Title: Tutorial 3A: Bootstrapping GNY model

Speakers: Aike Liu

Collection: Mini-Course of Numerical Conformal Bootstrap

Date: April 26, 2023 - 1:30 PM

URL: https://pirsa.org/23040143

Hyperion Tutorial 2

Bootstrap Fermions & GNY

Bootstrap Mini Course: Tutorial 3a

Aike Liu

California Institute of Technology

Caltech PI PERIMETER INSTITUTE

SIMONS FOUNDATION

`scalars-3d/fermions-3d`

`Bound.hs`

Crossing equations

OPE coefficients

Set up SDP

`Project.hs`

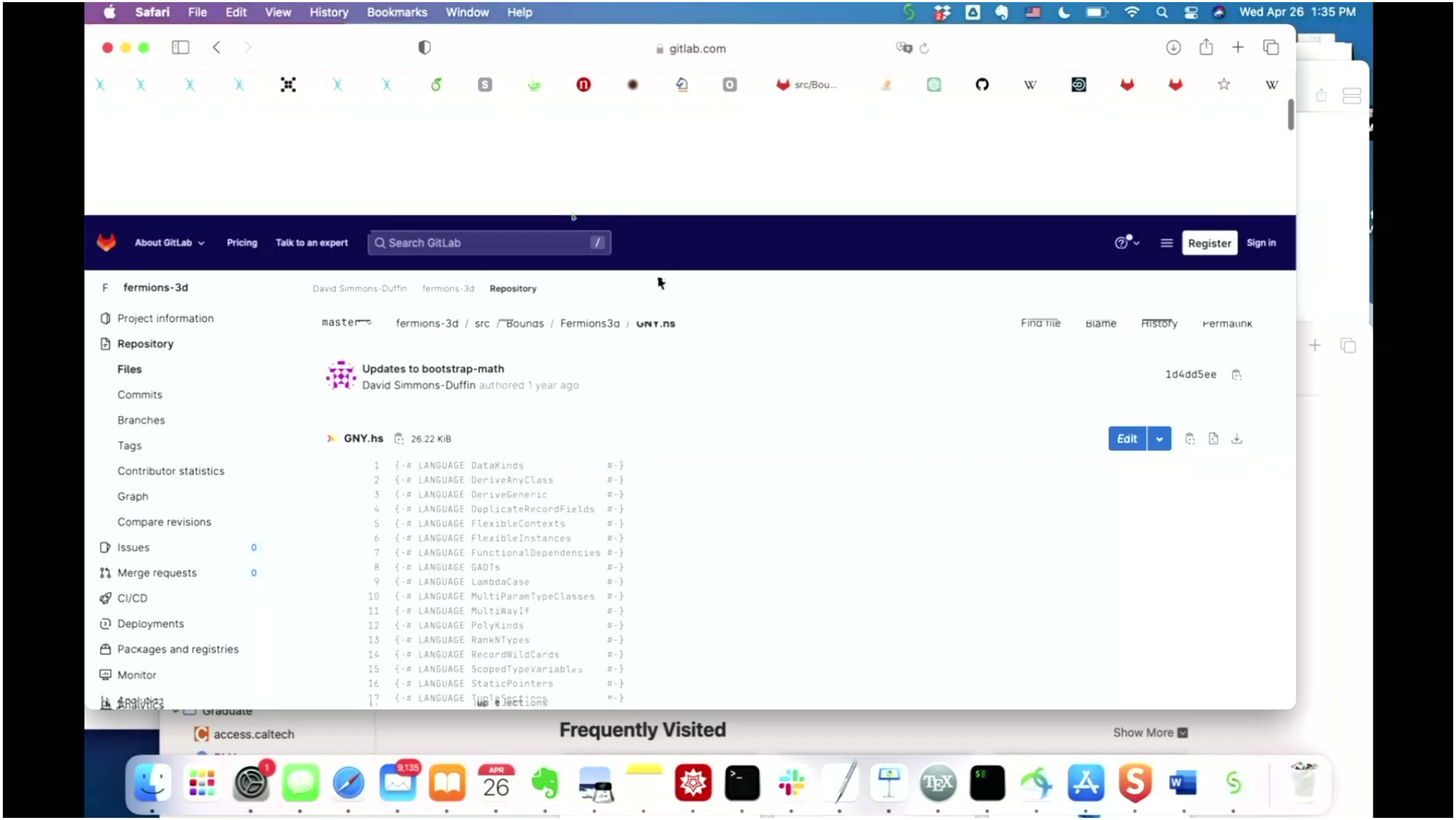
Impose gaps, set up OPE search

Numerical parameters

such as Λ, κ (pole order), precisions and SDPB parameters

submit HPC remote job

- Four fermions bootstrap <ψψψψ>
  - Crossing Equations
  - OPE channels in SO(3) basis
  - 3d Block types
- GNY bootstrap mixing (σ, ψ) with O(N) symmetry
  - O(N) representation
  - Crossing Equations with global symmetry blocks


Safari File Edit View History Bookmarks Window Help
Wed Apr 26 1:35 PM
gitlab.com
About GitLab
Pricing
Talk to an expert
Search GitLab
Register
Sign in
fermions-3d
David Simmons-Duffin / fermions-3d / Repository
Project information
Repository
Files
Commits
Branches
Tags
Contributor statistics
Graph
Compare revisions
Issues 0
Merge requests 0
CI/CD
Deployments
Packages and registries
Monitor
master fermions-3d / src / Bounds / Fermions3d / GNY.ns
Find file
Blame
History
Permalink
Updates to bootstrap-math
David Simmons-Duffin authored 1 year ago
1d4dd5ee
GNY.hs 26.22 KiB
Edit
{-# LANGUAGE DataKinds #-}
{-# LANGUAGE DeriveAnyClass #-}
{-# LANGUAGE DeriveGeneric #-}
{-# LANGUAGE DuplicateRecordFields #-}
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE FlexibleInstances #-}
{-# LANGUAGE FunctionalDependencies #-}
{-# LANGUAGE GADTs #-}
{-# LANGUAGE LambdaCase #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE MultiWayIf #-}
{-# LANGUAGE PolyKinds #-}
{-# LANGUAGE RankNTypes #-}
{-# LANGUAGE RecordWildCards #-}
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE StaticPointers #-}
{-# LANGUAGE TupleSections #-}
Graduate
access.caltech
Frequently Visited
Show More

- Four fermions bootstrap $\langle\psi\psi\psi\psi\rangle$
  - Crossing Equations
  - OPE channels in SO(3) basis
  - 3d Block types
- GNY bootstrap mixing (σ, ψ) with O(N) symmetry
  - O(N) representation
  - Crossing Equations with global symmetry blocks

$$\lambda^{a}_{\psi\psi\mathcal{O}} \qquad \langle\psi\psi|\psi\psi\rangle = -\langle\psi\psi|\psi\psi\rangle$$

$$[\tfrac{1}{2},-\tfrac{1}{2},0]^{\pm} \quad [\tfrac{1}{2},\tfrac{1}{2},-1]^{\pm}$$

$$\langle\uparrow\uparrow\uparrow\uparrow\rangle^{+}, \langle\uparrow\uparrow\uparrow\uparrow\rangle^{-}, \langle\uparrow\uparrow\downarrow\downarrow\rangle^{+}, \langle\uparrow\downarrow\uparrow\downarrow\rangle^{+}, \langle\downarrow\uparrow\uparrow\downarrow\rangle^{+}$$

$$\begin{aligned}
\partial_{\mathbf{y}}^{2n}\partial_{x}^{2m+1} g_{\langle\uparrow\uparrow\uparrow\uparrow\rangle^{+}} &= 0, \quad n \geq 1, m \geq 0, \\
\partial_{\mathbf{y}}^{2n}\partial_{x}^{2m+1} g_{\langle\uparrow\downarrow\uparrow\downarrow\rangle^{+}} &= 0, \quad n \geq 0, m \geq 0, \\
\partial_{\mathbf{y}}^{2n}\partial_{x}^{2m+1} \left(g_{\langle\uparrow\uparrow\downarrow\downarrow\rangle^{+}} + g_{\langle\downarrow\uparrow\uparrow\downarrow\rangle^{+}}\right) &= 0, \quad n \geq 0, m \geq 0, \\
\partial_{\mathbf{y}}^{2n}\partial_{x}^{2m} \left(g_{\langle\uparrow\uparrow\downarrow\downarrow\rangle^{+}} - g_{\langle\downarrow\uparrow\uparrow\downarrow\rangle^{+}}\right) &= 0, \quad n \geq 0, m \geq 0, \\
\partial_{\mathbf{y}}^{2n+1}\partial_{x}^{2m} g_{\langle\uparrow\uparrow\uparrow\uparrow\rangle^{-}} &= 0, \quad n \geq 0, m \geq 0,
\end{aligned}$$

External operators

<ψψψψ> so `ExternalOp` contains one operator only.

