Secure Compiler Seminar (2006/12/12)

米澤研 M2 佐藤秀明

今回の内容

- JavaScript Instrumentation for Browser Security.
  - Dachuan Yu, Ajay Chander, Nayeem Islam and Igor Serikov.
  - POPL 2007.
- JavaScript の安全な実行
  - チェックコードを挿入
  - 動的コード生成にも対応
  - Security Policy は自由に設定可能

JavaScript の潜在的危険性

- XSS: cross-site scripting
  - サーバ側の想定していないコードをページ内に埋め込む
  - 他ドメインからは秘密だったはずの cookie 情報が流出
- phishing
  - ロケーションバーの隠匿
  - ロケーションバーの詐称
- 煩い window pop-up

危険なコードを実行する前に検査したい

JavaScript 解析の難しさ

- 動的コード生成の存在
  - 評価結果が新たなコードになりうる

```
var i = 1;
document.write(
    "<script> i = 2; document.write(i); </scr" + "ipt>" + i);
```

出力：21

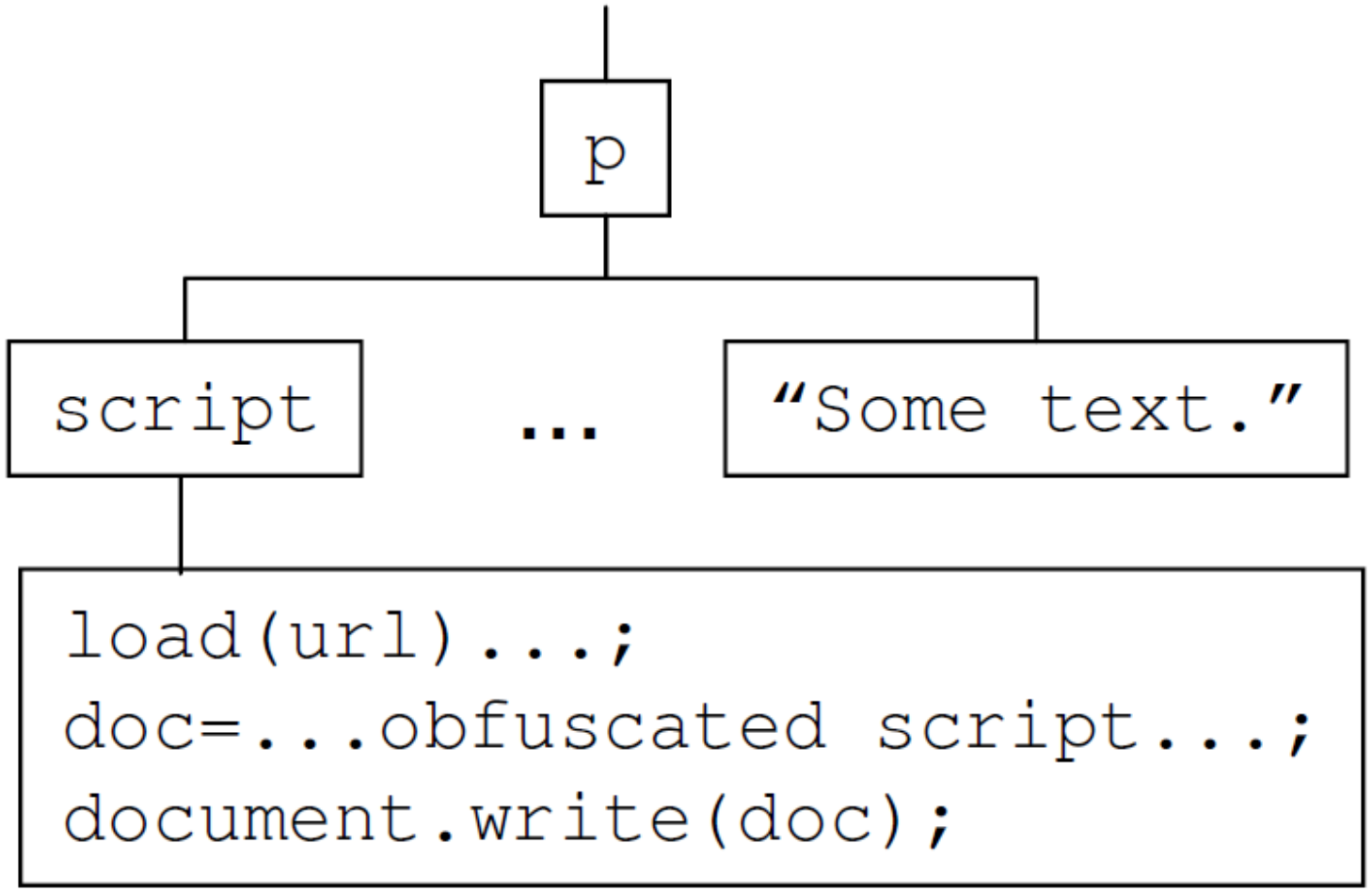
パーザの誤解を
防ぐためのおまじない

```
var i = 1;
document.write(
    "<script> i = 2; document.write(i); </scr" + "ipt>");
document.write(i);
```

