ALGORITHMIC
GAME SEMANTICS

Andrzej S. Murawski

UNIVERSITY OF LEICESTER
et rec add_identifier id id_type env =
match env with
  [ ] -> [ (id,[id_type]) ]
| (top_id, ty) :: env_tl ->
  if top_id = id then (top_id, ty :: env_tl)
  else (top_id, ty) :: add_identifier id id_type env_tl

et rec rem_identifier id env =
match env with
  [ ] -> raise Undeclared_identifier
| (top_id, ty) :: env_tl ->
  if top_id = id then (if ty=
  env_tl)
  else (top_id, ty) :: rem_identifier id env_tl
FULL ABSTRACTION

$M$ and $N$ are contextually equivalent ($M \cong N$) if they can be used interchangeably in any context (without affecting the computational outcome).

$$\forall C[-]. \quad C[M] \downarrow \iff C[N] \downarrow$$

$$[M] = [N] \iff M \cong N$$
GAMES FOR TYPES

• Who plays?

Opponent  Proponent

C[—]  M
• How do they play?

O begins. Subsequent moves must be justified by earlier moves made by the opposite player.
An arena $A$ is specified by a structure $\langle M_A, \lambda_A, \vdash_A \rangle$.

- $M_A$ is a set of moves.
- $\lambda_A : M_A \rightarrow \{O, P\} \times \{Q, A\}$ is a labelling function.
- $\vdash_A$ is an enabling relation between $\{\dagger\} + M_A$ and $M_A$.
  - If $\dagger \vdash m$ then $\lambda_A(m) = O$ and $n \vdash_A m$ implies $n = \dagger$.
  - If $m \vdash m'$ then $\lambda_A(m) \neq \lambda_A(m')$. 

A justified sequence over arena $A$ is a sequence of moves from $M_A$ together with an associated sequence of pointers satisfying the following conditions.

- The first move is enabled by $\dagger$ and has no outgoing pointer.
- Any other move $m$ must have a pointer to an earlier move $n$ such that $n \vdash_A m$.
A **justified sequence** over arena $A$ is a sequence of moves from $M_A$ together with an associated sequence of pointers satisfying the following conditions.

- The first move is enabled by $\star$ and has no outgoing pointer.
- Any other move $m$ must have a pointer to an earlier move $n$ such that $n \vdash_A m$.

**N.B.** Papers on game semantics use variations on the concept of a justified sequence to suit the programming paradigm being modelled.

A **play** is a justified sequence that additionally satisfies ... We shall write $P_A$ for the set of plays over arena $A$. 

---

Sunday, 25 September 2011
SOME EXAMPLES

- Sequential computation: **alternation**
- Absence of control effects: **well-bracketing**
- First-order store only: **visibility**

In his next move $P$ cannot use $\cdots$ for justification.
HISTORY

All the conditions were already present in


But it took a few years to match them with other computational paradigms.


REASONING WITH GAMES

• Plays have operational flavour.

• The course of play is often described through metaphores.

• This account has not been formalized yet.

• Operational game semantics: marriage of games and traces
Since the 1990s steady stream of full abstraction results have shown that if they can be used interchangeably in any context, then a function `f` such that `f \equiv \lambda x. x` is the set of moves `M` for which `C \vdash M \equiv f`.

<table>
<thead>
<tr>
<th>Questions</th>
<th>0, q_1</th>
<th>q_0, q_1</th>
<th>q, t_1, f_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answers</td>
<td>t_0, 0</td>
<td>t_0, f_0, t_1, f_1</td>
<td>* , t_0, f_0</td>
</tr>
</tbody>
</table>

**Plays**

In CBN, the plays are:

\[ q_0 \xrightarrow{(q_1 \leftarrow b_1)^*} b_0 \]

In CBV, the plays are:

\[ q \xleftarrow{\star} (b_1 \leftarrow b_0)^* \]
STRATEGIES

• Types are interpreted by games.
• Terms are interpreted by strategies.