No need for `ExternalDims`

```
data ExternalOp s = Psi
  deriving (Show, Eq, Ord, Enum, Bounded)

instance (Reifies s Rational) => HasRep (ExternalOp s) (B3d.ConformalRep Rational) where
  rep psi = B3d.ConformalRep (reflect psi) (1/2)
```

```
data ConformalRep a = ConformalRep
  { delta :: a
  , spin  :: HalfInteger
  } deriving (Eq, Ord, Show, Generic, Binary, F
```

```
data FourFermions3d = FourFermions3d
  { deltaPsi    :: Rational
  , spectrum    :: Spectrum (Int, B3d.Parity)
  , objective   :: Objective
  , spins       :: [Int]
  , blockParams :: B3d.Block3dParams
  } deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON)

data Objective
  = Feasibility
  | StressTensorOPEBound BoundDirection
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON)
```

Crossing Equations

<> is the infix operator defined in the **Semigroup** type class.

```
1 <> 1 = 1
1 <> 0 = 0
0 <> 1 = 0
0 <> 0 = 1
```

```
crossingEqsWithFlavorSign
  :: forall s a b . (Ord b, Fractional a, Eq a)
  => Sign 'XDir
  -> FourPointFunctionTerm (ExternalOp s) B3d.Q4Struct b a
  -> V 5 (Taylors 'XT, FreeVect b a)
crossingEqsWithFlavorSign fSign g0 = toV
  ( (V.filter positiveN .
     xtTaylors (fSign<>xOdd)  yEven, g (u,u,u,u) yEven)
  , (xtTaylors (fSign<>xOdd)  yEven, g (u,d,u,d) yEven)
  , (xtTaylors (fSign<>xOdd)  yEven, g (u,u,d,d) yEven + g (d,u,u,d) yEven)
  , (xtTaylors (fSign<>xEven) yEven, g (u,u,d,d) yEven - g (d,u,u,d) yEven)
  , (xtTaylors (fSign<>xEven) yOdd,  g (u,u,u,u) yOdd)
  )
  where
    u = 1/2
    d = -1/2
    g qs sgn = g0 Psi Psi Psi Psi (B3d.Q4Struct qs sgn)
    positiveN (TaylorCoeff (Derivative (_,n))) = n > 0
```

Crossing Equations

<> is the infix operator defined in the **Semigroup** type class.

```
1 <> 1 = 1
1 <> 0 = 0
0 <> 1 = 0
0 <> 0 = 1
```

```
crossingEqsWithFlavorSign
  :: forall s a b . (Ord b, Fractional a, Eq a)
  => Sign 'XDir
  -> FourPointFunctionTerm (ExternalOp s) B3d.Q4Struct b a
  -> V 5 (Taylors 'XT, FreeVect b a)
crossingEqsWithFlavorSign fSign g0 = toV
  ( (V.filter positiveN .
     xtTaylors (fSign<>xOdd)  yEven, g (u,u,u,u) yEven)
  , (xtTaylors (fSign<>xOdd)  yEven, g (u,d,u,d) yEven)
  , (xtTaylors (fSign<>xOdd)  yEven, g (u,u,d,d) yEven + g (d,u,u,d) yEven)
  , (xtTaylors (fSign<>xEven) yEven, g (u,u,d,d) yEven - g (d,u,u,d) yEven)
  , (xtTaylors (fSign<>xEven) yOdd,  g (u,u,u,u) yOdd)
  )
  where
    u = 1/2
    d = -1/2
    g qs sgn = g0 Psi Psi Psi Psi (B3d.Q4Struct qs sgn)
    positiveN (TaylorCoeff (Derivative (_,n))) = n > 0
```

$$\langle\uparrow\uparrow\uparrow\uparrow\rangle^+, \langle\uparrow\uparrow\uparrow\uparrow\rangle^-, \langle\uparrow\uparrow\downarrow\downarrow\rangle^+, \langle\uparrow\downarrow\uparrow\downarrow\rangle^+, \langle\downarrow\uparrow\uparrow\downarrow\rangle^+$$

$$\begin{aligned}
\partial_{\mathbf{y}}^{2n}\partial_x^{2m+1} g_{\langle\uparrow\uparrow\uparrow\uparrow\rangle^+} &= 0, \quad n \geq 1, m \geq 0, \\
\partial_{\mathbf{y}}^{2n}\partial_x^{2m+1} g_{\langle\uparrow\downarrow\uparrow\downarrow\rangle^+} &= 0, \quad n \geq 0, m \geq 0, \\
\partial_{\mathbf{y}}^{2n}\partial_x^{2m+1} \left(g_{\langle\uparrow\uparrow\downarrow\downarrow\rangle^+} + g_{\langle\downarrow\uparrow\uparrow\downarrow\rangle^+}\right) &= 0, \quad n \geq 0, m \geq 0, \\
\partial_{\mathbf{y}}^{2n}\partial_x^{2m} \left(g_{\langle\uparrow\uparrow\downarrow\downarrow\rangle^+} - g_{\langle\downarrow\uparrow\uparrow\downarrow\rangle^+}\right) &= 0, \quad n \geq 0, m \geq 0, \\
\partial_{\mathbf{y}}^{2n+1}\partial_x^{2m} g_{\langle\uparrow\uparrow\uparrow\uparrow\rangle^-} &= 0, \quad n \geq 0, m \geq 0,
\end{aligned}$$

Crossing Equations

<> is the infix operator defined in the **Semigroup** type class.

```
1 <> 1 = 1
1 <> 0 = 0
0 <> 1 = 0
0 <> 0 = 1
```

```
crossingEqsWithFlavorSign
  :: forall s a b . (Ord b, Fractional a, Eq a)
  => Sign 'XDir
  -> FourPointFunctionTerm (ExternalOp s) B3d.Q4Struct b a
  -> V 5 (Taylors 'XT, FreeVect b a)
crossingEqsWithFlavorSign fSign g0 = toV
  ( (V.filter positiveN .
     xtTaylors (fSign<>xOdd)  yEven, g (u,u,u,u) yEven)
  , (xtTaylors (fSign<>xOdd)  yEven, g (u,d,u,d) yEven)
  , (xtTaylors (fSign<>xOdd)  yEven, g (u,u,d,d) yEven + g (d,u,u,d) yEven)
  , (xtTaylors (fSign<>xEven) yEven, g (u,u,d,d) yEven - g (d,u,u,d) yEven)
  , (xtTaylors (fSign<>xEven) yOdd,  g (u,u,u,u) yOdd)
  )
  where
    u = 1/2
    d = -1/2
    g qs sgn = g0 Psi Psi Psi Psi (B3d.Q4Struct qs sgn)
    positiveN (TaylorCoeff (Derivative (_,n))) = n > 0
```

Crossing Matrix

```
crossingMatFF
  :: forall j s a. (KnownNat j, Fractional a, Eq a)
  => V j (OPECoefficient (ExternalOp s) (B3d.SO3Struct Rational Rational Delta) a)
  -> Tagged s (CrossingMat j 5 B3d.Block3d a)
crossingMatFF channel =
  pure $ mapBlocks B3d.Block3d $
  crossingMatrix channel (crossingEqs @s)
```