出力：22

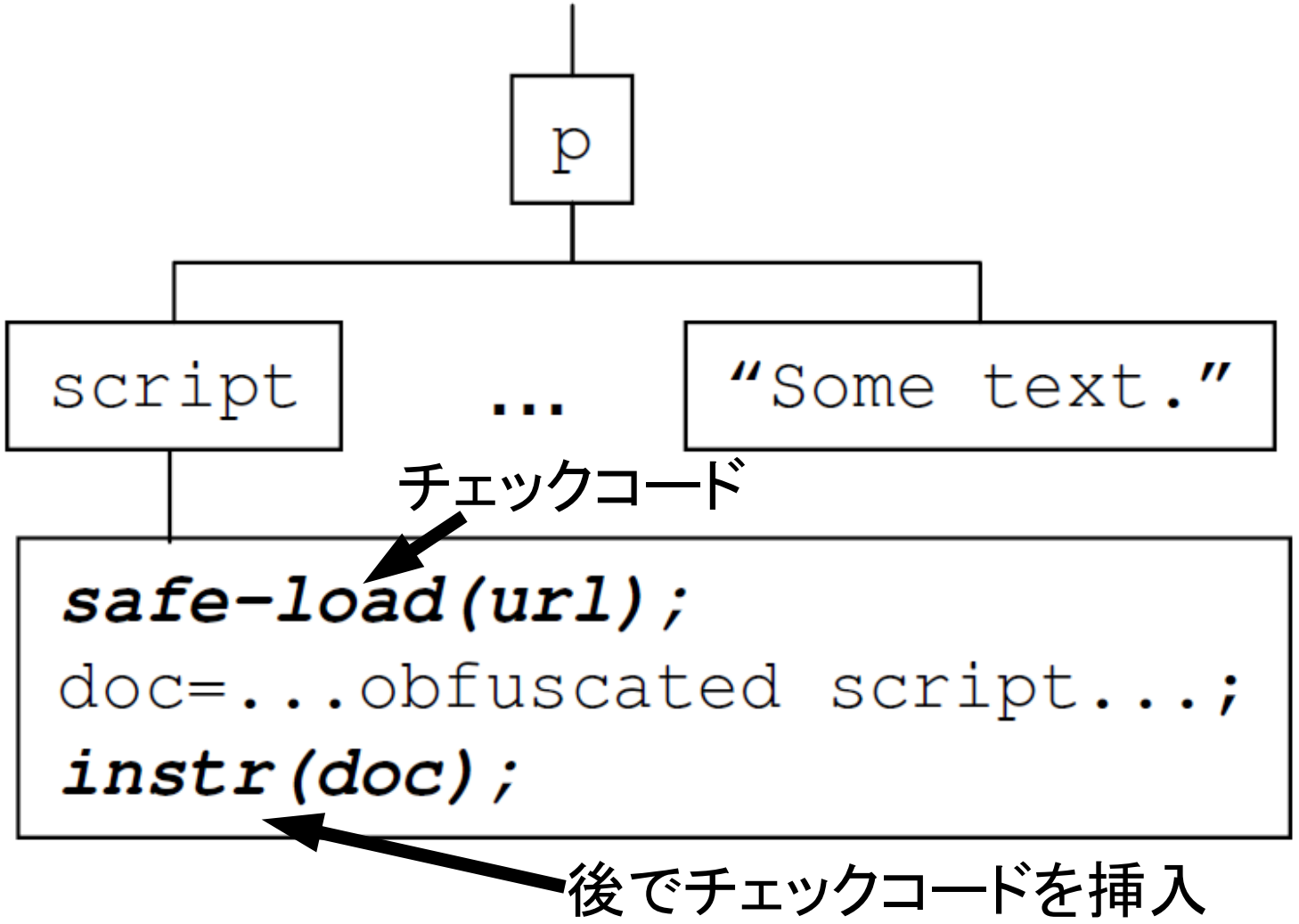
動的なコード変換の導入(1)