A strategy $\sigma$ in arena $A$ is a prefix-closed set of plays over $A$ such that

$$s \in \sigma \text{ and } s o \in P_A \text{ implies } so \in \sigma.$$
GAME CONSTRUCTORS

$A_1 \times A_2$

$A_1 \Rightarrow A_2$

$L$

$R$

$A_1 + A_2$
IDENTITY STRATEGY

\[ A \Rightarrow A \]

\[
\begin{array}{ccc}
\cdots & m_L & m_R \\
\cdots & \cdots & \cdots \\
\cdots & m'_R & m'_L \\
\end{array}
\]

\[
\begin{array}{ccc}
\cdots & m'_L & m'_R \\
\cdots & \cdots & \cdots \\
\cdots & m_R & m_L \\
\end{array}
\]
Given $\sigma : A_1 \Rightarrow A_2$ and $\tau : A_2 \Rightarrow A_3$
one can define $\sigma ; \tau : A_1 \Rightarrow A_3$.

- Moves in $A_2$ have a double identity.
- We can exploit the duality to play $\sigma$ and $\tau$ against each other in $A_2$.
- Following the exchange between $\sigma$ and $\tau$ we can hide the interaction in $A_2$ to obtain a play in $A_1 \Rightarrow A_3$. 
COMPOSITION

Given $\sigma : A_1 \Rightarrow A_2$ and $\tau : A_2 \Rightarrow A_3$, one can define $\sigma ; \tau : A_1 \Rightarrow A_3$.

- Moves in $A_2$ have a double identity.
- We can exploit the duality to play $\sigma$ and $\tau$ against each other in $A_2$.
- Following the exchange between $\sigma$ and $\tau$ we can hide the interaction in $A_2$ to obtain a play in $A_1 \Rightarrow A_3$. 

$$\sigma ; \tau = (\sigma |\!\!| A_2 \tau) \setminus A_2$$
COMPOSITIONAL INTERPRETATION

• The game-semantic denotations are obtained compositionally by induction on term structure.

• Free identifiers are interpreted by identity strategies.

• All other cases are handled through composition with suitably-crafted strategies.
POINTERS (CBN)

\[ f : (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \]

\[ f(\lambda x^{\text{int}}.f(\lambda y^{\text{int}}.x)) \]

\[ f(\lambda x^{\text{int}}.f(\lambda y^{\text{int}}.y)) \]
POINTERS (CBV)

\[ f : \text{int} \rightarrow \text{int} \rightarrow \text{int} \]

\[
\text{let val } g = f(0) \text{ in } \\
\text{let val } h = f(1) \text{ in } g(2)
\]

\[
\text{let val } g = f(0) \text{ in } \\
\text{let val } h = f(1) \text{ in } h(2)
\]
FULL ABSTRACTION

$M$ and $N$ are contextually equivalent if and only if they induce the same sets of complete plays (all questions must be answered).

EXAMPLE (O’HEARN)

**Idealized Algol**: an applied lambda calculus over `com`, `int` and `var` with call-by-name evaluation and fixed-point combinators.

\[
p : \text{com} \rightarrow \text{com} \vdash p(\Omega) : \text{com}
\]

\[
p : \text{com} \rightarrow \text{com} \vdash \text{new } x \text{ in } x := 0; \\
\quad p(x := 1); \\
\quad \text{if } x = 0 \text{ then skip else } \Omega : \text{com}
\]

The equivalence of the two terms cannot be validated using state-transformer semantics.
GAME-SEMANTIC ARGUMENT

\[ p : \text{com}_4 \rightarrow \text{com}_2 \vdash p : \text{com}_1 \rightarrow \text{com}_0 \]

\[ \text{run}_0 \text{ run}_2 (\text{run}_3 \text{ run}_1 \text{ done}_1 \text{ done}_3)^* \text{ done}_2 \text{ done}_0 \]

\[ p : \text{com}_4 \rightarrow \text{com}_2 \vdash p(\Omega) : \text{com}_0 \]

\[ \text{run}_0 \text{ run}_2 \text{ done}_2 \text{ done}_0 \]
\[ p(x := 1) \]

