.html`
- **UPPAAL** - networks of timed automata extended with data types
  `http://www.uppaal.org/`
- **StrataGEM** -
  `https://github.com/mundacho/stratagem`
- etc.

Demonstrarea teoremelor

Verificarea valorii de adevăr a teoremelor matematice postulate despre un sistem.

Etape:

1. Stabilirea axiomelor
2. Stabilirea concluziilor
3. Demonstrația propriu-zisă
   - folosește regulile inferenței logice pentru a stabili dacă concluziile sunt adevărate sau false

Constă mai mult dintr-un proces interactiv: necesită **intervenția utilizatorului** pentru a ghida instrumentul în direcția corectă

Mult mai puțin automatizabilă decât metoda verificării modelelor

Demonstrarea teoremelor - Un exemplu foarte simplu

```
theorem and_commutative (p q : Prop) : p ∧ q → q ∧ p :=
assume Hpq : p ∧ q,
have Hp : p, from and.elim_left Hpq,
have Hq : q, from and.elim_right Hpq,
show q ∧ p, from and.intro Hq Hp
```

Din "Theorem proving in Lean" -
https://leanprover.github.io/tutorial/

Demonstrarea teoremelor

Demonstratoare automate de teoreme:

- **ACL2** - `http://www.cs.utexas.edu/users/moore/acl2/`: high order logic
- **Coq** - `https://coq.inria.fr/`: high order logic
- **E** - `http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html`: first order logic
- **HOL** - `http://www.cl.cam.ac.uk/research/hvg/HOL/`: high order logic
- **Prover9** - `http://www.cs.unm.edu/~mccune/prover9/`: 1st order logic
- etc.

Verificarea echivalenței

Procesul verificării dacă două **implementări ale aceluiași sistem**, la nivele diferite de abstractizare, **sunt identice**

- Nu poate fi folosită pentru a verifica dacă un sistem are sau nu erori
- Ci numai pentru a semnala dacă între două implementări există sau nu diferențe funcționale

Se utilizează pe larg în industrie
ex: proiectarea circuitelor integrate digitale

Verificarea echivalenței

Verificatoare de echivalență:

- **CADP** - acceptă o gamă largă de formalisme de intrare
  `http://cadp.inria.fr/`
- **SYNOPSYS** - circuite hardware
  `http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/default.aspx`
- **Xilinx** - FPGA
  `http://www.xilinx.com/`
- etc.

Verificarea formală a protocoalelor de securitate

1. Modelarea a componentelor sistemului
   1.1 rețeaua
   1.2 adversarul
   1.3 protocolul de securitate
2. Specificarea proprietăților de securitate care trebuie verificate
   - autentificarea
   - confidențialitatea
   - integritatea
   - non-repudierea
   - disponibilitatea
3. Verificarea propriu-zisă

Modelul rețelei

Este furnizat de către instrumentul de verificare formală

Rețeaua - colecția tuturor **nodurilor oneste, plus adversarul**
Nodurile:

- statice
- identificate printr-un nume unic
- o singură interfață de comunicație
- adversarul are mai mult de o interfață - este modelat sub forma mai multor noduri
- legături simetrice: dacă un nod oarecare A poate să primească o transmisie de la un nod B, atunci și nodul B poate să primească o transmisie de la nodul A

Modelul adversarului

Este furnizat de către instrumentul de verificare formală

Adversar - nod care are ca și scop **devierea** și **alterarea în mod activ** a rulării protocolului de securitate

Acțiunile adversarului depind de protocolul de securitate țintă

Adversarul **apare ca fiind un utilizator legitim** al rețelei, putând iniția o conversație cu orice alt nod din rețea

Modelul adversarului - Capabilități

Modelul Dolev-Yao (modelul adversarului puternic)

Adversarul poate:

- genera orice tip de mesaj din protocolul țintă
- intercepta mesajele oricărui nod din rețea
- răspunde la orice tip de mesaj primit/interceptat
- cripta/decripta mesaje folosind cheile pe care le cunoaște
- modifica orice mesaj primit/interceptat

Adversarul este rețeaua, și rețeaua este adversarul

Modelul adversarului - restricții

Singura restricție - puterea finită de calcul

- Nu este în măsură să lanseze atacuri cripto analitice cu scopul de a compromite chei simetrice sau asimetrice
- Nu poate inversa funcțiile folosite pentru hash

Modelul protocolului de securitate

Trebuie realizat de către persoana care face verificarea

Depinde de limbajul de modelare acceptat de către instrumentul de verificare formală

Utilizatorul specifică apoi proprietățile de securitate pe care dorește să le verifice

Verificarea propriu-zisă

Depinde de instrumentul de verificare formală utilizat

Intrări:

- modelul protocolului de securitate
- proprietățile de securitate care trebuie verificate

Ieșire:

- proprietățile de securitate sunt îndeplinite
- proprietățile de securitate NU sunt îndeplinite
  - contraexemplu - **un exemplu de atac**

Verificarea formală a protocoalelor de securitate

Verificatoare de modele specializate pe probleme de securitate:

- **AVISPA**
  `http://www.avispa-project.org/`

- **AVANTSSAR**
  `http://www.avantssar.eu/`

- **Casper & FDR2**
  `http://www.cs.ox.ac.uk/gavin.lowe/Security/Casper/`
  (latest version **FDR4** - `https://www.cs.ox.ac.uk/projects/fdr/`)

Verificarea formală a protocoalelor de securitate

Verificatoare de modele specializate pe probleme de securitate:

- **SPaCIoS**
  `http://www.spacios.eu/`
- **ProVerif**
  `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`
- **Scyther**
  `https://www.cs.ox.ac.uk/people/cas.cremers/scyther/`
- etc.

Bibliografie

- John Franco, **Formal Methods**, University of Cincinnati
  `http://gauss.ececs.uc.edu/Courses/c626/lectures.html`

- Vasileios Koutavas, **Formal Verification**, University of Dublin
  `https://www.scss.tcd.ie/Vasileios.Koutavas/teaching/cs4004-4504/mt1819/`

- Radek Pelanek, **Formal Verification and Model Checking**, Masaryk University
  `https://www.fi.muni.cz/~xpelanek/IA158/slides/verification.pdf`