- 元のコード



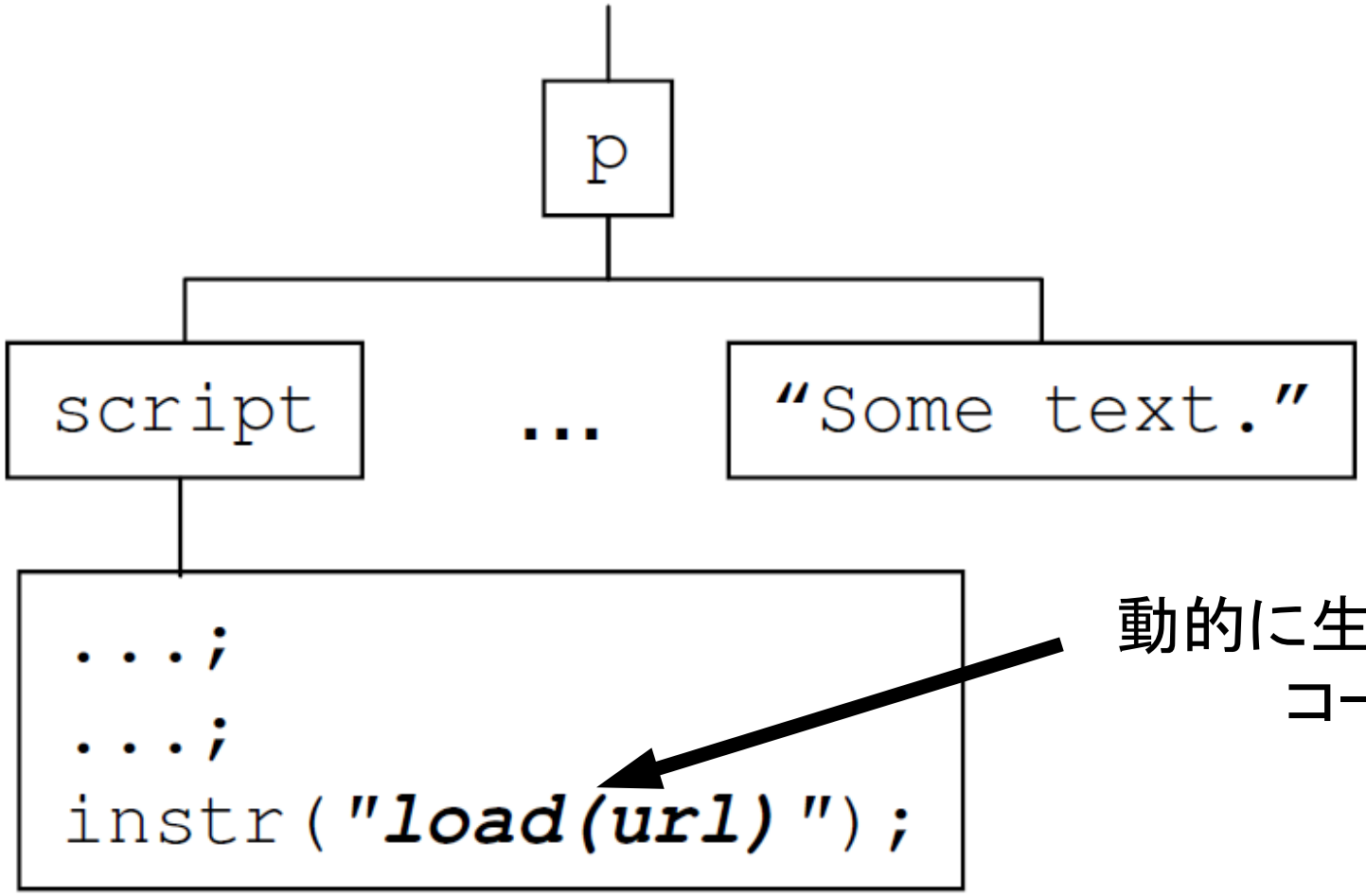
動的なコード変換の導入 (2)