- \( p \)
  
  \[ \text{run}_0 \text{run}_2 (\text{run}_3 \text{run}_1 \text{done}_1 \text{done}_3)^* \text{done}_2 \text{done}_0 \]

- \( x := 1 \)
  
  \[ \text{run}_0 \text{write}(1) \text{ok} \text{done}_0 \]

- \( p(x := 1) \)
  
  \[ \text{run}_0 \text{run}_2 (\text{run}_3 \text{write}(1) \text{ok} \text{done}_3)^* \text{done}_2 \text{done}_0 \]
\[ p(x := 1) \]

- \( p \)

\[ \text{run}_0 \text{run}_2 (\text{run}_3 \text{run}_1 \text{done}_1 \text{done}_3)^* \text{ done}_2 \text{ done}_0 \]

- \( x := 1 \)

\[ \text{run}_0 \text{write}(1) \text{ ok} \text{ done}_0 \]

- \( p(x := 1) \)

\[ \text{run}_0 \text{run}_2 (\text{run}_3 \text{write}(1) \text{ ok} \text{ done}_3)^* \text{ done}_2 \text{ done}_0 \]
\( p( x \leftarrow 1 ) \)

- \( p \)
  
  \[ \text{run}_0 \text{ run}_2 ( \text{run}_3 \text{ run}_1 \text{ done}_1 \text{ done}_3 )^* \text{ done}_2 \text{ done}_0 \]

- \( x \leftarrow 1 \)
  
  \[ \text{run}_0 \text{ write}(1) \text{ ok} \text{ done}_0 \]

- \( p(x \leftarrow 1) \)
  
  \[ \text{run}_0 \text{ run}_2 ( \text{run}_3 \text{ write}(1) \text{ ok} \text{ done}_3 )^* \text{ done}_2 \text{ done}_0 \]
\[ p(x := 1) \]

- \( p \)
  \[ \text{run}_0 \text{ run}_2 (\text{run}_3 \text{ run}_1 \text{ done}_1 \text{ done}_3)^* \text{ done}_2 \text{ done}_0 \]

- \( x := 1 \)
  \[ \text{run}_0 \text{ write}(1) \text{ ok} \text{ done}_0 \]

- \( p(x := 1) \)
  \[ \text{run}_0 \text{ run}_2 (\text{run}_3 \text{ write}(1) \text{ ok} \text{ done}_3)^* \text{ done}_2 \text{ done}_0 \]
\[ p(x := 1) \]

- \( p \)
  - \( \text{run}_0 \text{ run}_2 (\text{run}_3 \text{ run}_1 \text{ done}_1 \text{ done}_3)^* \text{ done}_2 \text{ done}_0 \)

- \( x := 1 \)
  - \( \text{run}_0 \text{ write}(1) \text{ ok} \text{ done}_0 \)

- \( p(x := 1) \)
  - \( \text{run}_0 \text{ run}_2 (\text{run}_3 \text{ write}(1) \text{ ok} \text{ done}_3)^* \text{ done}_2 \text{ done}_0 \)
\[ p(x := 1) \]
\[ \text{run}_0 \text{run}_2 (\text{run}_3 \text{write}(1) \text{ ok } \text{done}_3)^* \text{ done}_2 \text{ done}_0 \]

- \( x := 0; p(x := 1); \textbf{if } x = 0 \textbf{ then } () \textbf{ else } \Omega \)

\[ \text{run}_0 \text{ write}(0) \text{ ok } \text{run}_2 (\text{run}_3 \text{write}(1) \text{ ok } \text{done}_3)^* \text{ done}_2 \text{ read } 0 \text{ done}_0 \]

- \textbf{new } \textbf{x } \textbf{ in } x := 0; p(x := 1); \textbf{if } x = 0 \textbf{ then } () \textbf{ else } \Omega \]

\[ \text{run}_0 \text{ run}_2 \text{ done}_2 \text{ done}_0 \]

**new** is interpreted by composition with a strategy ensuring that **read**’s and **write**(i)’s match.

Same complete plays imply equivalence.
RECIPE

• Analyze the underlying process of composition.