OPE channels

OPE	$\mathcal{O} \in (l, P, \mu)$
$\psi \times \psi$	$(l \in 2\mathbb{Z}$, even
	$(l \in 2\mathbb{Z}$, odd,
	$(l \in 2\mathbb{Z}+1$, odd,

```
data Channel j b where
  ParityEven_SpinEven :: B3d.ConformalRep Delta -> Channel 2 B3d.Block3d
  -- | When l=0, only one of the parity even spin even structures is
  -- allowed
  ParityEven_Scalar   :: Delta                  -> Channel 1 B3d.Block3d
  ParityOdd_SpinEven  :: B3d.ConformalRep Delta -> Channel 1 B3d.Block3d
  ParityOdd_SpinOdd   :: B3d.ConformalRep Delta -> Channel 1 B3d.Block3d
  IdentityChannel     :: Channel 1 B3d.IdentityBlock3d
  StressTensorChannel :: Channel 1 B3d.Block3d
```

This is a data type containing **6 constructors**. For general channels, we have three channels depending on parity and spins. Notice that we have one special case on **Parity_Even** *spin = 0* operator.

Unlike the channels in Ising Bootstrap where the only type parameter is **j**, the channel of spinning bootstrap has two phantom type variables **j** and **b**. This is because identity block is different than general blocks in `Blocks_3d`.

```haskell
mat (ParityEven_SpinEven internalRep) = crossingMatFF $
  fmap (opeCoeffIdentical_ Psi internalRep) $
  -- | These structures are related by an l-dependent linear
  -- transformation to r1, r2 from [1]
  toV (so3 0 l, so3 1 l)
  where
    l = B3d.spin internalRep
```

OPE	$\mathcal{O} \in (l, P, \mu)$	$\langle \psi^i \psi^j \mathcal{O}^a \rangle$ Structures
$\psi \times \psi$	$l \in 2\mathbb{Z}$, even	$\|0, l\rangle$
		$\|1, l\rangle$
	$l \in 2\mathbb{Z}$, odd,	$(\sqrt{l+1}\|1, l+1\rangle - \sqrt{l}\|1, l-1\rangle)$
	$l \in 2\mathbb{Z}+1$, odd.	$(\sqrt{l+1}\|1, l-1\rangle + \sqrt{l}\|1, l+1\rangle)$

```haskell
mat (ParityOdd_SpinOdd internalRep) = crossingMatFF $
  fmap (opeCoeffIdentical_ Psi internalRep) $ toV r4
  where
    l = B3d.spin internalRep
    -- | This is related by an l-dependent coefficient to r4 from [1]
    r4 = sqrt (realToFrac l + 1) *^ so3 1 (l-1)
       + sqrt (realToFrac l)   *^ so3 1 (l+1)
```

OPE channels

OPE	$\mathcal{O} \in (l, P, \mu)$
$\psi \times \psi$	$(l \in 2\mathbb{Z}$, even
	$(l \in 2\mathbb{Z}$, odd,
	$(l \in 2\mathbb{Z}+1$, odd,

```
data Channel j b where
  ParityEven_SpinEven :: B3d.ConformalRep Delta -> Channel 2 B3d.Block3d
  -- | When l=0, only one of the parity even spin even structures is
  -- allowed
  ParityEven_Scalar   :: Delta                  -> Channel 1 B3d.Block3d
  ParityOdd_SpinEven  :: B3d.ConformalRep Delta -> Channel 1 B3d.Block3d
  ParityOdd_SpinOdd   :: B3d.ConformalRep Delta -> Channel 1 B3d.Block3d
  IdentityChannel     :: Channel 1 B3d.IdentityBlock3d
  StressTensorChannel :: Channel 1 B3d.Block3d
```

This is a data type containing **6 constructors**. For general channels, we have three channels depending on parity and spins. Notice that we have one special case on **Parity_Even** *spin = 0* operator.

Unlike the channels in Ising Bootstrap where the only type parameter is **j**, the channel of spinning bootstrap has two phantom type variables **j** and **b**. This is because identity block is different than general blocks in `Blocks_3d`.

OPE channels

```haskell
data Channel j b where
  ParityEven_SpinEven :: B3d.ConformalRep Delta -> Channel 2 B3d.Block3d
  -- | When l=0, only one of the parity even spin even structures is
  -- allowed
  ParityEven_Scalar   :: Delta                  -> Channel 1 B3d.Block3d
  ParityOdd_SpinEven  :: B3d.ConformalRep Delta -> Channel 1 B3d.Block3d
  ParityOdd_SpinOdd   :: B3d.ConformalRep Delta -> Channel 1 B3d.Block3d
  IdentityChannel     :: Channel 1 B3d.IdentityBlock3d
  StressTensorChannel :: Channel 1 B3d.Block3d
```

```haskell
so3 :: (Num a, Eq a) => HalfInteger -> HalfInteger -> FreeVect B3d.SO3StructLabel a
so3 j12 j123 = vec (B3d.SO3StructLabel j12 j123)
```

this is a singleton vector

```haskell
mat
  :: forall s a j b . (Reifies s Rational, Floating a, Eq a)
  => Channel j b
  -> Tagged s (CrossingMat j 5 b a)
```

```haskell
mat (ParityEven_SpinEven internalRep) = crossingMatFF $
  fmap (opeCoeffIdentical_ Psi internalRep) $
  -- | These structures are related by an l-dependent linear
  -- transformation to r1, r2 from [1]
  toV (so3 0 l, so3 1 l)
  where
    l = B3d.spin internalRep
```

OPE	$\mathcal{O} \in (l, P, \mu)$	$\langle \psi^i \psi^j \mathcal{O}^a \rangle$ Structures
$\psi \times \psi$	$l \in 2\mathbb{Z}$, even	$\|0, l\rangle$
		$\|1, l\rangle$
	$l \in 2\mathbb{Z}$, odd,	$(\sqrt{l+1}\|1, l+1\rangle - \sqrt{l}\|1, l-1\rangle)$
	$l \in 2\mathbb{Z}+1$, odd.	$(\sqrt{l+1}\|1, l-1\rangle + \sqrt{l}\|1, l+1\rangle)$

```haskell
mat (ParityOdd_SpinOdd internalRep) = crossingMatFF $
  fmap (opeCoeffIdentical_ Psi internalRep) $ toV r4
  where
    l = B3d.spin internalRep
    -- | This is related by an l-dependent coefficient to r4 from [1]
    r4 = sqrt (realToFrac l + 1) *^ so3 1 (l-1)
       + sqrt (realToFrac l)   *^ so3 1 (l+1)
```

```
mat IdentityChannel =
  pure $ mapBlocks B3d.IdentityBlock3d $
  crossingMatrix (toV identityOpe) (crossingEqs @s)
  where
    identityOpe o1 o2
      | o1 == o2  = vec (rep @(ExternalOp s) o1)
      | otherwise = 0
```

```haskell
-- | Data needed for "bulk" positivity conditions not associated with
-- external operators, the identity or the stress tensor.
data BulkConstraint where
  BulkConstraint
    :: KnownNat j -- ^ Existentially quantify over j
    => DeltaRange -- ^ Isolated or Continuum constraint
    -> Channel j B3d.Block3d -- ^ The associated Channel
    -> BulkConstraint


bulkConstraints :: FourFermions3d -> [BulkConstraint]
bulkConstraints f = do
  parity <- [minBound .. maxBound]
  l <- case parity of
    B3d.ParityEven -> filter even (spins f)
    B3d.ParityOdd  -> spins f
  (delta, range) <- listDeltas (l, parity) (spectrum f)
  let internalRep = B3d.ConformalRep delta (fromIntegral l)
  pure $ case (parity, even l, l) of
    (B3d.ParityEven, True, 0)  -> BulkConstraint range $ ParityEven_Scalar delta
    (B3d.ParityEven, True, _)  -> BulkConstraint range $ ParityEven_SpinEven internalRep
    (B3d.ParityOdd,  True, _)  -> BulkConstraint range $ ParityOdd_SpinEven internalRep
    (B3d.ParityOdd,  False, _) -> BulkConstraint range $ ParityOdd_SpinOdd internalRep
    _ -> error "Internal representation disallowed"
```