- ロード時のコード変換



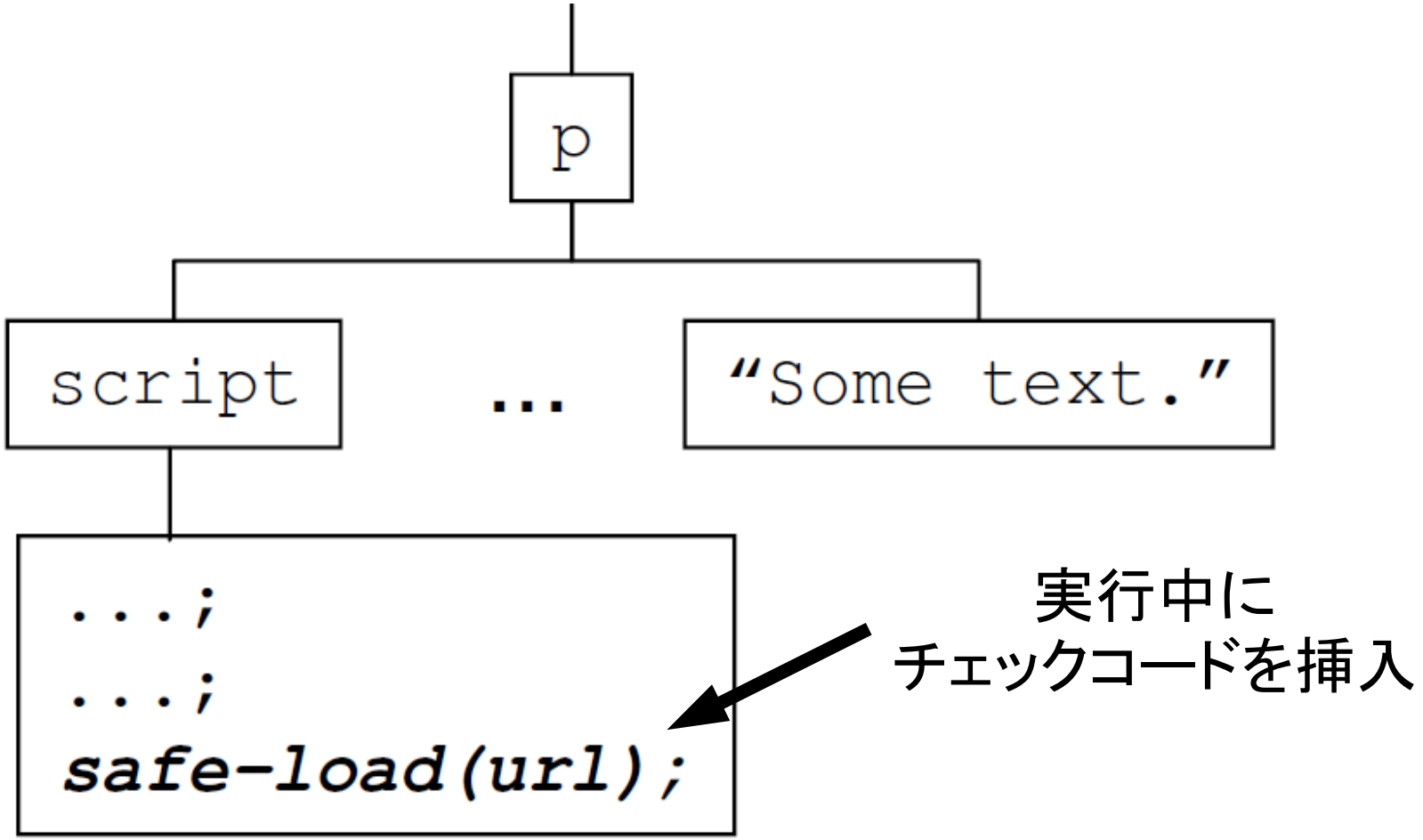
動的なコード変換の導入 (3)

- 実行の途中経過



動的なコード変換の導入 (4)

- 実行時のコード変換



CoreScript の Syntax(1)

- JavaScript のサブセット (Figure 4 を参照)

$$W ::= (\Sigma, \chi, B, C)$$

  - W: CoreScript 実行中の状態
  - Σ: ウェブ全体
  - χ: スクリプト内の変数と関数
  - B: ブラウザ
  - C: Cookie

ハンドラから実体への写像として表現

CoreScript の Syntax(2)

- document の分類
  - D: 普通の document
  - D^v: script を含まない document
- セキュリティに関する action の分類
  - A: 普通の action
  - A^v: 引数に document しかとらない action

CoreScript の Evaluation Rules(1)

- action と expression の評価 (Figure 5 を参照)
  - 変数をその値へ写しているだけ
- global な状態遷移 (Figure 6 を参照)
  - どれか 1 つの window が非決定的に実行される

