Pillage Games

```
In[1]:= Needs["Theorema`"]
```

These formalisations have originally been presented in the following publication:
Manfred Kerber, Colin Rowat, Wolfgang Windsteiger. Using Theorema in the Formalization of Theoretical Economics. Conference on Intelligent Computer Mathematics, 2011 (Calculemus track). No. 6824 in Lecture Notes in Computer Science, Springer, pp. 58–73, 2011

Basic Definitions

▪ An unrelated test function

▪ Agents

We introduce the set of **agents**, also called players. Subsets of this set are called **coalitions**.

```
TS_In[2]:= Definition["agents", any[n], I[n] := {i | }]
                                                i=1,…,n

Compute[I[10], using → Definition["agents"],
 built-in → {Built-in ["Sets"], Built-in ["Quantifiers"], Built-in ["Numbers"]}]

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Pillage games have at least two agents; we formalize this as an additional condition on any number n. This check is intended to be made whenever a set of agents occurs.

```
TS_In[3]:= Definition["appropriateLength", any[n],
            appropriateLength[n] : ⇔ (n ∈ ℕ ⋀ n ≥ 2)]
```

▪ A simple lemma and its proof

The set of agents of a pillage game contains at least agents 1 and 2:

Theorema doesn't know that this is the same as "1∈I[n]" and this is the formulation that we need for the proof of Lemma 2.

```
TS_In[4]:= Lemma["1inI", any[n], with[appropriateLength[n]],
            {1} ⊆ I[n] ]

TS_In[5]:= Lemma["2inI", any[n], with[appropriateLength[n]],
            {2} ⊆ I[n] ]
```

The proof:

```
Prove[Lemma["1inI"], using → {Definition["appropriateLength"], Definition["agents"]},
 built-in → {Built-in ["Numbers"]}, by → SetTheoryPCSProver,
 ProverOptions → {SimplifyFormula → True}]

- ProofObject -
```

The proof object contains some redundant steps; here is how to get rid of them by cleaning up the proof object:

The following lemma is just slightly more complex, but proving it exceeds the default search depth of 30, so we have to increase it. 33 works, 32 won't.

TS_In[6]:= **Lemma["1and2inI", any[n], with[appropriateLength[n]], $\{1, 2\} \subseteq I[n]$]**

```
Prove[Lemma["1and2inI"], using → {Definition["appropriateLength"], Definition["agents"]},
 built-in → {Built-in ["Numbers"]}, by → SetTheoryPCSProver,
 ProverOptions → {SimplifyFormula → True}, SearchDepth → 33]
```

- ProofObject -

Allocations

An **allocation** is a vector of *n* non-negative numbers, which sum to 1 (where *n* is the number of agents). The allocation represents the possessions of each agent.

TODO

TODO

TS_In[7]:= **Definition["allocation", any[x, n],**

$$\mathrm{allocation}_n[x] :\Leftrightarrow \left(n \in \mathbb{N} \bigwedge \mathrm{is\text{-}tuple}[x] \bigwedge (|x| = n) \bigwedge \underset{i \in I[n]}{\forall} x_i \geq 0 \bigwedge \left(\sum_{i \in I[n]} x_i = 1\right)\right)\Big]$$

Monotonicity axioms

Weak Coalition Monotonicity

Taking a new member into a coalition does not decrease the coalition's power (where π is the power function).

TS_In[8]:=

Definition["WC", any[π, n], bound[allocation$_n$[x]],

$$\mathrm{WC}[\pi, n] :\Leftrightarrow \underset{\substack{C1, C2 \\ C1 \subset C2 \bigwedge C2 \subseteq I[n]}}{\forall} \underset{x}{\forall} \pi[C2, x] \geq \pi[C1, x]\Big]$$

Weak Resource Monotonicity

Weakly increasing the holdings of a coalition's members does not decrease the coalition's power.

TODO

TS_In[9]:=

Definition["WR", any[π, n], bound[allocation$_n$[x], allocation$_n$[y]],

$$\mathrm{WR}[\pi, n] :\Leftrightarrow \left(n \in \mathbb{N} \bigwedge \underset{\substack{C \\ C \subseteq I[n]}}{\forall} \underset{x, y}{\forall} \left(\left(\underset{i \in C}{\forall} y_i \geq x_i\right) \Rightarrow \pi[C, y] \geq \pi[C, x]\right)\right)\Big]$$

To make things easier for Theorema, we additionally state the contrapositive of this axiom.

TODO

TS_In[10]:= **Definition["WR contrapositive", any[π, n], bound[allocation$_n$[x], allocation$_n$[y]],**

$$\mathrm{WR}[\pi, n] :\Leftrightarrow \left(n \in \mathbb{N} \bigwedge \underset{\substack{C \\ C \subseteq I[n]}}{\forall} \underset{x, y}{\forall} \left(\pi[C, x] > \pi[C, y] \Rightarrow \left(\underset{i \in C}{\exists} x_i > y_i\right)\right)\right)\Big]$$

Strong Resource Monotonicity

Strictly increasing the holdings of a coalition's members strictly increases the coalition's power. (This only really makes sense for non-empty coalitions.)

TODO

TS_In[11]:=