• Understand what “really happens”.

• Express strategy-building in a concrete way as an operation on formal languages.

• Remember to encode pointers, if necessary.

• Prove language equivalence using the chosen representation.
\textbf{TYPE ORDER}

\begin{align*}
\text{ord}(\theta) &= \begin{cases} 
0 & \theta \equiv \text{com, int, var} \\
\max(\text{ord}(\theta_1) + 1, \text{ord}(\theta_2)) & \theta \equiv \theta_1 \rightarrow \theta_2
\end{cases}
\end{align*}

- \text{IA}_k \text{ consists of terms of the form}

\[ x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta \]

with \( \text{ord}(\theta_i) < k \) and \( \text{ord}(\theta) \leq k \).

- Looping and recursion are not available in \text{IA}_k.

- We write \( Y_k \) to stress the availability of the fixed-point combinator \( Y_\theta : (\theta \rightarrow \theta) \rightarrow \theta \) for \( \theta \) of order \( k \).
DECIDABILITY

We assume finite ground types!

<table>
<thead>
<tr>
<th>pure</th>
<th>+while</th>
<th>+Y₀</th>
<th>+Y₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA₁</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>IA₂</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>IA₃</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>IA₄</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

The results were obtained using FA, DPDA and VPA.
## BIBLIOGRAPHY

<table>
<thead>
<tr>
<th>FA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan R. Ghica, Guy McCusker: Reasoning about Idealized ALGOL Using Regular Languages. <strong>ICALP 2000</strong>: 103-115</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DPDA</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>VPA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrzej S. Murawski, Igor Walukiewicz: Third-Order Idealized Algol with Iteration Is Decidable. <strong>FoSSaCS 2005</strong>: 202-218</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>undecidability</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrzej S. Murawski: On Program Equivalence in Languages with Ground-Type References. <strong>LICS 2003</strong>: 108-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
COMPLEXITY

Equivalence of terms in beta-normal form.

<table>
<thead>
<tr>
<th></th>
<th>pure</th>
<th>+ while</th>
<th>+ Y₀</th>
<th>+ Y₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA₁</td>
<td>CONP-complete</td>
<td>PSPACE-complete</td>
<td>?</td>
<td>—</td>
</tr>
<tr>
<td>IA₂</td>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
<td>?</td>
<td>—</td>
</tr>
<tr>
<td>IA₃</td>
<td>EXPTIME-complete</td>
<td>EXPTIME-complete</td>
<td>?</td>
<td>—</td>
</tr>
<tr>
<td>IA₄</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Non-elementary in general.
UNDECIDABILITY

• It may seem surprising that program equivalence in a language over finite datatypes is undecidable.

• This is all due to the rich structure of interactions afforded by higher-order types.

• At fourth order there are patterns of interaction between $\Omega$ and $\Pi$ that resemble actions of a queue.

• Moreover, there exists a program that can detect whether $\Omega$ follows the queue-pattern.

• Game semantics tames higher-order interaction.
NONDETERMINISM

May-termination \( \downarrow_{\text{may}} \)
Must-termination \( \downarrow_{\text{must}} \)

\[ \forall C[\_]. \quad C[M] \downarrow_{\text{may}} \iff C[N] \downarrow_{\text{may}} \]

\[ \forall C[\_]. \quad C[M] \downarrow_{\text{must}} \iff C[N] \downarrow_{\text{must}} \]

\[ \forall C[\_]. \quad C[M] \downarrow_{\text{may}} \downarrow_{\text{must}} \iff C[N] \downarrow_{\text{may}} \downarrow_{\text{must}} \]

– May-equivalence

\[ \forall C[\_]. \quad C[M] \downarrow_{\text{may}} \iff C[N] \downarrow_{\text{may}} \]

– Must-equivalence

\[ \forall C[\_]. \quad C[M] \downarrow_{\text{must}} \iff C[N] \downarrow_{\text{must}} \]

– May & Must-equivalence
**MAY-EQUIVALENCE**

Characterization via complete plays still applies.