$$h \vdash W \rightsquigarrow W' : A^v$$

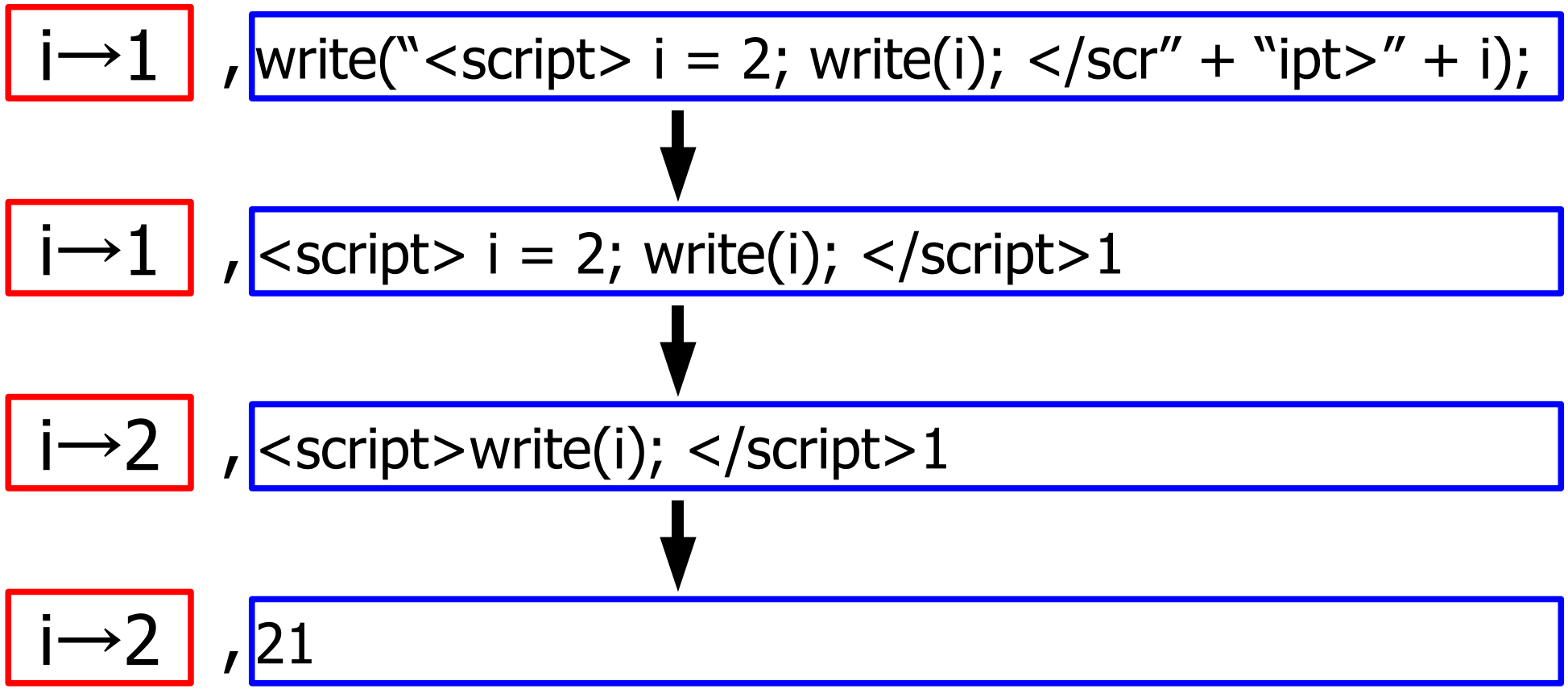
  - h: 実行を進める window のハンドラ
  - W, W': 遷移前後での各状態
  - A^v: 状態遷移中に起きた action 列

CoreScript の Evaluation Rules(2)

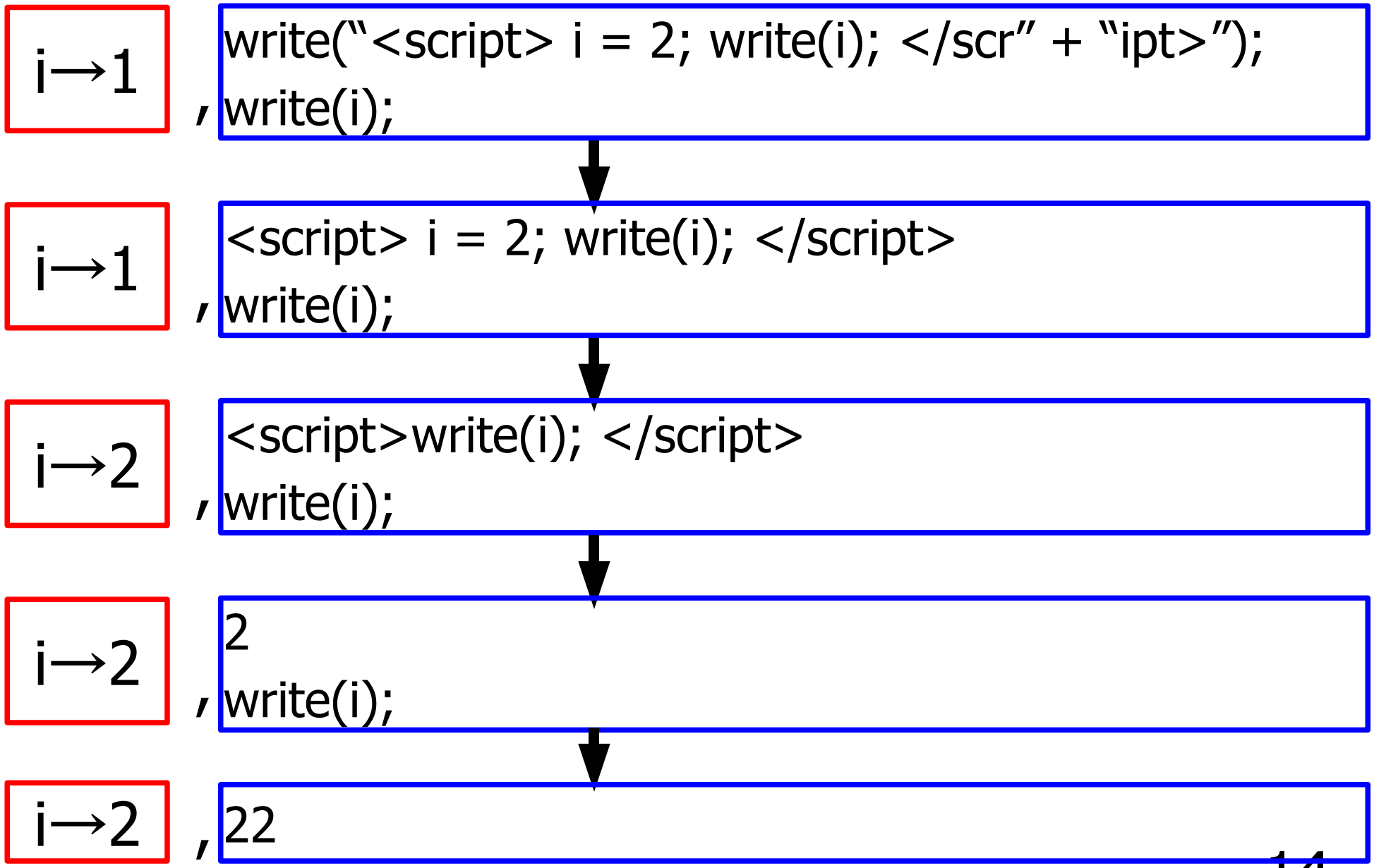
- (14) 式で用いられる補助関数 (Figure 7 を参照)
  - focus(): 次に実行される statement を返す
  - step(): 状態を 1 ステップ進める
    - adv(): 実行が進んだ window の写像情報を更新
    - stepDoc(): script 中の実行された statement を消費