```
Definition["SR", any[π, n], bound[allocationₙ[x] , allocationₙ[y]],
```

$$\mathrm{SR}[\pi, n] :\Leftrightarrow \left(n \in \mathbb{N} \bigwedge \underset{\substack{C \\ C \subseteq I[n] \wedge C \neq \emptyset}}{\forall} \; \underset{x,y}{\forall} \left(\left(\underset{i \in C}{\forall} y_i > x_i \right) \Rightarrow \pi[C, y] > \pi[C, x] \right) \right)]$$

Power function

The **power function** of a pillage game satisfies the three axioms stated above.

TS_In[12]:=

```
Definition["powerfunction", any[π, n],
```

$$\mathrm{powerfunction}[\pi, n] :\Leftrightarrow \bigwedge \begin{cases} \mathrm{WC}[\pi, n] \\ \mathrm{WR}[\pi, n] \\ \mathrm{SR}[\pi, n] \end{cases}]$$

For convenient reuse, we package the definition of a power function and its dependencies into a *theory*.

Lemma 1

The power of a coalition depends only on the holdings of its members, not on the holdings of the other agents. I.e. any power function π(C,x) can be represented by another, $\pi' \left(C, \{x_i\}_{i\in C}\right)$, which depends only on the resource holdings of its coalition members.

Note the precedences of ∀ vs. ⇒!

TS_In[14]:=

```
Lemma["powerfunction-independent", any[π, n, C, x, y],
```

$$\mathrm{with}[\mathrm{allocation}_n[x] \wedge \mathrm{allocation}_n[y] \wedge C \subseteq I[n] \wedge \mathrm{powerfunction}[\pi, n]],$$

$$\underset{i \in C}{\forall} (x_i = y_i) \Rightarrow (\pi[C, x] = \pi[C, y])]$$

Auxiliary axiom: trichotomy of the real numbers

For proving these, we need one axiom for the trichotomy of ≤, =, and ≥ on the real numbers.

There is a potential problem in the next axiom, since it should be stated for real numbers only.

TS_In[15]:=

```
Axiom["trichotomy", any[a, b],
 (a = b) ⇔ (a ≥ b ⋀ b ≥ a)]
```

Proof

The only axiom about power functions that we actually need for this proof is the weak resource monotonicity.

Note once more that the search depth has been fine-tuned manually.

```
PLemma1 = Prove[Lemma["powerfunction-independent"],
  using → {Definition["powerfunction"], Definition["WR"], Axiom["trichotomy"]},
  by → PredicateProver, SearchDepth → 86]
```

```
- ProofObject -
```

TODO

```
Save["eco1.m", {PLemma1}]
```

Alternative proof using the whole theory

We can also use the whole power functions theory to do the proof.

This requires a higher search depth: The more redundant information you give to a theorem prover, the more difficult you make its task.