This code is restructured a little compared with IsingSigEps.hs

The bootstrapConstraint function is moved to **ffSDP**

```
mat IdentityChannel =
  pure $ mapBlocks B3d.IdentityBlock3d $
  crossingMatrix (toV identityOpe) (crossingEqs @s)
  where
    identityOpe o1 o2
      | o1 == o2  = vec (rep @(ExternalOp s) o1)
      | otherwise = 0
```

```haskell
ffSDP
  :: forall a m.
     ( Binary a, Typeable a, RealFloat a, Applicative m
     , BlockFetchContext B3d.Block3d a m
     )
  => FourFermions3d
  -> SDPB.SDP m a
ffSDP f@FourFermions3d{..} = runTagged deltaPsi $ do
  let dv = derivsVecFF f
  bulk <- for (bulkConstraints f) $
    \(BulkConstraint range c) ->
      BSDP.bootstrapConstraint blockParams dv range <$> mat c
  unit <- mat IdentityChannel
  stress <- mat StressTensorChannel
  let
    stressConstraint = BSDP.isolatedConstraint blockParams dv stress
    (cons, obj, norm) = case objective of
      Feasibility ->
        ( stressConstraint : bulk
        , BSDP.bootstrapObjective blockParams dv $ zero `asTypeOf` unit
        , BSDP.bootstrapNormalization blockParams dv $ unit
        )
      StressTensorOPEBound dir ->
        ( bulk
        , BSDP.bootstrapObjective blockParams dv unit
        , BSDP.bootstrapNormalization blockParams dv $ boundDirSign dir *^ stress
        )
  return $ SDPB.SDP
    { SDPB.objective     = obj
    , SDPB.normalization = norm
    , SDPB.positiveConstraints      = cons
    }
```

Compare to isingSigEpsSDP, this function replace some **do** notation with **let**

```
instance ToSDP FourFermions3d where
  type SDPFetchKeys FourFermions3d = '[ B3d.BlockTableKey ]
  toSDP = ffSDP

instance SDPFetchBuildConfig FourFermions3d where
  sdpFetchConfig _ _ boundFiles =
    liftIO . B3d.readBlockTable (blockDir boundFiles) :&: FetchNil
  sdpDepBuildChain _ bConfig boundFiles =
    SomeBuildChain $ noDeps $ block3dBuildLink bConfig boundFiles

instance Static (Binary FourFermions3d)              where closureDict = cPtr (static Dict)
instance Static (Show FourFermions3d)                where closureDict = cPtr (static Dict)
instance Static (ToSDP FourFermions3d)               where closureDict = cPtr (static Dict)
instance Static (ToJSON FourFermions3d)              where closureDict = cPtr (static Dict)
instance Static (SDPFetchBuildConfig FourFermions3d) where closureDict = cPtr (static Dict)
instance Static (BuildInJob FourFermions3d)          where closureDict = cPtr (static Dict)
```

```haskell
ffSDP
  :: forall a m.
     ( Binary a, Typeable a, RealFloat a, Applicative m
     , BlockFetchContext B3d.Block3d a m
     )
  => FourFermions3d
  -> SDPB.SDP m a
ffSDP f@FourFermions3d{..} = runTagged deltaPsi $ do
  let dv = derivsVecFF f
  bulk <- for (bulkConstraints f) $
    \(BulkConstraint range c) ->
      BSDP.bootstrapConstraint blockParams dv range <$> mat c
  unit <- mat IdentityChannel
  stress <- mat StressTensorChannel
  let
    stressConstraint = BSDP.isolatedConstraint blockParams dv stress
    (cons, obj, norm) = case objective of
      Feasibility ->
        ( stressConstraint : bulk
        , BSDP.bootstrapObjective blockParams dv $ zero `asTypeOf` unit
        , BSDP.bootstrapNormalization blockParams dv $ unit
        )
      StressTensorOPEBound dir ->
        ( bulk
        , BSDP.bootstrapObjective blockParams dv unit
        , BSDP.bootstrapNormalization blockParams dv $ boundDirSign dir *^ stress
        )
  return $ SDPB.SDP
    { SDPB.objective     = obj
    , SDPB.normalization = norm
    , SDPB.positiveConstraints      = cons
    }
```

Compare to isingSigEpsSDP, this function replace some **do** notation with **let**

```haskell
ffSDP
  :: forall a m.
     ( Binary a, Typeable a, RealFloat a, Applicative m
     , BlockFetchContext B3d.Block3d a m
     )
  => FourFermions3d
  -> SDPB.SDP m a
ffSDP f@FourFermions3d{..} = runTagged deltaPsi $ do
  let dv = derivsVecFF f
  bulk <- for (bulkConstraints f) $
    \(BulkConstraint range c) ->
      BSDP.bootstrapConstraint blockParams dv range <$> mat c
  unit <- mat IdentityChannel
  stress <- mat StressTensorChannel
  let
    stressConstraint = BSDP.isolatedConstraint blockParams dv stress
    (cons, obj, norm) = case objective of
      Feasibility ->
        ( stressConstraint : bulk
        , BSDP.bootstrapObjective blockParams dv $ zero `asTypeOf` unit
        , BSDP.bootstrapNormalization blockParams dv $ unit
        )
      StressTensorOPEBound dir ->
        ( bulk
        , BSDP.bootstrapObjective blockParams dv unit
        , BSDP.bootstrapNormalization blockParams dv $ boundDirSign dir *^ stress
        )
  return $ SDPB.SDP
    { SDPB.objective     = obj
    , SDPB.normalization = norm
    , SDPB.positiveConstraints      = cons
    }
```

Compare to isingSigEpsSDP, this function replace some **do** notation with **let**

GNY (σ, ψ) with O(N)

1. Define data types for the external operators
2. Define data types for global symmetry representations
3. Define your computational problem (physical model and constraints) and goals (test feasible solutions or bound OPEs)
4. Write down crossing equations
5. Define OPE channels and three-point structures
6. Convert the bootstrap problem into an SDP

$$\sigma: \quad (0, \text{odd}, \bullet)$$
$$\psi_i: \quad (\tfrac{1}{2}, \text{even}, \square).$$

$$\left.\begin{array}{l} \{\lambda^a_{\sigma\sigma\mathcal{O}},\ \lambda^a_{\sigma\psi\mathcal{O}},\ \lambda^a_{\psi\psi\mathcal{O}}\} \\ \\ \langle\sigma\sigma|\psi\psi\rangle = \langle\psi\sigma|\sigma\psi\rangle, \\ \langle\psi\sigma|\psi\sigma\rangle = -\langle\psi\sigma|\psi\sigma\rangle \\ \langle\psi\psi|\psi\psi\rangle = -\langle\psi\psi|\psi\psi\rangle \end{array}\right\} \quad \begin{array}{l} T^{ijk}_{a_{\text{conf}}, a_{\text{flavor}}} = t^{ijk}_{a_{\text{conf}}} Q^{ijk}_{a_{\text{flavor}}}, \\ T^{ijkl}_{I_{\text{conf}}, a_{\text{flavor}}} = t^{ijkl}_{I_{\text{conf}}} Q^{ijkl}_{I_{\text{flavor}}}. \end{array}$$

src/Bounds/Scalars3d/ONRep.hs

```
import  Bounds.Scalars3d.ONRep  (ON3PtStruct (..),
                                 ON4PtStruct (..),
                                 ONRep (..),
                                 evalONBlock)
```