評価順序を厳密に定式化

動的コード生成の評価順序 (1)



動的コード生成の評価順序 (2)



セキュリティポリシーの設定

- ユーザが自由にポリシーを定義
  - 多様化する攻撃に柔軟に対応
- 危険性を検知したらユーザに判断を仰ぐ
  - 直ちに実行を中止するのはよくない
- 内部的には edit automaton でポリシーを表現

Edit Automaton の定義

- action 列を安全なものに置換する automaton

$$\Pi = (Q, q_0, \delta)$$

  - Π: edit automaton
  - Q: 状態の集合
  - q_0: 初期状態
  - δ: 遷移関数

$$\delta(q, A) = (q', A')$$

  - q, q': 遷移前後の各状態
  - A: 入力 action
  - A': 出力 action

Policy Consistency

- Definition 1 (Policy Consistency)

edit automaton $\Pi = (Q, q_0, \delta)$ が consistent

$\Updownarrow$

$\delta(q, A)=(q', A') \Rightarrow \delta(q, A')=(q', A')$ が常に成立

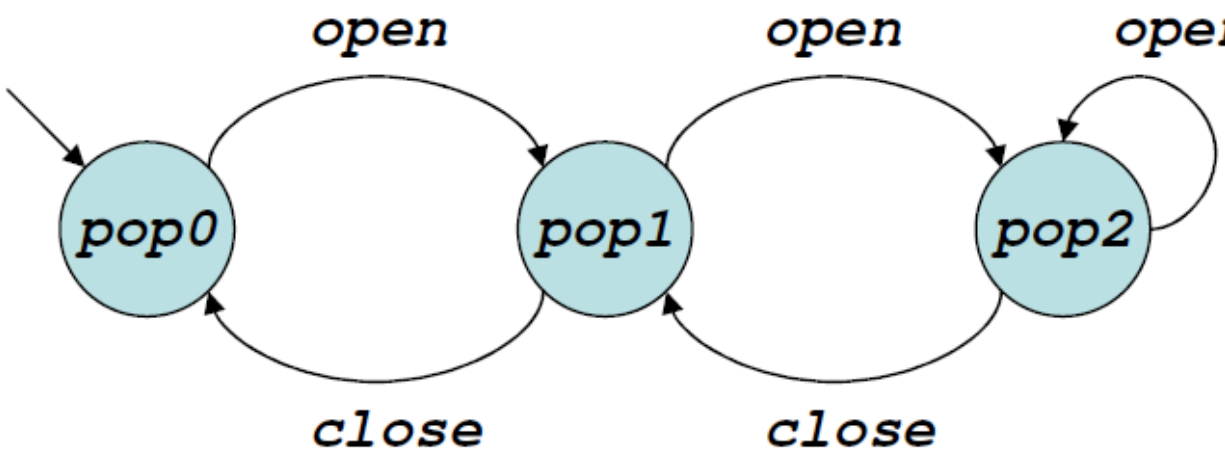
  - 既に安全な action はそれ以上置換されない

- Theorem 1 (Sound Advice)