```
PLemma2 = Prove[Lemma["powerfunction-independent"],
  using → {Theory["powerfunction"], Axiom["trichotomy"]},
  by → PredicateProver, SearchDepth → 250]
```

```
$Aborted
```

```
Save["eco1full.m", {PLemma2}]
```

Simplifying the proofs to show only the necessary steps

TODO

```
? branches
```

```
Option of ProofSimplifier with possible values: Proved, Pending, Failed,
  Disproved and list combinations of these. All (default) means list of all.
```

```
PLemma1Simp = Block[{$RecursionLimit = Infinity}, Transform[PLemma1,
   by → ProofSimplifier, TransformerOptions → {branches → Proved, steps → Useful}]]
```

```
- ProofObject -
```

```
Save["ecosimp1.m", {PLemma1Simp}]
```

TODO

```
PLemma2Simp = Block[{$RecursionLimit = Infinity}, Transform[PLemma2,
   by → ProofSimplifier, TransformerOptions → {branches → Proved, steps → Useful}]]
```

```
Not used: {Definition (allocation), Definition (WC), Definition (SR)}

Not used: {Definition (allocation), Definition (WC), Definition (SR)}
```

```
Save["ecosimp1full.m", {PLemma2Simp}]
```

Reference: prover options

```
Options[SetTheoryPCSProver]
```

```
{DisableProver → {PND}, EarlyRewriting → False, TransformRanges → True,
 AllowIntroduceQuantifiers → False, ApplyBuiltIns → True, BackChaining → False,
 BackChainingDisjunction → False, BackChainingEquivalence → False,
 BackChainingImplication → False, ChooseFromFiniteSet → False,
 DisableInferenceRule → {"KBComposeIntersection", "KBFiniteChoice", "KBInferNonEmpty"},
 EarlyCaseDistinction → True, GRWTarget → {"kb", "goal"}, InferMembershipIntersection → False,
 KBRWOnlySimplification → False, MatchExistential → "TryAllAtOnce",
 ModusPonensException → None, ModusPonensKB → True, PNDLevel → 1,
 RWBooleanLiteralCombinations → True, RWCombine → False, RWExistentialGoal → False,
 RWHigherOrder → False, RWInnermost → True, RWInsideQuantifiers → False,
 RWSetOperators → False, RWTuples → True, SemanticMatch → True, SimplifyFormula → False,
 STPFunctionProperties → True, STPLevel → 100, STPMembershipByInclusion → False,
 TryAlternatives → False, TrySubgoal → False, TrySubgoalDisjunction → False,
 TrySubgoalEquivalence → False, TrySubgoalImplication → False,
 UseCyclicRules → False, UseEqualitiesFirst → True, UseNonMembership → True}
```

```
Options[PredicateProver]
```

■ Show the most recent proof (including aborted ones)

```
ProofShow[]
```

■ Statistics about the proof objects

TODO

```
Depth[PLemma1]
```

```
Depth[PLemma2]
```

```
Depth[PLemma1Simp]
```

```
Depth[PLemma2Simp]
```

Auxiliary Concepts

■ Permuted Allocation

■ Allocation permutation operator

We define an operator that returns the k-th element of an **allocation permuted** by σ:

TS_In[16]:= Definition["perm", any[x, σ, k],
  $\text{perm}[x, \sigma]_k := x_{\sigma[k]}$]

■ Permutation

A **permutation** is a bijective function from a set onto itself.

TS_In[17]:= Definition["permutation", any[σ, A],
  $\text{permutation}[\sigma, A] :\Leftrightarrow \sigma :: A \xrightarrow{\text{bij.}} A$]

- **Allocations are closed unter permutation**

If you permute an allocation, it's still an allocation:

```
TS_In[18]:= Proposition["alloc perm", any[n, σ, x], with[allocation_n[x] ⋀ permutation[σ, I[n]]],
              allocation_n[perm[x, σ]]]
```

Making one aspect of the above explicit: Permuting an allocation doesn't change the sum of the agents' holdings.

TS_In[19]:= Proposition["sum perm", any[σ, A, x], with[permutation[σ, A]],

$$\sum_{i\in A} x_{\sigma[i]} = \sum_{i\in A} x_i$$

]

To make it easier to prove that allocations are closed under permutation, we take it for granted that permutations don't change the sum of an allocation.

TODO

TODO

```
Prove[Proposition["alloc perm"],
 using → {Proposition["sum perm"], Definition["allocation"], Definition["perm"]},
 by → SetTheoryPCSProver, ProverOptions → {AllowIntroduceQuantifiers → True}]

- ProofObject -
```

Allocation Swap

- **The "swap" permutation**

This permutation swaps the i-th and j-th components of a tuple.

TS_In[20]:= Definition["swap", any[i, j, k],

$$\sigma_{i,j}[k] := \begin{cases} j & \Leftarrow k = i \\ i & \Leftarrow k = j \\ k & \Leftarrow \text{otherwise} \end{cases}$$

]

- **"swap" is a permutation**

The permutation that swaps agent 1 and 2 really is a permutation, on any admissible set of agents (which always has at least 2 agents).