O(N) representations of a single operator

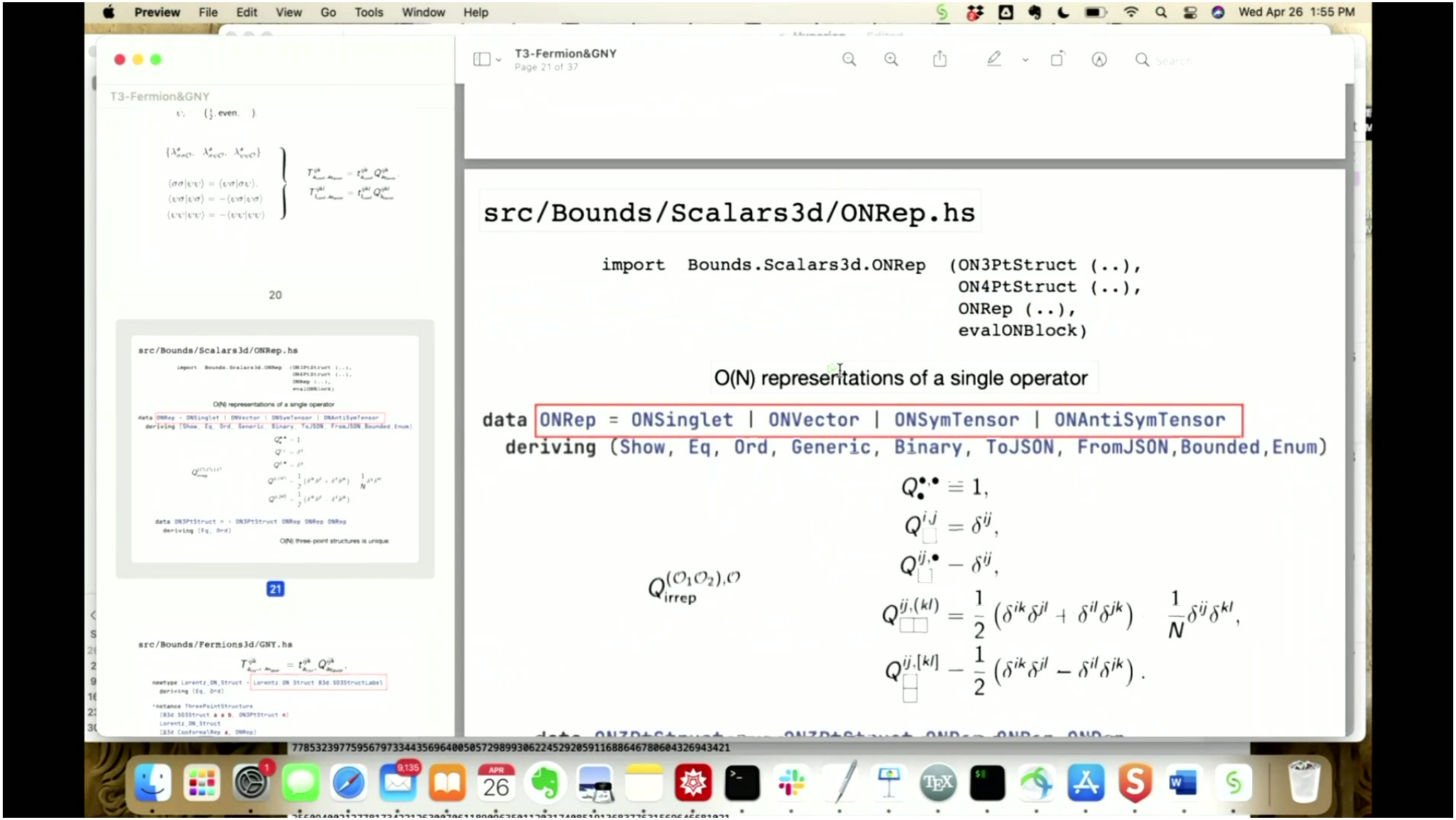
```
data ONRep = ONSinglet | ONVector | ONSymTensor | ONAntiSymTensor
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON,Bounded,Enum)
```

$$Q_{\irrep}^{(\mathcal{O}_1\mathcal{O}_2),\mathcal{O}}$$

$$Q_{\bullet}^{\bullet,\bullet} = 1,$$
$$Q_{\square}^{i,j} = \delta^{ij},$$
$$Q_{\square}^{ij,\bullet} = \delta^{ij},$$
$$Q_{\square\square}^{ij,(kl)} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} + \delta^{il}\delta^{jk}\right) - \frac{1}{N}\delta^{ij}\delta^{kl},$$
$$Q_{\begin{smallmatrix}\square\\\square\end{smallmatrix}}^{ij,[kl]} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} - \delta^{il}\delta^{jk}\right).$$

```
data ON3PtStruct n = ON3PtStruct ONRep ONRep ONRep
  deriving (Eq, Ord)
```

O(N) three-point structures is unique.



src/Bounds/Scalars3d/ONRep.hs

```
import  Bounds.Scalars3d.ONRep  (ON3PtStruct (..),
                                 ON4PtStruct (..),
                                 ONRep (..),
                                 evalONBlock)
```

O(N) representations of a single operator

```
data ONRep = ONSinglet | ONVector | ONSymTensor | ONAntiSymTensor
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON,Bounded,Enum)
```

$$Q^{(\mathcal{O}_1\mathcal{O}_2),\mathcal{O}}_{\text{irrep}}$$

$$Q^{\bullet,\bullet}_{\bullet} = 1,$$
$$Q^{i,j}_{\square} = \delta^{ij},$$
$$Q^{ij,\bullet}_{\square} - \delta^{ij},$$
$$Q^{ij,(kl)}_{\square\square} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} + \delta^{il}\delta^{jk}\right) \quad \frac{1}{N}\delta^{ij}\delta^{kl},$$
$$Q^{ij,[kl]}_{\begin{smallmatrix}\square\\\square\end{smallmatrix}} - \frac{1}{2}\left(\delta^{ik}\delta^{jl} - \delta^{il}\delta^{jk}\right).$$

O(N) representations of a single operator

```
data ONRep = ONSinglet | ONVector | ONSymTensor | ONAntiSymTensor
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON,Bounded,Enum)
```

$$Q^{(\mathcal{O}_1\mathcal{O}_2),\mathcal{O}}_{\text{irrep}}$$

$$Q^{\bullet,\bullet}_{\bullet} = 1,$$
$$Q^{i,j}_{\square} = \delta^{ij},$$
$$Q^{ij,\bullet}_{\square} = \delta^{ij},$$
$$Q^{ij,(kl)}_{\square\square} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} + \delta^{il}\delta^{jk}\right) - \frac{1}{N}\delta^{ij}\delta^{kl},$$
$$Q^{ij,[kl]}_{\begin{smallmatrix}\square\\\square\end{smallmatrix}} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} - \delta^{il}\delta^{jk}\right).$$

```
data ON3PtStruct n = ON3PtStruct ONRep ONRep ONRep
  deriving (Eq, Ord)
```

O(N) three-point structures is unique.

src/Bounds/Fermions3d/GNY.hs

$$T^{ijk}_{a_{\text{conf}},a_{\text{flavor}}} = t^{ijk}_{a_{\text{conf}}} Q^{ijk}_{a_{\text{flavor}}},$$

src/Bounds/Scalars3d/ONRep.hs

```
import  Bounds.Scalars3d.ONRep  (ON3PtStruct (..),
                                 ON4PtStruct (..),
                                 ONRep (..),
                                 evalONBlock)
```