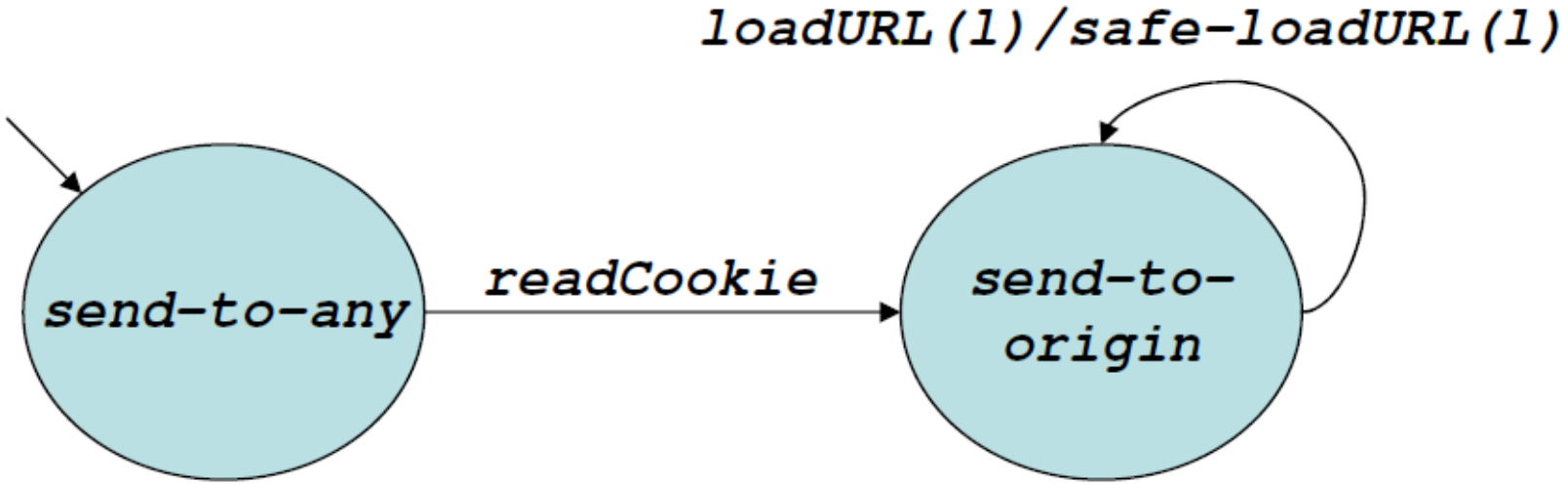
Π を consistent と仮定する。このとき
$\vec{A}$ が $\vec{A'}$ に置換されたならば、
$\vec{A'}$ は置換なしで Π に受理される。

Edit Automaton の例

- 同時に 3 個以上のファイルを open させない



- Cookie 情報の流出をチェック



複数ポリシーの合成

- 複数ポリシーの衝突が問題
  - 多種類の攻撃に同時に対応したい
  - 複数ポリシーを合成後も consistency を保ちたい
- 合成例：一方の置換を他方がスルー

$\Pi_1 = (\{p_i \mid i = 0...n\}, p_0, \delta_1)$,

$\Pi_2 = (\{q_j \mid j = 0...m\}, q_0, \delta_2)$ について

$\Pi_1+\Pi_2 = (\{p_iq_j \mid i = 0...n, j = 0...m\}, p_0q_0, \delta)$

ただし $\delta(p_iq_j, A) =$

$$\begin{cases} (p_lq_k, A') & \text{if } \delta_1(p_i, A) = (p_l, A') \text{ and } \delta_2(q_j, A') = (q_k, A') \\ (p_lq_k, A') & \text{if } \delta_2(q_j, A) = (q_k, A') \text{ and } \delta_1(p_i, A') = (p_l, A') \\ (p_lq_k, \varepsilon) & \text{otherwise} \end{cases}$$

プログラムの実行

- チェックコード挿入 (Figure 12 を参照)
  - check(): 安全性をチェックする関数
  - instr(): 実行中にチェック関数の挿入処理を行う
- policy つきのプログラム実行 (Figure 13 を参照)
  - check() の中のみ edit automaton で action 置換

Correctness

- Theorem 2 (Soundness)
  - チェックコードの挿入処理を施されたプログラムは、ユーザの設定したポリシーをみたす。
- Theorem 3, 4 (Transparency)
  - チェックコード挿入処理とpolicy つきプログラム実行は、もともとポリシーをみたすプログラムの振る舞いに影響を与えない。

現実世界への適用(1)

- 一見 syntax error なケース

```
<script>
document.write("<scr");
document.write("ipt> malic");
var i = 1;
document.write("ious code; </sc");
document.write("ript>");
</script>

<script>
document.write("<scr");</script>ipt>
malicious code
</script>
```