```
TS_In[21]:= Proposition["swapperm", any[n], with[appropriateLength[n]],
              permutation[σ_{1,2}, I[n]]]
```

TODO

```
Prove[Proposition["swapperm"],
 using -> {Definition["swap"], Definition["permutation"]}, by -> SetTheoryPCSProver]

$Aborted
```

- **Swapping is idempotent**

Swapping twice restores the same permutation.

```
TS_In[22]:= Proposition["swap idempotent", any[i], σ1,2[σ1,2[i]] = i]
```

TODO

```
Prove[Proposition["swap idempotent"], using → Definition["swap"], by → SetTheoryPCSProver]

- ProofObject -
```

When swapping the first two elements in a tuple, the second element in the new tuple is equal to the first of the original.

```
TS_In[23]:= Lemma["perm swap", any[x],
             perm[x, σ1,2]2 = x1]
```

It is instructive to try the proof first without letting Theorema know about equality of numbers, and then with this built-in knowledge.

- Theorema's provers apply purely logical inference rules.
- Any additional knowledge beyond that has to be provided explicitly: either by "using" additional axioms or previously established results, or by letting Theorema know that it should make use of some computational built-ins.
- Activating the "=" built-in connective means that Theorema will understand that "2=1" is false.

```
Prove[Lemma["perm swap"], using → {Definition["perm"], Definition["swap"]},
 by → SetTheoryPCSProver(*,built-in →Built-in ["Connectives"]["="]*)]

- ProofObject -
```

Additional Axioms

Anonymity

Power is invariant under permutations of the agents, i.e. an agent's identity is irrelevant to a coalition's power; all that matters is its presence or absence, and its holdings. (Consequently, dominance and the stable set are invariant under permutations of the agents.)

Formalization: From a coalition Cx and an allocation x, obtain Cy and y by permuting the agents using a permutation σ. Then, Cy is as powerful at y as Cx is at x.

TS_In[36]:=

$$\text{Definition}\Big[\text{"AN"}, \text{any}[\pi, n], \text{bound}[\text{allocation}_n[x], \text{allocation}_n[y]],$$
$$\text{AN}[\pi, n] :\Leftrightarrow \left(n \in \mathbb{N} \wedge \left(\underset{\substack{\sigma \\ \text{permutation}[\sigma, I[n]]}}{\forall}\ \underset{\substack{Cx, Cy, x, y \\ Cx \subseteq I[n] \wedge Cy \subseteq I[n]}}{\forall}\ \underset{i}{\forall} \left(((i \in Cx) \Leftrightarrow (\sigma[i] \in Cy)) \wedge (x_i = y_{\sigma[i]})\right) \Rightarrow (\pi[Cx, x] = \pi[Cy, y])\right)\right)\Big]$$

The power of one agent does not change under a permutation that swaps agents 1 and 2:

```
TS_In[25]:= Lemma["AN swap", any[n, π, x], with[appropriateLength[n] ⋀ AN[π, n] ⋀ allocationn[x]],
             π[{1}, x] = π[{2}, perm[x, σ1,2]]]
```

```
PLemmaANperm =
 Prove[Lemma["AN swap"], using → {Proposition["swapperm"], Proposition["swap idempotent"],
    Proposition["alloc perm"], Lemma["1inI"], Lemma["2inI"], Definition["AN"],
    Definition["perm"], Definition["swap"]}, by → SetTheoryPCSProver, SearchDepth → 80,
  ProverOptions → {AllowIntroduceQuantifiers → True, UseCyclicRules → False,
    EarlyRewriting → False, GRWTarget -> {"goal", "kb"}, DisableProver → {STC}}]

$Aborted

Transform[PLemmaANperm, TransformerOptions → {branches → Proved, steps → Useful}]

