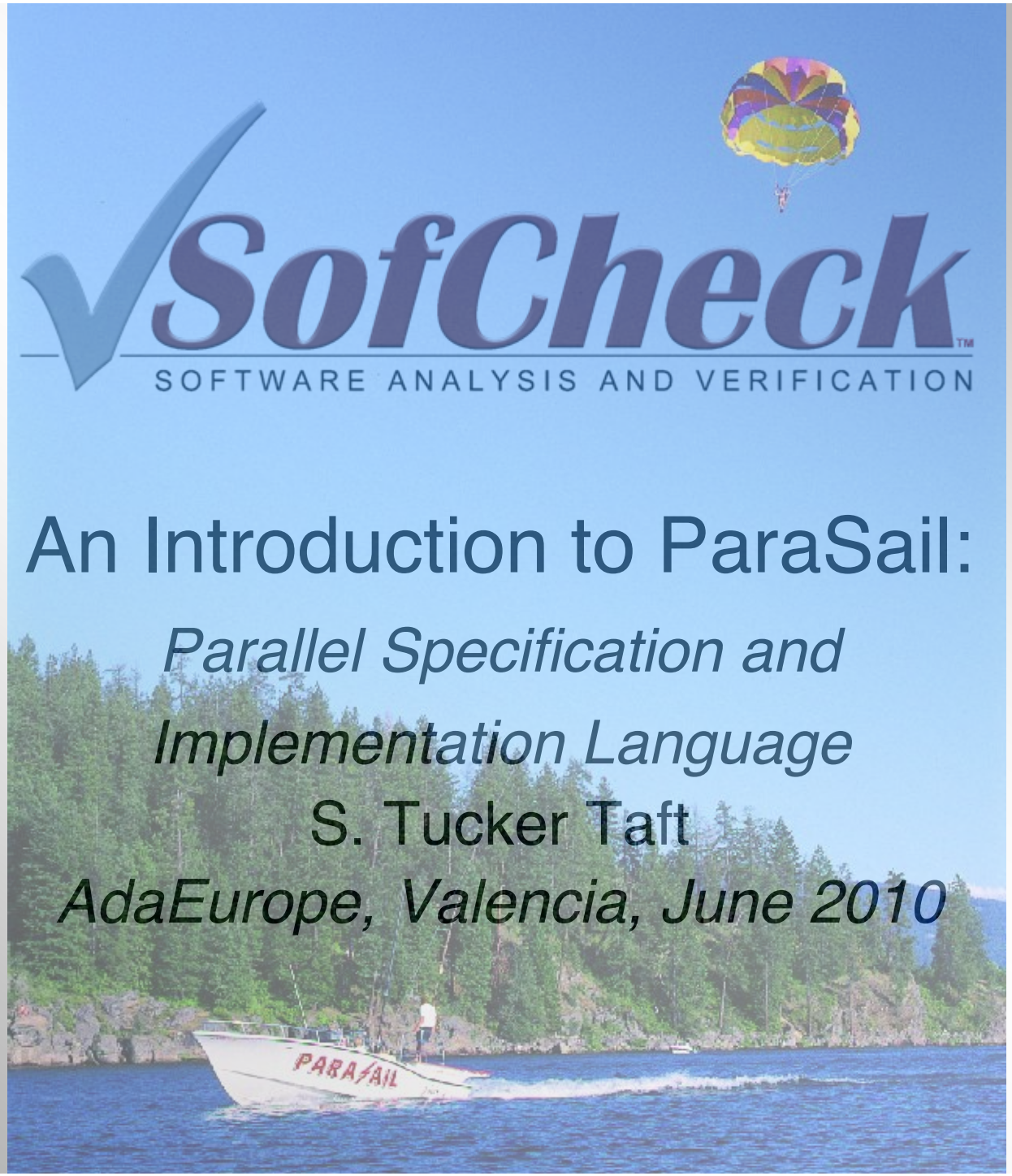
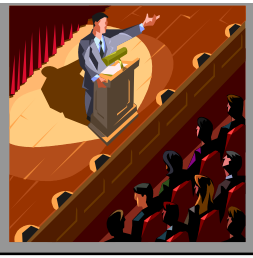


An Introduction to ParaSail:

Parallel Specification and Implementation Language

S. Tucker Taft

AdaEurope, Valencia, June 2010



Outline of Presentation

- Why design a new language for safety-critical systems from scratch?
- What makes ParaSail interesting?
- Parallelism in ParaSail
- Annotations in ParaSail
- More examples of ParaSail
- How does ParaSail compare?
- Open issues in ParaSail

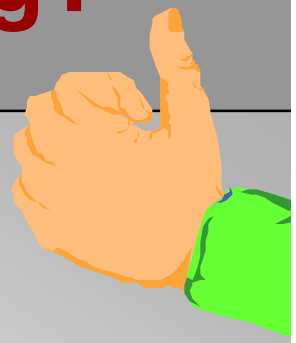
SofCheck

Why Design A New Language for Safety-Critical Systems?