O(N) representations of a single operator

```
data ONRep = ONSinglet | ONVector | ONSymTensor | ONAntiSymTensor
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON,Bounded,Enum)
```

$$Q_{\text{irrep}}^{(\mathcal{O}_1\mathcal{O}_2),\mathcal{O}}$$

$$Q_{\bullet}^{\bullet,\bullet} = 1,$$
$$Q_{\square}^{i,j} = \delta^{ij},$$
$$Q_{\square}^{ij,\bullet} = \delta^{ij},$$
$$Q_{\square\square}^{ij,(kl)} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} + \delta^{il}\delta^{jk}\right) - \frac{1}{N}\delta^{ij}\delta^{kl},$$
$$Q_{\begin{smallmatrix}\square\\\square\end{smallmatrix}}^{ij,[kl]} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} - \delta^{il}\delta^{jk}\right).$$

```
data ON3PtStruct n = ON3PtStruct ONRep ONRep ONRep
  deriving (Eq, Ord)
```

O(N) three-point structures is unique.

src/Bounds/Scalars3d/ONRep.hs

Evaluate O(N) flavor blocks, the result is a number

```
evalONBlock :: (Reifies n Rational, Fractional a, Eq a) => Block (ON3PtStruct n) (ON4PtStruct n) -> a
evalONBlock (Block (ON3PtStruct r1 r2 r) (ON3PtStruct r4 r3 r') f)
  | r /= r' = 0
  | otherwise = case ((r1, r2, r3, r4), r, f) of
      ((ONVector, ONVector, ONVector, ONVector), _, _) -> FV.coeff f (onVectorBlock r)
      ((ONSinglet, ONSinglet, ONSinglet, ONSinglet), ONSinglet, Unique) -> 1
      ((ONVector,  ONVector,  ONSinglet, ONSinglet), ONSinglet, Unique) -> 1
      ((ONSinglet, ONSinglet, ONVector,  ONVector),  ONSinglet, Unique) -> 1
      ((ONVector,  ONSinglet, ONVector,  ONSinglet), ONVector,  Unique) -> 1
```

Coefficient of f, a ON4PtStruct, in the vector (onVectorBlock r)

```
onVectorBlock
  :: forall n a . (Reifies n Rational, Fractional a, Eq a)
  => ONRep
  -> FreeVect (ON4PtStruct n) a
onVectorBlock r = case r of
  ONSinglet       -> v12_34
  ONSymTensor     -> 1/2 *^ (v13_24 + v23_14) - 1/nGroup *^ v12_34
  ONAntiSymTensor -> 1/2 *^ (v23_14 - v13_24)
  _               -> 0
  where
```

ON4PtStruct's are invariant under crossing

ONRep's are representations of operators

src/Bounds/Scalars3d/ONRep.hs

```
import  Bounds.Scalars3d.ONRep  (ON3PtStruct (..),
                                 ON4PtStruct (..),
                                 ONRep (..),
                                 evalONBlock)
```

O(N) representations of a single operator

```
data ONRep = ONSinglet | ONVector | ONSymTensor | ONAntiSymTensor
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON,Bounded,Enum)
```

$$Q_{\text{irrep}}^{(\mathcal{O}_1\mathcal{O}_2),\mathcal{O}}$$

$$Q_{\bullet}^{\bullet,\bullet} = 1,$$

$$Q_{\square}^{i,j} = \delta^{ij},$$

$$Q_{\square}^{ij,\bullet} = \delta^{ij},$$

$$Q_{\square\square}^{ij,(kl)} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} + \delta^{il}\delta^{jk}\right) - \frac{1}{N}\delta^{ij}\delta^{kl},$$

$$Q_{\begin{smallmatrix}\square\\\square\end{smallmatrix}}^{ij,[kl]} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} - \delta^{il}\delta^{jk}\right).$$

```
data ON3PtStruct n = ON3PtStruct ONRep ONRep ONRep
  deriving (Eq, Ord)
```

O(N) three-point structures is unique.

src/Bounds/Fermions3d/GNY.hs

$$T^{ijk}_{a_{\text{conf}},a_{\text{flavor}}} = t^{ijk}_{a_{\text{conf}}} Q^{ijk}_{a_{\text{flavor}}},$$

```
newtype Lorentz_ON_Struct = Lorentz_ON_Struct B3d.SO3StructLabel
  deriving (Eq, Ord)

instance ThreePointStructure
  (B3d.SO3Struct a a b, ON3PtStruct n)
  Lorentz_ON_Struct
  (B3d.ConformalRep a, ONRep)
  (B3d.ConformalRep b, ONRep) where
  makeStructure (l1,r1) (l2,r2) (l3,r3) (Lorentz_ON_Struct so3Struct) =
    (makeStructure l1 l2 l3 so3Struct, ON3PtStruct r1 r2 r3)
```

As mentioned, O(N) three-point structure is unique, hence, we just need a “wrapper”.

And declare this newtype an instance of **ThreePointStructure**

src/Bounds/Fermions3d/GNY.hs

```
data ChannelType = ChannelType HalfInteger B3d.Parity ONRep
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON, ToJS

data GNY = GNY
  { externalDims :: ExternalDims
  , nGroup       :: Rational
  , spectrum     :: Spectrum ChannelType
  , objective    :: Objective
  , spins        :: [HalfInteger]
  , blockParams  :: B3d.Block3dParams
  } deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON)
```

```
data Spectrum r = Spectrum
  { twistGap       :: Rational
  , deltaGaps      :: Map r Delta
  , deltaIsolateds :: Map r (Set Delta)
  } deriving (Show, Eq, Ord, Generic, Bin
```

src/Bounds/Fermions3d/GNY.hs

$$T^{ijk}_{a_{\text{conf}},a_{\text{flavor}}} = t^{ijk}_{a_{\text{conf}}} Q^{ijk}_{a_{\text{flavor}}},$$

```
newtype Lorentz_ON_Struct = Lorentz_ON_Struct B3d.SO3StructLabel
  deriving (Eq, Ord)

instance ThreePointStructure
  (B3d.SO3Struct a a b, ON3PtStruct n)
  Lorentz_ON_Struct
  (B3d.ConformalRep a, ONRep)
  (B3d.ConformalRep b, ONRep) where
  makeStructure (l1,r1) (l2,r2) (l3,r3) (Lorentz_ON_Struct so3Struct) =
    (makeStructure l1 l2 l3 so3Struct, ON3PtStruct r1 r2 r3)
```

As mentioned, O(N) three-point structure is unique, hence, we just need a “wrapper”.

And declare this newtype an instance of **ThreePointStructure**

src/Bounds/Scalars3d/ONRep.hs

```
import  Bounds.Scalars3d.ONRep  (ON3PtStruct (..),
                                 ON4PtStruct (..),
                                 ONRep (..),
                                 evalONBlock)
```

O(N) representations of a single operator

```
data ONRep = ONSinglet | ONVector | ONSymTensor | ONAntiSymTensor
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON,Bounded,Enum)
```

$$Q_{\text{irrep}}^{(\mathcal{O}_1\mathcal{O}_2),\mathcal{O}}$$

$$Q_{\bullet}^{\bullet,\bullet} = 1,$$
$$Q_{\square}^{i,j} = \delta^{ij},$$
$$Q_{\square}^{ij,\bullet} = \delta^{ij},$$
$$Q_{\square\square}^{ij,(kl)} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} + \delta^{il}\delta^{jk}\right) - \frac{1}{N}\delta^{ij}\delta^{kl},$$
$$Q_{\begin{smallmatrix}\square\\\square\end{smallmatrix}}^{ij,[kl]} = \frac{1}{2}\left(\delta^{ik}\delta^{jl} - \delta^{il}\delta^{jk}\right).$$

```
data ON3PtStruct n = ON3PtStruct ONRep ONRep ONRep
  deriving (Eq, Ord)
```

O(N) three-point structures is unique.

