Overview

1. What Is `polymake` About?
   - Design Goals
   - Ingredients

Overview

1. What Is `polymake` About?
   - Design Goals
   - Ingredients
2. Open Object Model
   - Applications, Objects and Properties

Overview

1. What Is `polymake` About?
   - Design Goals
   - Ingredients
2. Open Object Model
   - Applications, Objects and Properties
3. Rule Based Computation
   - Algorithms as Edges in a Directed Graph

Overview

1. What Is `polymake` About?
   - Design Goals
   - Ingredients
2. Open Object Model
   - Applications, Objects and Properties
3. Rule Based Computation
   - Algorithms as Edges in a Directed Graph
4. Conclusion

Design Goals

Three Golden Rules

1. Infinite extendibility
   - Allow to model new (mathematical) objects and integrate them seemlessly
2. Scalability with the user's ability to write programs
   - Include basic functionality for programming illiterates
   - Do not restrict programming experts
3. Do not re-invent the wheel!
   - Interfaces to existing code (in arbitrary language/design)

Some Ingredients

- Hybrid design: Perl (interpreted) and C++ (compiled)
  - Perl: Server side (= organization/communication)
  - C++: Client side (= computation)

Some Ingredients

- Hybrid design: Perl (interpreted) and C++ (compiled)
  - Perl: Server side (= organization/communication)
  - C++: Client side (= computation)
- Shell type user interface
  - (extension of) Perl as language

Some Ingredients

- Hybrid design: Perl (interpreted) and C++ (compiled)
  - Perl: Server side (= organization/communication)
  - C++: Client side (= computation)
- Shell type user interface
  - (extension of) Perl as language
- Technical features include:
  - C++ template library (extends STL, uses template meta-programming)
  - shared memory communication between client/server, transaction safe

Objects and Properties

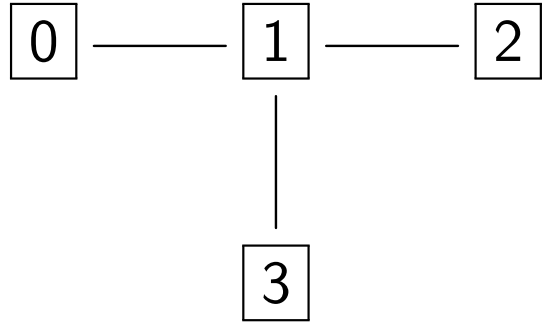
- hierarchy of big object types (modelling mathematical concepts)
  - e.g., polytopes, simplicial complexes, graphs, ...
  - under control of client/server system
  - with templates
- properties as class members (functions or data)
  - strongly typed
  - a type is a built-in Perl type, a C++ class type, or a big object type
  - immutable
- new big object types and properties to a given big object type can be added at will
- big object types grouped into applications ($\approx$ name spaces)

Graphs as Big Objects

```
declare object Graph<Dir=Undirected> {

  property ADJACENCY : props::Graph<Dir>;
  property N_NODES : Int;
  property NODE_DEGREES : Array<Int>;
  property CONNECTED : Bool;
  ...
}
```

Example:




```
$g=new Graph(ADJACENCY=>[[1],[0,2,3],[1],[1]]);
print $g->CONNECTED;
1
```

Rule Based Computation

- status of a big object defined by properties known/present
  - philosophy: object immutable
  - new properties computed = knowledge about object augmented
- rule produces new properties from known ones
  - source and target properties
  - preconditions
  - weights (vaguely reflect complexity)
  - labels
- server side scheduler computes shortest weighted path from sources to targets
  - rules *implicitly* define graph w/ subsets of properties as nodes
  - Dijkstra type algorithm

Anatomy of a Rule

```
rule CUBICAL : FACET_SIZES {
   my $cubical=1;
   foreach my $fs (@{$this->FACET_SIZES}) {
      $cubical=0, last if $fs != 8;
   }
   $this->CUBICAL=$cubical;
}
precondition : GRAPH.BIPARTITE, COMBINATORIAL_DIM {
  $this->GRAPH->BIPARTITE && $this->COMBINATORIAL_DIM==4
}
weight 1.10;
```

Conclusion and More

Applicability

- flexible design for projects which continuously evolve
- high abstraction level on the user's side

Other Features

- XML file format; XSLT for changes between revisions
- shared memory client/server communication
  - C++ exceptions properly translated into Perl
- C++ template library

▸ Client/Server Example

Wiki

`http://www.opt.tu-darmstadt.de/polymake`

Client/Server Communication: Perl Side

```
rule VOLUME : VERTICES, TRIANGULATION.FACETS {
   volume($this, $this->VERTICES,
                 $this->TRIANGULATION->FACETS);
}
precondition : BOUNDED;
precondition : FULL_DIM;
weight 2.30;
```

◂ Conclusion

Client/Server Communication: C++ Side

```
template <typename MatrixTop , typename Triangulation>
void volume(perl::Object p, const GenericMatrix<MatrixTop>& Points
                            const Triangulation& tr)
{
   typedef typename MatrixTop::element_type Coord;
   Coord volume(0); int d=tr.front().size()-1;

   for (typename Entire<Triangulation>::const_iterator s=entire(tr
      const typename MatrixTop::persistent_type sim=Points.minor(*
      Coord v=abs(det(sim));
      volume += v;
   }
   volume /= Integer::fac(d);
   p.take("VOLUME") << volume;
}

FunctionTemplate4perl(
  "volume(Polytope Matrix Array< Set<Int> >) : void"
);
```

◂ Conclusion