- 80+% of safety-critical systems are developed in C and C++, two of the least safe languages invented in the last 40 years
- SPARK needs some real competition
- Every 32 years you should start from scratch
- In 10 years, many chips will have 64+ cores
- If we don't, someone else will
- It's what I do

SofCheck

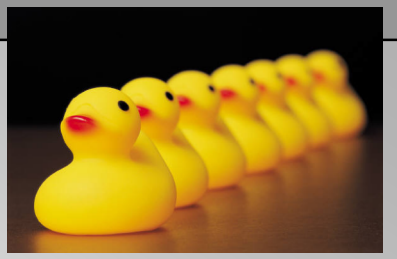
What makes ParaSail Interesting?



- Pervasive (implicit and explicit) parallelism
- Inherently safe:
  - preconditions, postconditions, constraints, etc., integrated throughout the syntax
  - no global variables
  - no run-time checks; all checking at compile-time
- Small number of flexible concepts:
  - Modules, Types, Objects, Operations
- User-defined literals, indexing, aggregates, physical units checking
- It's cool

SofCheck

Parallelism in ParaSail



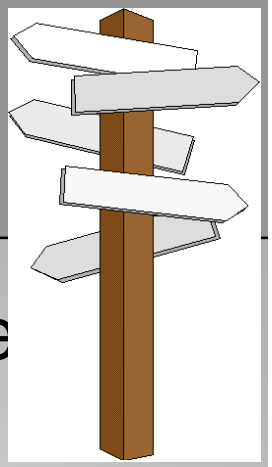
- Parallel by default
  - parameters are evaluated in parallel
  - have to work harder to make code run sequentially
- Easy to create even more parallelism
  - Process(X) || Process(Y) || Process(Z);
- Lock-based and lock-free concurrent objects
  - Lock-based objects also support queued access
  - User-defined delay and timed call based on queued access
- No global variables
  - Can only access or update variable state via parameters
- Compiler prevents aliasing and unsafe access to non-concurrent variables

SofCheck

Examples of ParaSail Parallelism

```
Z := F(U) + G(V);     // F(U) and G(V) eval'ed in parallel
Process(A) || Process(B) || Process(C);  // All 3 in parallel

for X => Root then X.Left || X.Right while X not null
  concurrent loop
    Process(X);         // Process called on each node in parallel
end loop;

concurrent interface Box<Element is Assignable<>> is
    function Create() -> Box;  // Creates an empty box
    procedure Put(var M : locked Box; E : Element);
    function Get(var M : queued Box) -> Element; // May wait
    function Get_Now(var M : locked Box) -> optional Element;
end interface Box;

type Item_Box is Box<Item>;
var My_Box : Item_Box := Create();
```

Annotations in ParaSail



- Preconditions, Postconditions, Constraints, e
  all use same Hoare-like syntax: {X != 0}
- All assertions are checked at compile-time
  - no run-time checks inserted
- Location of assertion determines whether is a:
  - precondition (before “->”)
  - postcondition (after “->”)
  - assertion (between statements)
  - constraint (in type definition)

SofCheck

Examples of ParaSail Annotations

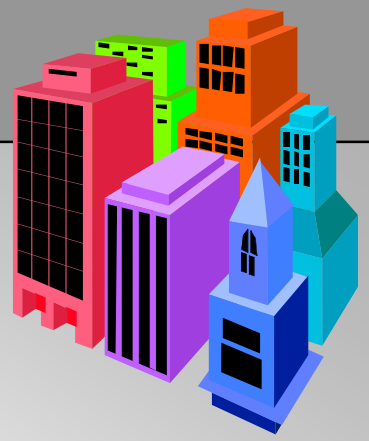
```
interface Stack <Component is Assignable<>; Size_Type is Integer<>> is

    function Max_Stack_Size(S : Stack) -> Size_Type {Max_Stack_Size > 0};

    function Count(S : Stack) -> Size_Type
      {Count <= Max_Stack_Size(S)};

    function Create(Max : Size_Type {Max > 0}) -> Stack
      {Max_Stack_Size(Create) == Max and Count(Create) == 0};

    function Is_Empty(S : Stack) -> Boolean
      {Is_Empty == (Count(S) == 0)};

    function Is_Full(S : Stack) -> Boolean
      {Is_Full == (Count(S) == Max_Stack_Size(S))};

    procedure Push(var S : Stack {not Is_Full(S)}; X : Component)
      {Count(S') == Count(S) + 1};

    function Top(S : Stack {not Is_Empty(S)}) -> Component;

    procedure Pop(var S : Stack {not Is_Empty(S)})
      {Count(S') == Count(S) - 1};

end interface Stack;
```