src/Bounds/Fermions3d/GNY.hs

```
data ChannelType = ChannelType HalfInteger B3d.Parity ONRep
  deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON, ToJS

data GNY = GNY
  { externalDims :: ExternalDims
  , nGroup       :: Rational
  , spectrum     :: Spectrum ChannelType
  , objective    :: Objective
  , spins        :: [HalfInteger]
  , blockParams  :: B3d.Block3dParams
  } deriving (Show, Eq, Ord, Generic, Binary, ToJSON, FromJSON)
```

```
data Spectrum r = Spectrum
  { twistGap       :: Rational
  , deltaGaps      :: Map r Delta
  , deltaIsolateds :: Map r (Set Delta)
  } deriving (Show, Eq, Ord, Generic, Bin
```

```
mat
  :: forall j b n s a . (Reifies s ExternalDims, Reifies n Rational, Fractio
  => Channel j b
  ->  Tagged '(n,s) (CrossingMat j 22 b a)

mat (SpinEven_ParityEven_TensorSym iConformalRep) = crossingMatGNY $ toV
  ( opeCoeffIdentical_ Psi iRep (so3 0 l)
  , opeCoeffIdentical_ Psi iRep (so3 1 l)
  )
  where
    iRep = (iConformalRep, ONSymTensor)
    l = B3d.spin iConformalRep
```

OPE	$\mathcal{O} \in (l, P, \mu)$	$\langle \psi^i \psi^j \mathcal{O}^a \rangle$ Structures
$\psi \times \psi$	$(l \in 2\mathbb{Z}$, even, $\mu \in \{\bullet, \square\square\})$	$T_\mu^{ija}\|0,l\rangle$ $T_\mu^{ija}\|1,l\rangle$
	$(l \in 2\mathbb{Z}+1$, even, $\mu = \begin{smallmatrix}\square\\\square\end{smallmatrix})$	$T_\mu^{ija}\|0,l\rangle$ $T_\mu^{ija}\|1,l\rangle$
	$(l \in 2\mathbb{Z}$, odd, $\mu \in \{\bullet, \square\square\})$	$T_\mu^{ija}(\sqrt{l+1}\|1,l+1\rangle - \sqrt{l}\|1,l-1\rangle)$
	$(l \in 2\mathbb{Z}+1$, odd, $\mu \in \{\bullet, \square\square\})$	$T_\mu^{ija}(\sqrt{l+1}\|1,l-1\rangle + \sqrt{l}\|1,l+1\rangle)$
	$(l \in (2\mathbb{Z})_{\geq 2}$, odd, $\mu = \begin{smallmatrix}\square\\\square\end{smallmatrix})$	$T_\mu^{ija}(\sqrt{l+1}\|1,l-1\rangle + \sqrt{l}\|1,l+1\rangle)$
	$(l \in 2\mathbb{Z}+1$, odd, $\mu = \begin{smallmatrix}\square\\\square\end{smallmatrix})$	$T_\mu^{ija}(\sqrt{l+1}\|1,l+1\rangle - \sqrt{l}\|1,l-1\rangle)$

```haskell
-- External
mat ExternalOpChannel =
  pure . mapBlocksFreeVect (evalFlavorCoeff . unzipBlock) $
  crossingMatrixExternal opeCoefficients (crossingEqsGNY @n) [Sig,Psi]
  where
    evalFlavorCoeff (b,f) = evalONBlock f *^ vec (B3d.Block3d b)
    opeCoefficients :: V 1 (OPECoefficientExternal (ExternalOp s) (B3d.SO3Struct Rational Rational Del
    opeCoefficients = toV $ \o1 o2 o3 ->
      FV.mapBasis (makeStructure (rep o1) (rep o2) (rep o3)) $
      case (o1, o2, o3) of
        (Psi,Psi,Sig) -> - so3 1 1
        (Psi,Sig,Psi) -> so3 (1/2) 0
        (Sig,Psi,Psi) -> - so3 (1/2) 1 --Flipped sign
        _             -> 0
```

```haskell
-- External
mat ExternalOpChannel =
  pure . mapBlocksFreeVect (evalFlavorCoeff . unzipBlock) $
  crossingMatrixExternal opeCoefficients (crossingEqsGNY @n) [Sig,Psi]
  where
    evalFlavorCoeff (b,f) = evalONBlock f *^ vec (B3d.Block3d b)
    opeCoefficients :: V 1 (OPECoefficientExternal (ExternalOp s) (B3d.SO3Struct Rational Rational Del
    opeCoefficients = toV $ \o1 o2 o3 ->
      FV.mapBasis (makeStructure (rep o1) (rep o2) (rep o3)) $
      case (o1, o2, o3) of
        (Psi,Psi,Sig) -> - so3 1 1
        (Psi,Sig,Psi) -> so3 (1/2) 0
        (Sig,Psi,Psi) -> - so3 (1/2) 1 --Flipped sign
        _             -> 0
```

src/Bounds/Scalars3d/ONRep.hs

```
data ON4PtStruct n = QPlus | QMinus | Q3 | Unique
  deriving (Eq, Ord)
```

$$\langle\sigma\sigma\sigma\sigma\rangle \quad : \quad Q = 1$$

$$\langle\psi\psi\sigma\sigma\rangle \quad : \quad Q = \delta^{ij}$$

$$\langle\psi\psi\psi\psi\rangle : Q^{+} = \delta^{ij}\delta^{kl} + \delta^{il}\delta^{jk}, \quad \text{crossing even},$$

$$Q^{3} = \delta^{ik}\delta^{jl}, \quad \text{crossing even},$$

$$Q^{-} = \delta^{ij}\delta^{kl} - \delta^{il}\delta^{jk}, \quad \text{crossing odd}.$$

src/Bounds/Fermions3d/GNY.hs

Crossing Equations from <ψψψψ> <σσσσ>

```
crossingEqSSSS
  :: forall s a b .
     FourPointFunctionTerm (ExternalOp s) B3d.Q4Struct b a
  -> V 1 (Taylors 'XT, FreeVect b a)
crossingEqSSSS g = toV
  (xtTaylors xOdd yEven, g Sig Sig Sig Sig (B3d.Q4Struct (0,0,0,0) yEven))
```

j1 = j2 = j3 =j4 = 0

From FourFermions.hs

Add flavor structures

```
crossingEqsWithFlavorSign
  :: forall s a b . (Ord b, Fractional a, Eq a)
  => Sign 'XDir
  -> FourPointFunctionTerm (ExternalOp s) B3d.Q4Struct b a
  -> V 5 (Taylors 'XT, FreeVect b a)
crossingEqsWithFlavorSign fSign g0 = toV
```

Crossing Equations of <ψψσσ>, <ψσψσ>

```
crossingEqsFFSS
  :: forall s a b . (Ord b, Fractional a, Eq a)
  => FourPointFunctionTerm (ExternalOp s) B3d.Q4Struct b a
  -> V 6 (Taylors 'XT, FreeVect b a)
crossingEqsFFSS g0 = toV
  ( ( xtTaylors xOdd  yEven, g s s u u yEven + g u s s u yEven ) -- ^
  , ( xtTaylors xEven yEven, g s s u u yEven - g u s s u yEven ) -- ^
```

$$g^{\sigma\sigma\psi\psi}_{[00\frac{1}{2}\frac{1}{2}]^+}(x,t) + g^{\psi\sigma\sigma\psi}_{[\frac{1}{2}00\frac{1}{2}]^+}(x,t), \text{xy-odd, y-even} \to \text{x-odd, y-even}$$

$$g^{\sigma\sigma\psi\psi}_{[00\frac{1}{2}\frac{1}{2}]^+}(x,t) - g^{\psi\sigma\sigma\psi}_{[\frac{1}{2}00\frac{1}{2}]^+}(x,t), \text{xy-even, y-even} \to \text{x-even, y-even}$$