<table>
<thead>
<tr>
<th></th>
<th>pure</th>
<th>+<strong>while</strong></th>
<th>+$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$EA_1$</td>
<td>PSPACE-complete</td>
<td>EXPSPACE-complete</td>
<td>–</td>
</tr>
<tr>
<td>$EA_2$</td>
<td>EXPSPACE-complete</td>
<td>EXPSPACE-complete</td>
<td>–</td>
</tr>
<tr>
<td>$EA_3$</td>
<td>2-EXPTIME-complete</td>
<td>2-EXPTIME-complete</td>
<td>–</td>
</tr>
<tr>
<td>$EA_4$</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
MUST-EQUIVALENCE

Russell Harmer, Guy McCusker: A Fully Abstract Game Semantics for Finite Nondeterminism. LICS 1999: 422-430

A strategy $\sigma$ on an arena $A$ is a pair $(T_\sigma, D_\sigma)$. The first component $T_\sigma$, known as the traces of $\sigma$, is a non-empty set of even-length legal plays of $A$ satisfying

$$
sab \in T_\sigma \Rightarrow s \in T_\sigma.
$$

We write $\text{dom}(\sigma)$ for the domain of $\sigma$, i.e. the set

$$
\{sa \in L_A \mid \exists b. sab \in T_\sigma\}
$$

and $\text{cc}(\sigma)$ for the contingency closure of $\sigma$, i.e. $T_\sigma \cup \text{dom}(\sigma)$. Given $sa \in \text{dom}(\sigma)$, let $\text{rng}_\sigma(sa) = \{b \in M_A \mid sab \in T_\sigma\}$.

The second component $D_\sigma$ is known as the divergences of $\sigma$; it’s a set of odd-length legal plays of $A$ satisfying

Characterization via quotienting.
WINNING REGIONS

Let $O$ and $P$ play a reachability game over the traces of $\sigma$. $O$ will be declared a winner if he reaches a complete play without encountering any divergences. This induces winning regions for $O$ and $P$.

Two terms are *must-equivalent* if and only if any difference between the induced strategies (trace or divergence) is compensated by a winning region for $P$.

**MUST-EQUIVALENCE**

<table>
<thead>
<tr>
<th></th>
<th>pure</th>
<th>+<strong>while</strong></th>
<th>+<strong>Y₀</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EA₁</strong></td>
<td>PSPACE-complete</td>
<td>2-EXPTIME-complete</td>
<td>-</td>
</tr>
<tr>
<td><strong>EA₂</strong></td>
<td>2-EXPTIME-complete</td>
<td>2-EXPTIME-complete</td>
<td>-</td>
</tr>
<tr>
<td><strong>EA₃</strong></td>
<td>3-EXPTIME-complete</td>
<td>3-EXPTIME-complete</td>
<td>-</td>
</tr>
<tr>
<td><strong>EA₄</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
PROBABILISTIC EQUIVALENCE

\[ \forall C[-]. \quad C[M] \Downarrow_p \iff C[N] \Downarrow_p \]
PROBABILISTIC STRATEGIES

The definition comes in two steps. First of all, we define a prestrategy $\sigma$ on an arena $A$ to be a (set-theoretic) function $\sigma : L^\text{even}_A \to [0, \infty]$. Such a prestrategy is a strategy iff

1. $\sigma(\varepsilon) = 1$;
2. if $sa \in L^\text{odd}_A$ then $\sigma(s) \geq \sum_{t \in \text{ie}(sa)} \sigma(t)$.

PROBABILISTIC LANGUAGE EQUIVALENCE

Two probabilistic programs are equivalent if and only if the corresponding probabilistic strategies assign the same probabilities to all complete plays.