SofCheck

More Annotation Examples

```
type Age is new Integer<First => 0, Last => 200>;
type Youth is Age {Youth <= 20};
type Senior is Age {Senior >= 50};

function GCD(X, Y : Integer {X > 0 and Y > 0}) -> Integer
   {GCD > 0 and GCD <= X and GCD <= Y and
    X mod GCD == 0 and Y mod GCD == 0} is
  var Result := X;      {Result > 0 and X mod Result == 0}
  var Next := Y mod X; {Next <= Y and Y - Next mod Result == 0}

  while Next != 0 loop
    {Next > 0 and Next < Result and Result <= X}
    const Old_Result := Result;
    Result := Next; {Result < Old_Result}
    Next := Old_Result mod Result;
    {Result > 0 and Result <= Y and Old_Result - Next mod Result == 0}
  end loop;

  return Result;
end function GCD;
```

Overall ParaSail Model



- ParaSail has four basic concepts:
  - Module
    - has an Interface, and Classes that implement it
    - **interface** M <Formal **is** Int<>> **is ...**
  - Type
    - is an instance of a Module
    - **type** T **is** M <Actual>;
  - Object
    - is an instance of a Type
    - **var** Obj : T := T::Create(...);
  - Operation
    - is defined in a Module, and
    - operates on one or more Objects of specified Types.

SofCheck

User-defined Indexing, Literals, etc.

- User-defined indexing
  - Any type with **operator** “[]” defined
- User-defined literals
  - Any type with **operator** “from_univ” defined from:
    - Univ_Integer, Univ_Real,
    - Univ_String, Univ_Character
    - Univ_Enumeration
- User-defined ordering
  - Define single binary **operator** “=?” (pronounced “*compare*”)
  - Returns #less, #equal, #greater, #unordered
  - Implies “<=“, “<“, “==“, “!=“, “>”, “>=“, “in X..Y”, “not in X..Y”

SofCheck

More Examples of ParaSail

```
concurrent class Box <Element is Assignable<>> is
    var Content : optional mutable Element; // starts null and can change size
  exports
    function Create() -> Box is  // Creates an empty box
      return (Content => null);
    end function Create;

    procedure Put(var M : locked Box; E : Element) is
      M.Content := E;
    end procedure Put;

    function Get(var M : queued Box) -> Element // May wait
        queued until Content not null is
      const Result := M.Content;
      M.Content := null;
      return Result;
    end function Get;

    function Get_Now(var M : locked Box) -> optional Element is
      return M.Content;
    end function Get_Now;
end class Box;
```

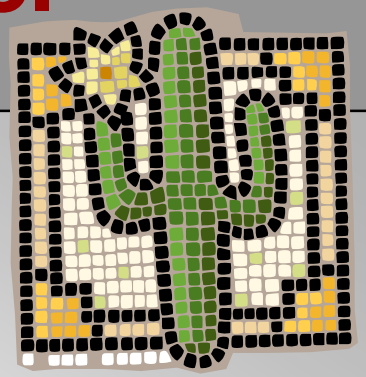
SofCheck

Clock Example

```
abstract concurrent interface Clock <Time_Type is Ordered<>> is
  function Now(C : Clock) -> Time_Type;
  procedure Delay_Until(C : queued Clock; Wakeup : Time_Type)
    {Now(C') >= Wakeup};  // queued until Now(C) >= Wakeup
end interface Clock;

concurrent interface Real_Time_Clock<...> extends Clock<...> is
  function Create(...) -> Real_Time_Clock;
  ...
end interface Real_Time_Clock;

var My_Clock : Real_Time_Clock <...> := Create(...);
const Too_Late := Now(My_Clock) + Max_Wait;

select  // multi-way parallel queued call
     const Data := Get(My_Box) => Process(Data);
  || Delay_Until(My_Clock, Wakeup => Too_Late) =>
          Put_Line(Out_Stream, "My_Box not filled in time");
end select;
```

SofCheck

Walk Parse Tree in Parallel



```
type Node_Kind is Enum < [#leaf, #unary, #binary] >;
  ...
for X => Root while X not null loop
  case X.Kind of
    #leaf =>
      Process_Leaf(X);
    #unary =>
      Process_Unary(X) ||
      continue loop with X => X.Operand;
    #binary =>
      Process_Binary(X) ||
      continue loop with X => X.Left ||
      continue loop with X => X.Right;
  end case;
end loop;
```

SofCheck

How does ParaSail Compare to ...