ProofShow[]
```

- **Continuity in resources**
- **Responsiveness**
- **Domination**

- **Preliminary: Win and Lose Set**

For any transition from an allocation x to an allocation y, the **win set** W is the set of agents who benefit from the transition, whereas the **lose set** L is the set of agents who suffer from the transition. We define both of them at once.

TS_In[28]:=

$$\text{Definition}\Big[\text{"WinLose"}, \text{any}[n, x, y], \quad W[n,x,y] := \{\underset{i\in I[n]}{i \mid} y_i > x_i\}\ \text{"W"} \quad L[n,x,y] := \{\underset{i\in I[n]}{i \mid} x_i > y_i\}\ \text{"L"}\Big]$$

- **Domination**

An allocation x **dominates** an allocation y if the coalition given by the win set of the transition x→y is more powerful in x than the lose set is.

TS_In[29]:=

$$\text{Definition}[\text{"dominates"}, \text{any}[\pi, n, x, y], \ \text{dominates}[y, x, \pi, n] :\Leftrightarrow \pi[W[n,x,y], x] > \pi[L[n,x,y], x]]$$

Lemma 2

When the opposing coalitions consist of one element each and the power function is anonymous then the coalition which wins is the one with the bigger holdings.

- **Direct statement (just ⇛" part)**

For some arbitrary transition x→y we assume wlog that agent 1 wins and agent 2 loses.

TODO

TS_In[30]:=

$$\text{Lemma}\Big[\text{"ANdominates"}, \text{any}[n, x, y], \text{with}[\text{appropriateLength}[n] \wedge \text{allocation}_n[x] \wedge \text{allocation}_n[y] \wedge (W[n,x,y] = \{1\}) \wedge (L[n,x,y] = \{2\})], \underset{\substack{\pi \\ AN[\pi,n]\wedge \text{powerfunction}[\pi,n]}}{\forall} (\text{dominates}[y, x, \pi, n] \Rightarrow x_1 > x_2)\Big]$$

Without auxiliary theory, Theorema can't prove this within reasonable time. (Last time I tried it for half an hour on a dual-core 2.5 GHz machine.)

```
PLemmaANdominates =
 Prove[Lemma["ANdominates"], using → {Lemma["AN swap"], Definition["dominates"],
    Definition["powerfunction"], Definition["SR"]}, by → SetTheoryPCSProver,
  SearchDepth → 50, ProverOptions → {AllowIntroduceQuantifiers → True,
    UseCyclicRules → True, EarlyRewriting → False, DisableProver → {STKBR, STC}}]
```

```
$Aborted
```

```
ProofShow[]
```

Alternative formulation (incomplete)

TODO

Lemma["ANdominates-aux", any[n, x, y],
 with[$\text{allocation}_n[x] \wedge \text{allocation}_n[y] \wedge (y_i > x_i \Leftrightarrow (i = 1)) \wedge (y_i < x_i \Leftrightarrow (i = 2))$],
 $\underset{\text{powerfunction}[\pi,n]\wedge \text{AN}[\pi,n]}{\underset{\pi}{\forall}}$ dominates[y, x, π, n] $\Leftrightarrow x_1 > x_2$]

A technical approach that works

Preliminaries

convenience definition of an allocation with an admissible number of agents:

```
TS_In[31]:= Definition["appAlloc", any[n, x],
             appAlloc[n, x] :⇔ (appropriateLength[n] ∧ allocation_n[x])]
```

convenience definition of a well-defined pair of allocation and anonymous power function (not used below):

```
TS_In[32]:= Definition["appPowAlloc", any[n, π, x],
             appPowAlloc[n, π, x] :⇔ (appAlloc[n, x] ∧ AN[π, n])]
```

A straightforward result that combines several preliminaries we need:

- The power of one agent does not change under a permutation that swaps agents 1 and 2, and
- the permutation of an allocation by swapping agents 1 and 2 is still an allocation

TS_In[33]:= Lemma["AN all", any[n, π, x], with[appAlloc[n, x] ∧ AN[π, n]],

$$\bigwedge \begin{cases} \pi[\{1\}, x] = \pi[\{2\}, \text{perm}[x, \sigma_{1,2}]] \\ \text{allocation}_n[\text{perm}[x, \sigma_{1,2}]] \end{cases}$$]

```
Prove[Lemma["AN all"], using → {Lemma["AN swap"], Proposition["alloc perm"],
   Proposition["swapperm"], Definition["appAlloc"]}, by → PredicateProver]
```