  - あからさまに怪しいから reject すべきでは？

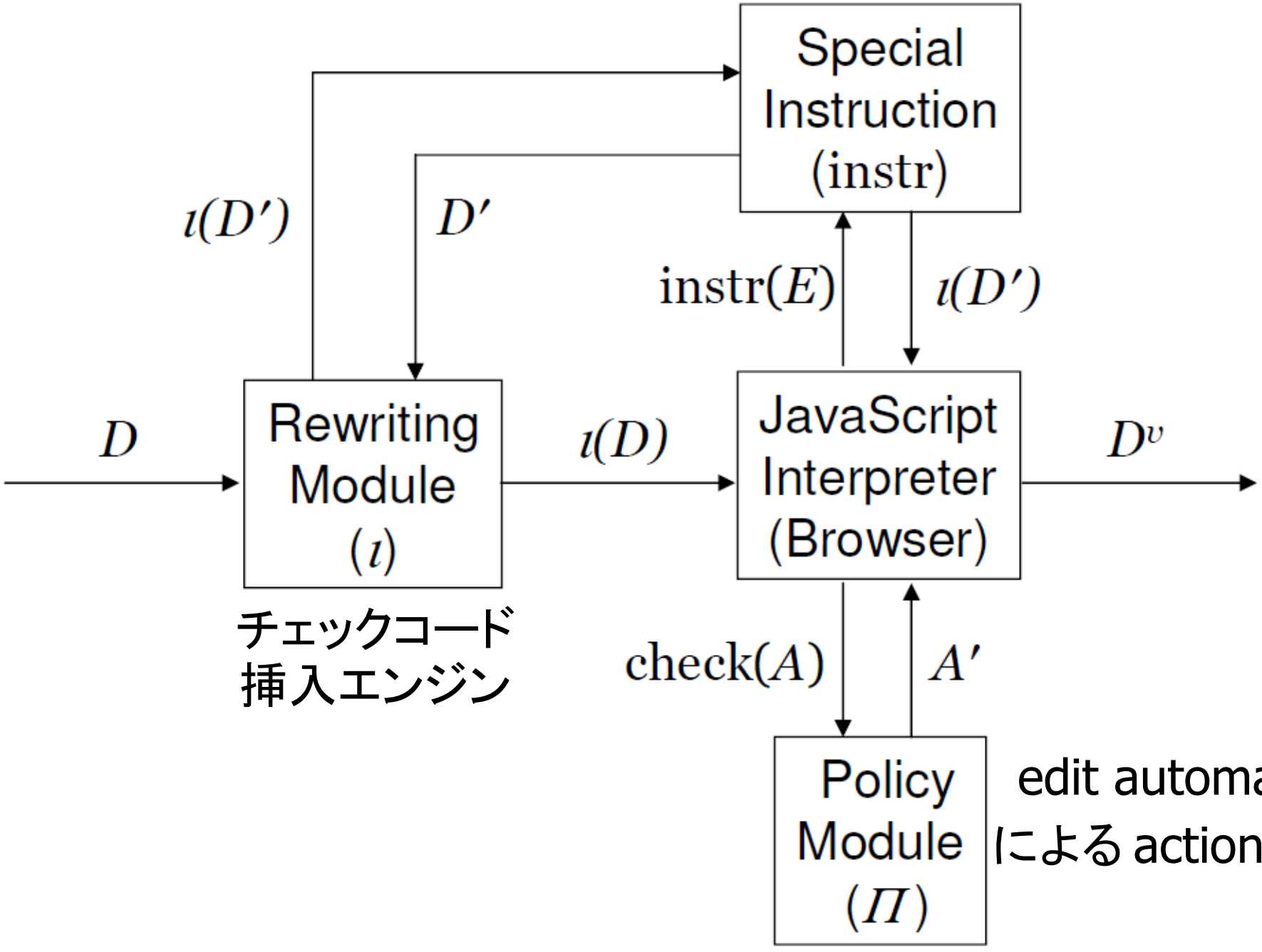
現実世界への適用 (2)

- DOM の任意のノード位置にコード生成されたら？
  - 生成された場所を見つけて instr() しておけばよい
- たくさんの action を扱えるようにする必要性

実装

- コンセプト確認のため簡単な方法をとった
  - プロキシでチェックコード挿入
  - check() と instr() は JavaScript の関数として実装
  - instr() は XMLHttpRequest で非同期に通信
  - edit automaton も javaScript で実装
- 本来は JavaScript のインタプリタに組み込むべき
  - 我々の実装が上書きされる危険性

実装概念図


Special
Instruction
(instr)
$\iota(D')$
D'
instr(E)
$\iota(D')$
D
Rewriting
Module
(ι)
$\iota(D)$
JavaScript
Interpreter
(Browser)
D^v
チェックコード
挿入エンジン
check(A)
A'
Policy
Module
(Π)
edit automaton
による action 置換

実験結果

- pop-up を自在に操ることが可能
  - 出現回数、サイズ、位置、chrome
- Cookie 情報の流出する危険性を警告として通知
  - Cookie アクセス後の他ドメインへのアクセスを検知
  - XSS 脆弱性をもつ現実のサーバに対して効果を確認

まとめ

- JavaScript を安全に実行するための枠組み
  - チェックコードの動的な挿入をサポート
  - action 置換ポリシーを edit automaton で表現
  - システムの soundness と transparency を証明
  - pop-up 制御と XSS 阻止に対する効果を確認