- C/C++ -- built-in safety; built-in parallelism
- Ada -- eliminates race conditions, increases parallelism, eliminates run-time checks, simplifies language
- Java -- eliminates race conditions, increases parallelism, avoids garbage collection

SofCheck

Problems with Tasks/Threads
(courtesy of Ted Baker)

- Implicitly share access to global data
  - encourages undisciplined sharing
  - hides data flow within internal task logic
- Mix concerns that should be separable
  - semantics vs. performance
- Limit concurrency, ability to use more cores
  - hard coded
- Limit fine-grained concurrency
  - single thread of control, heavy weight

Problems with Protected Objects
(courtesy of Ted Baker)

- Implicitly share access to global data
  - same as with tasks
- Overly general & overly complex semantics
  - limit cache-friendly optimization

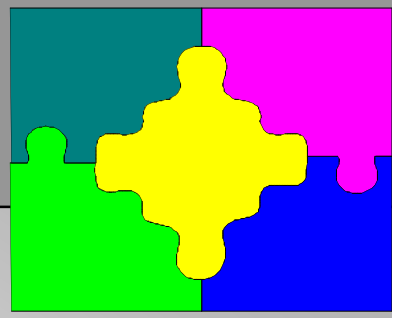
Some of the Open Issues in ParaSail

- Some syntactic details
  - e.g. postconditions:
    - {Count(S') == Count(S) + 1} vs.
    - {Count(S) == Count(old(S)) + 1} vs. ???
  - e.g. formal "writable" parameters
    - procedure Foo(var X : T); vs.
    - procedure Foo(X : in out T);

- Do we need pointers at all?
  - if so, when and where?
- If no global variables, how best to provide access to global "singleton" objects from environment
  - such as "the" database or "the" user or "the" filesystem
  - "Context" object with singletons as components passed to main subprogram?

SofCheck

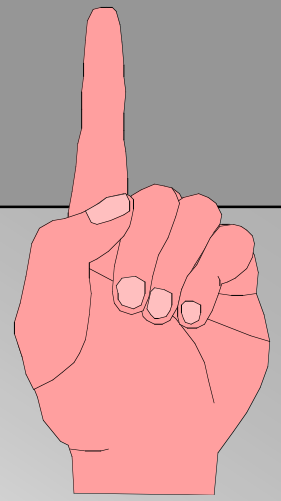
Ultimate Test: Physical Units Example



```
interface Float_With_Units
  <Base is Float<>; Name : Univ_String; Short_Hand : Univ_String;
   Unit_Dimensions : Array <Element_Type => Univ_Real,
     Index_Type => Dimension_Enum> := [others => 0.0]; Scale : Univ_Real> is

    operator "from_univ"(Value : Univ_Real)
      {Value in Base::First*Scale .. Base::Last*Scale} -> Float_With_Units;

    operator "to_univ"(Value : Float_With_Units) -> Result : Univ_Real
      {Result in Base::First*Scale .. Base::Last*Scale};

    operator "+"(Left, Right : Float_With_Units) -> Result : Float_With_Units
      {[[Result]] == [[Left]] + [[Right]]};

    operator "=?"(Left, Right : Float_With_Units) -> Ordering;

    operator "*"(Left : Float_With_Units; Right : Right_Type is Float_With_Units<>)
      -> Result : Result_Type is Float_With_Units<Unit_Dimensions =>
                                    Unit_Dimensions + Right_Type.Unit_Dimensions>
        {[[Result]] == [[Left]] * [[Right]]};

    operator "/"(Left : Left_Type is ...
end interface Float_With_Units;

type Meters is Float_With_Units<Name => "centimeters", Short_Hand => "cm",
  Unit_Dimensions => [#m => 1.0, #k => 0.0, #s => 0.0], Scale => 0.01>;
```

SofCheck

Conclusions



- It is fun to start from scratch now and then
- Can unify and simplify
- Can focus on new issues
  - pervasive parallelism
  - integrated annotations enforced at compile-time
- Read the blog if you are interested...

http://parasail-programming-language.blogspot.com

SofCheck

SofCheck™

SOFTWARE ANALYSIS AND VERIFICATION

11 Cypress Drive
Burlington, MA 01803-4907

Tucker Taft

tucker.taft@sofcheck.com

http://parasail-programming-language.blogspot.com

+1 (781) 750-8068 x220