```
- ProofObject -
```

The actual lemma (⇐” part)

TODO

```
TS_In[34]:= Lemma["ANdominates 1", any[n, x, y],
              with[appAlloc[n, x] ⋀ allocation_n[y] ⋀ (W[n, x, y] = {1}) ⋀ (L[n, x, y] = {2})],
                       ∀                    (dominates[y, x, π, n] ⇒ x₁ > x₂)]
                       π
              AN[π,n]⋀powerfunction[π,n]
```

The proof and its post-processing:

```
TS_In[37]:= PLemmaANdominates =
              Prove[Lemma["ANdominates 1"], using → {Definition["dominates"], Lemma["AN all"],
                 Definition["WR contrapositive"], Lemma["2inI"], Definition["powerfunction"],
                 Lemma["perm swap"], Definition["appAlloc"]}, by → SetTheoryPCSProver,
               SearchDepth → 80, ProverOptions → {AllowIntroduceQuantifiers → False, RWCombine → True,
                 DisableProver → {STC}, DisableInferenceRule → {¢KBSetEquality, ¢KBInclusion}}]

TS_Out[37]=  - ProofObject -

          Block[{$RecursionLimit = ∞}, ProofShow[]]

TS_In[38]:= Block[{$RecursionLimit = ∞},
              Transform[PLemmaANdominates, TransformerOptions → {branches → Proved, steps → Useful}]]

TS_Out[38]=  - ProofObject -

          Block[{$RecursionLimit = ∞}, ProofShow[PLemmaANdominates]]
```

- **The actual lemma (⇍" part)**

```
TS_In[35]:= Lemma["ANdominates 2", any[n, x, y],
              with[appAlloc[n, x] ⋀ allocation_n[y] ⋀ (W[n, x, y] = {1}) ⋀ (L[n, x, y] = {2})],
                       ∀                    (x₁ > x₂ ⇒ dominates[y, x, π, n])]
                       π
              AN[π,n]⋀powerfunction[π,n]
```

The proof:

```
TS_In[39]:= Prove[Lemma["ANdominates 2"],
              using → {Definition["dominates"], Lemma["AN all"], Definition["SR"], Lemma["2inI"],
                Definition["powerfunction"], Lemma["perm swap"], Definition["appAlloc"]},
              builtin  → {Builtin ["Operators"]["¬", "="]}, by → SetTheoryPCSProver,
              SearchDepth → 80, ProverOptions → {AllowIntroduceQuantifiers → False, RWCombine → True,
                DisableProver → {STP}, DisableInferenceRule → {¢KBSetEquality, ¢KBInclusion}}]

TS_Out[39]=  - ProofObject -

          Block[{$RecursionLimit = ∞}, ProofShow[]]

TS_In[40]:= Block[{$RecursionLimit = ∞},
              Transform[%, TransformerOptions → {branches → Proved, steps → Useful}]]

TS_Out[40]=  - ProofObject -
```

```
Options[SetTheoryPCSProver]

{DisableProver → {PND}, EarlyRewriting → False, TransformRanges → True,
 AllowIntroduceQuantifiers → False, ApplyBuiltIns → True, BackChaining → False,
 BackChainingDisjunction → False, BackChainingEquivalence → False,
 BackChainingImplication → False, ChooseFromFiniteSet → False,
 DisableInferenceRule → {KBComposeIntersection, KBFiniteChoice, KBInferNonEmpty},
 EarlyCaseDistinction → True, GRWTarget → {kb, goal}, InferMembershipIntersection → False,
 KBRWOnlySimplification → False, MatchExistential → TryAllAtOnce, ModusPonensException → None,
 ModusPonensKB → True, PNDLevel → 1, RWBooleanLiteralCombinations → True,
 RWCombine → False, RWExistentialGoal → False, RWHigherOrder → False, RWInnermost → True,
 RWInsideQuantifiers → False, RWSetOperators → False, RWTuples → True, SemanticMatch → True,
 SimplifyFormula → False, STPFunctionProperties → True, STPLevel → 100,
 STPMembershipByInclusion → False, TryAlternatives → False, TrySubgoal → False,
 TrySubgoalDisjunction → False, TrySubgoalEquivalence → False, TrySubgoalImplication → False,
 UseCyclicRules → False, UseEqualitiesFirst → True, UseNonMembership → True}

ResetComputation[]
```