Add flavor structures

Concatenate All Crossing Equations

```
crossingEqsGNY
  :: forall n s a b . (Ord b, Fractional a, Eq a)
  => FourPointFunctionTerm (ExternalOp s) (B3d.Q4Struct, ON4PtStruct n) b a
  -> V 22 (Taylors 'XT, FreeVect b a)
crossingEqsGNY g =
  FFFF.crossingEqsWithFlavorSign xEven (mapOps (const Psi) (map4pt (,QPlus)  g)) L.++
  FFFF.crossingEqsWithFlavorSign xEven (mapOps (const Psi) (map4pt (,Q3)     g)) L.++
  FFFF.crossingEqsWithFlavorSign xOdd  (mapOps (const Psi) (map4pt (,QMinus) g)) L.++
  crossingEqsFFSS (map4pt (,Unique) g) L.++
  crossingEqSSSS  (map4pt (,Unique) g)
```

```
g qs sgn = g0 Psi Psi Psi Psi (B3d.Q4Struct qs sgn)
g Sig Sig Sig Sig (B3d.Q4Struct (0,0,0,0) yEven))
```

```
map4pt f g o1 o2 o3 o4 s = g o1 o2 o3 o4 (f s)
( , QPlus)  structure = (structure,  QPlus)
```

$$T^{ijkl}_{l_{\text{conf}},a_{\text{flavor}}} = t^{ijkl}_{l_{\text{conf}}} Q^{ijkl}_{l_{\text{flavor}}}.$$

Crossing Equations

```
crossingEqsGNY
  :: forall n s a b . (Ord b, Fractional a, Eq a)
  => FourPointFunctionTerm (ExternalOp s) (B3d.Q4Struct, ON4PtStruct n) b a
  -> V 22 (Taylors 'XT, FreeVect b a)
crossingEqsGNY g =
  FFFF.crossingEqsWithFlavorSign xEven (mapOps (const Psi) (map4pt (,QPlus)  g)) L.++
  FFFF.crossingEqsWithFlavorSign xEven (mapOps (const Psi) (map4pt (,Q3)     g)) L.++
  FFFF.crossingEqsWithFlavorSign xOdd  (mapOps (const Psi) (map4pt (,QMinus) g)) L.++
  crossingEqsFFSS (map4pt (,Unique) g) L.++
  crossingEqSSSS  (map4pt (,Unique) g)
```

```
g qs sgn = g0 Psi Psi Psi Psi (B3d.Q4Struct qs sgn)

mapOps f g o1 o2 o3 o4 s = g (f o1) (f o2) (f o3) (f o4) s

const Psi operator =  Psi
```

```haskell
crossingMatGNY
  :: forall j n s a. (Reifies n Rational, KnownNat j, Fractional a, Eq a)
  => V j (OPECoefficient (ExternalOp s) (B3d.SO3Struct Rational Rational Delta, ON3PtStruct n) a)
  -> Tagged '(n,s) (CrossingMat j 22 B3d.Block3d a)
crossingMatGNY channel =
  pure $ mapBlocksFreeVect (evalFlavorCoeff . unzipBlock) $
  crossingMatrix channel ((crossingEqsGNY @n @s))
  where
    evalFlavorCoeff (b,f) = evalONBlock f *^ vec (B3d.Block3d b)
```

```haskell
evalFlavorCoeff (b,f) = evalONBlock f *^ vec (B3d.Block3d b)
```

Conformal Block Flavor Block

src/Bounds/Scalars3d/ONRep.hs

Evaluate O(N) flavor blocks, the result is a number

```
evalONBlock :: (Reifies n Rational, Fractional a, Eq a) => Block (ON3PtStruct n) (ON4PtStruct n) -> a
evalONBlock (Block (ON3PtStruct r1 r2 r) (ON3PtStruct r4 r3 r') f)
  | r /= r' = 0
  | otherwise = case ((r1, r2, r3, r4), r, f) of
      ((ONVector, ONVector, ONVector, ONVector), _, _) -> FV.coeff f (onVectorBlock r)
      ((ONSinglet, ONSinglet, ONSinglet, ONSinglet), ONSinglet, Unique) -> 1
      ((ONVector,  ONVector,  ONSinglet, ONSinglet), ONSinglet, Unique) -> 1
      ((ONSinglet, ONSinglet, ONVector,  ONVector),  ONSinglet, Unique) -> 1
      ((ONVector,  ONSinglet, ONVector,  ONSinglet), ONVector,  Unique) -> 1
```

Coefficient of f, a ON4PtStruct, in the vector (onVectorBlock r)

```
onVectorBlock
  :: forall n a . (Reifies n Rational, Fractional a, Eq a)
  => ONRep
  -> FreeVect (ON4PtStruct n) a
onVectorBlock r = case r of
  ONSinglet       -> v12_34
  ONSymTensor     -> 1/2 *^ (v13_24 + v23_14) - 1/nGroup *^ v12_34
  ONAntiSymTensor -> 1/2 *^ (v23_14 - v13_24)
  _               -> 0
  where
```

ON4PtStruct's are invariant under crossing

ONRep's are representations of operators

```haskell
bulkConstraints :: GNY -> [BulkConstraint]
bulkConstraints f = do
  parity <- [minBound .. maxBound]
  oNRep <- [minBound .. maxBound]
  l <- case (parity, oNRep) of
    (_, ONVector)                    -> filter (not . HI.isInteger) (spins f)
    (B3d.ParityEven,ONSinglet)       -> filter HI.isEvenInteger (spins f)
    (B3d.ParityOdd, ONSinglet)       -> filter HI.isInteger (spins f)
    (B3d.ParityEven,ONSymTensor)     -> filter HI.isEvenInteger (spins f)
    (B3d.ParityOdd, ONSymTensor)     -> filter HI.isInteger (spins f)
    (B3d.ParityEven,ONAntiSymTensor) -> filter HI.isOddInteger (spins f)
    (B3d.ParityOdd, ONAntiSymTensor) -> filter (\x -> (x/=0) && (HI.isInteger x)) (spins f)
  (delta, range) <- listDeltas (ChannelType l parity oNRep) (spectrum f)
  let iConformalRep = B3d.ConformalRep delta l
  pure $ case (parity, HI.isEvenInteger l, l, oNRep) of
    (B3d.ParityEven, True,  0, ONSinglet)       -> BulkConstraint range $ Scalar_ParityEven_Singlet delta
    (B3d.ParityEven, True,  _, ONSinglet)       -> BulkConstraint range $ SpinEven_ParityEven_Singlet iConformalRep
    (B3d.ParityOdd,  True,  _, ONSinglet)       -> BulkConstraint range $ SpinEven_ParityOdd_Singlet iConformalRep
    (B3d.ParityOdd,  False, _, ONSinglet)       -> BulkConstraint range $ SpinOdd_ParityOdd_Singlet iConformalRep
    (B3d.ParityEven, True,  0, ONSymTensor)     -> BulkConstraint range $ Scalar_ParityEven_TensorSym delta
    (B3d.ParityEven, True,  _, ONSymTensor)     -> BulkConstraint range $ SpinEven_ParityEven_TensorSym iConformalRep
    (B3d.ParityOdd,  True,  _, ONSymTensor)     -> BulkConstraint range $ SpinEven_ParityOdd_TensorSym iConformalRep
    (B3d.ParityOdd,  False, _, ONSymTensor)     -> BulkConstraint range $ SpinOdd_ParityOdd_TensorSym iConformalRep
    (B3d.ParityEven, False, _, ONAntiSymTensor) -> BulkConstraint range $ SpinOdd_ParityEven_TensorAntiSym iConformalRep
    (B3d.ParityOdd,  False, _, ONAntiSymTensor) -> BulkConstraint range $ SpinOdd_ParityOdd_TensorAntiSym iConformalRep
    (B3d.ParityOdd,  True,  _, ONAntiSymTensor) -> BulkConstraint range $ SpinEven_ParityOdd_TensorAntiSym iConformalRep
    (B3d.ParityEven, False, _, ONVector)        -> BulkConstraint range $ ParityEven_Vector iConformalRep
    (B3d.ParityOdd,  False, _, ONVector)        -> BulkConstraint range $ ParityOdd_Vector iConformalRep
    _ -> error $ "Internal representation disallowed: " ++ show((parity,l,oNRep))
```

Practice

1. Take
   - Bounds/Fermions3d/GNY.hs

   and rewrite them into codes that bootstrap
   - (ε, ψ)
   - or all three operators (σ,ε,ψ)