APEX tool

DINING CRYPTOGRAPHERS

Overview

Coin experiments

Self-stabilization

Random tree shapes

Anonymity

• Dining Cryptographers (Chaum)
• Was it one of them?
• Correctness Verification
• Verifying Anonymity
• Anonymity Verification
• What can Cryptographer 1 see?
• Anonymity certification
• DC statistics
• PRISM vs APEX
• Lack of anonymity

Conclusion

Sunday, 25 September 2011
WAS IT ONE OF THEM?

One of the cryptographers paid \( \iff \) 

"Disagree" is odd

\[
f : \{0, 1, 2, 3\} \to \{0, 1\}
\]

\[
x : \text{int}\%4 \vdash
\]

\[
\text{var}\%4 \text{ whopaid}; \text{ var}\%2 \text{ first}; \text{ var}\%2 \text{ left};
\]

\[
\text{ var}\%2 \text{ right}; \text{ var}\%2 \text{ parity}; \text{ var}\%4 \text{ i};
\]

\[
\text{ whopaid} := x; \text{ first} := \text{ coin}; \text{ right} := \text{ first}; \text{ i} := 1;
\]

\[
\text{ while (i) do}
\]

\[
\{ \text{ left} := \text{ if (i=3) then first else coin;}
\]

\[
\text{ if not((left=right)+(whopaid=i))}
\]

\[
\text{ then parity} := \text{ not(parity)};
\]

\[
\text{ right} := \text{ left};
\]

\[
\text{ i} := \text{ i+1}
\]

\[
\}
\]

\[
\text{ parity} : \text{ int}\%2
\]
CORRECTNESS

Overview
Coin experiments
Self-stabilization
Random tree shapes
Anonymity

• Dining Cryptographers (Chaum)
• Was it one of them?
• Correctness Verification
• Anonymity Verification
• What can Cryptographer 1 see?
• Anonymity certification
• DC statistics
• Lack of anonymity

Conclusion

0

1_x, 1
(0,1)

2_x, 1
(1,1)

3_x, 1
ANONYMITY (VIEWS)

cn: var%2, ch: var%2 |- 

var%4 whopaid;
whopaid := 2;

if (whopaid <= 1) then diverge else 
{
    var%2 first; var%2 left; var%2 right; var%4 i;

    first:=coin; right:=first; i:=1;

    while (i) do
    {
        left:=if (i=3) then first else coin;
        if (i=1) then { cn:=right; cn:=left };
        if ((left=right)+(whopaid=i)) then ch:=1 else ch:=0;
        right := left;
        i := i+1
    }
}:: com
WHAT CAN HE SEE?

2 paid

3 paid
WHAT CAN HE SEE?
MORE CRYPTOGRAPHERS
OTHER TOOLS

• Homer


• MAGE

Adam Bakewell, Dan R. Ghica: On-the-Fly Techniques for Game-Based Software Model Checking. TACAS 2008: 78-92
CALL-BY-VALUE EVALUATION

Call-by-value Idealized Algol

**RML**: an ML-like language with integer references, including “bad” ones

\[
\text{ref int} = (\text{unit} \to \text{int}) \times (\text{int} \to \text{unit})
\]

**PRO**: Finite alphabet, if finitely many values!

**CON**: Equivalences relying on ref int may be affected.

Samson Abramsky, Guy McCusker: Call-by-Value Games. CSL 1997: 1-17
SOME SURPRISES?

- unit → unit → unit is problematic.

\[ q \rightarrow \ast q \leftarrow a \cdots q \leftarrow a \]

There are many \( a \)'s to point at...

- \((\text{unit} \rightarrow \text{unit}) \rightarrow (\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit}\) is undecidable.

SOME RESULTS

Assume finite ground types and absence of recursion.

- Regular

\[(\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit} \vdash \text{unit} \rightarrow \text{unit}\]

- Visibly context-free

\[(\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit} \vdash (\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit}\]

SUMMARY

• Many decision procedures have been obtained via game semantics in recent years.

• Some have been implemented and observed to beat alternative approaches.

• Several tools use game semantics as a main engine.

• Ready for “realistic” applications?